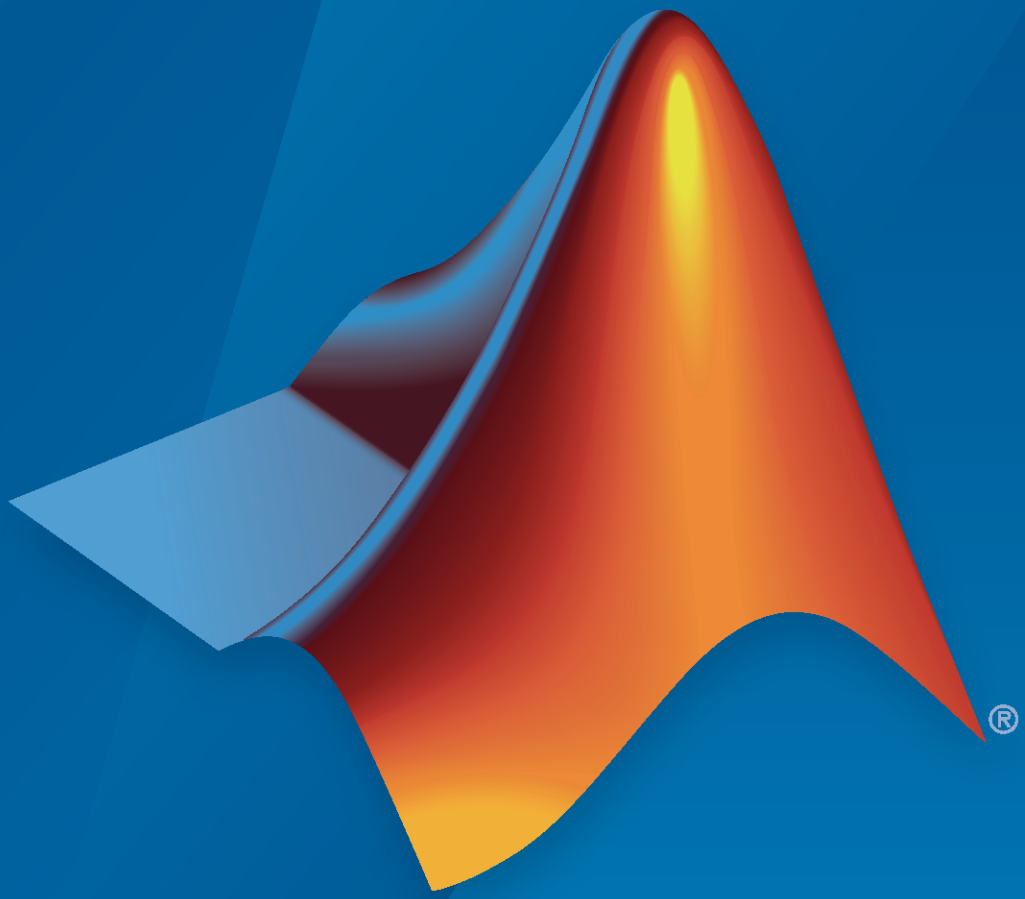
C2000™ Microcontroller Blockset

Reference



MATLAB® & SIMULINK®

R2023a

MathWorks®

How to Contact MathWorks

Latest news: `www.mathworks.com`

Sales and services: `www.mathworks.com/sales_and_services`

User community: `www.mathworks.com/matlabcentral`

Technical support: `www.mathworks.com/support/contact_us`

Phone: 508-647-7000

The MathWorks, Inc.
1 Apple Hill Drive
Natick, MA 01760-2098

C2000™ Microcontroller Blockset Reference

Revision History

March 2023	Online only	New for Version 1.0 (Release R2023a)

Contents

1 Configuration Parameters

2 Blocks

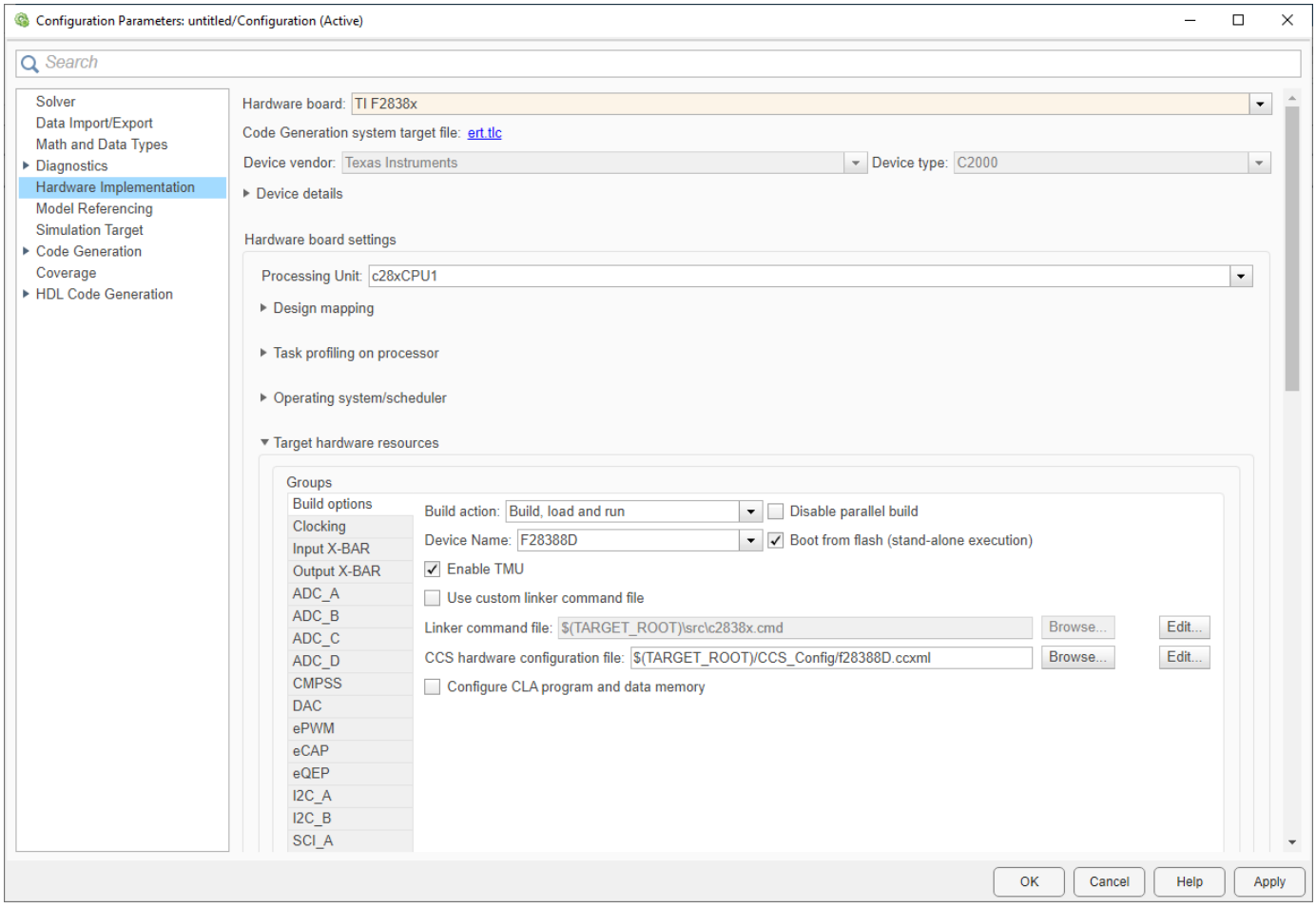
3 Appendix

Configuration Parameters

Model Configuration Parameters for Texas Instruments C2000 Processors

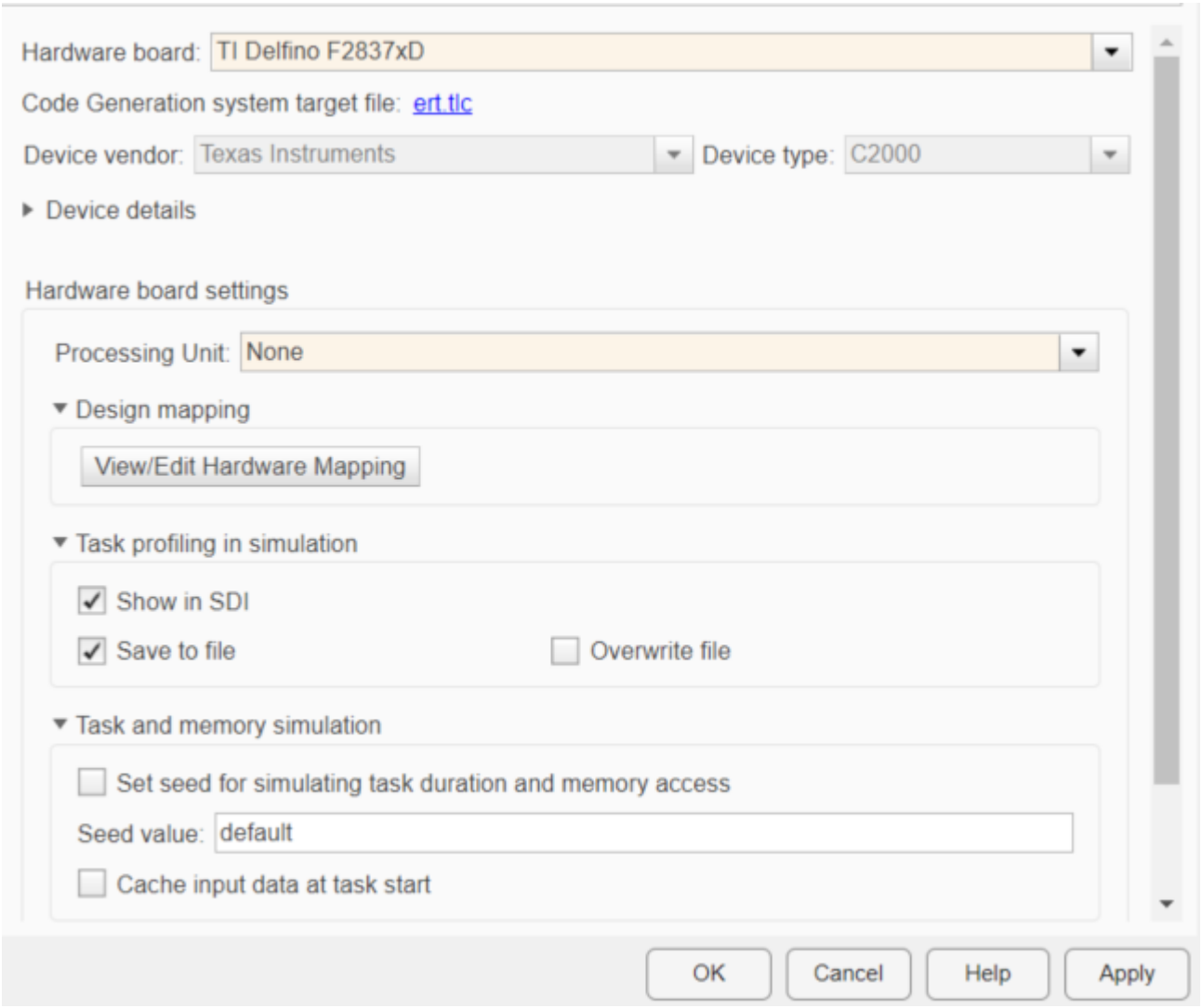
To configure hardware parameters for Texas Instruments C2000 processors:

1. In the Simulink® Editor, select **Modeling > Model Settings**.
2. In the Configuration Parameter dialog box, click **Hardware Implementation**.



3. Set the **Hardware board** parameter to your C2000 processors.
4. The parameter values under **Hardware board settings** are automatically populated to their default values.

   You can optionally adjust these parameters for your particular use case.



Note When selecting the processing unit, choose the central processing unit (CPU) or control law accelerator (CLA) or CortexM4 onto which to deploy the model block in the model. The default values varies based on the processor selected.

5 Click **Apply**.

Note In the **Hardware board** drop-down list, some processors have multiple options. Select the generic option for controlCARDs and custom boards, and select the LaunchPad option for LaunchPads. For example, select **TI Delfino F2837xS** as the generic option, and select **TI Delfino F28377S Launchpad** as the LaunchPad option. Based on your selection, the default values for clock settings, pin selection, and memory mapping change.

Hardware board settings

Parameter	Description	Default Value
“Processing Unit” on page 1-111	Processor or CLA for model block in the MCU model.	`None`

Design Mapping

Parameter	Description	Default Value
“Design Mapping” (SoC Blockset)	Open the hardware mapping tool.	not applicable

Task profiling in simulation

Parameter	Description	Default Value
“Show in SDI” (SoC Blockset)	Show the task execution data collected in simulation in the **Simulation Data Inspector** application.	`on`
“Save to file” (SoC Blockset)	Save the task execution data to a file.	`on`
“Overwrite file” (SoC Blockset)	Overwrite the last task execution data file.	`off`

Task profiling on processor

Parameter	Description	Default Value
“Show in SDI” (SoC Blockset)	Show the task execution data collected on hardware in the **Simulation Data Inspector** application.	`off`
“Save to file” (SoC Blockset)	Save the task execution data to a file.	`off`
“Overwrite file” (SoC Blockset)	Overwrite the last task execution data file.	`off`
“Instrumentation” (SoC Blockset)	Choose to perform code instrumentation or Kernel instrumentation.	`Code`
“Profiling duration” (SoC Blockset)	Choose whether to perform Kernel profiling for an unlimited or limited time duration.	`Unlimited`

Simulation settings

Parameter	Description	Default Value
“Set random number generator seed” (SoC Blockset)	Set the random number generator seed.	`off`
“Seed Value” (SoC Blockset)	Specify the seed value for the simulation of task duration deviation.	`default`
“Cache input data at task start” (SoC Blockset)	Cache the input data at the start of a task.	`off`

Hardware Board Settings

For each hardware board you select, you can configure the board parameters according to your requirements.

Scheduler Options

Parameter	Description	Default Value
Base rate trigger on page 1-168	Set the static priority of the base rate task in the operating system.	`Timer 0`

Build Options

Parameter	Description	Default Value
Build action on page 1-122	Define how Embedded Coder® responds when you build your model.	`Build, load, and run`
Disable parallel build on page 1-122	Select to compile the generated code and driver source codes in parallel order for faster build and deployment speed.	`off`
Device name on page 1-122	Select your device from the selected processor family.	
Enable TMU on page 1-122	Enables support for Trigonometric Math Unit (TMU).	`enabled`
Boot From Flash (stand alone execution) on page 1-122	Specify if the application loads to the flash memory.	`enabled`
Use custom linker command file on page 1-122	Indicates that the custom linker command file must be used during the build action.	`off`
Linker command file on page 1-122	The path to the memory description file required during linking.	
CCS hardware configuration file on page 1-122	The Code Composer Studio™ file required for downloading the application on the hardware.	
Enable DMA to access ePWM Registers instead of CLA on page 1-122	Select to access ePWM Registers	
Enable DMA to peripheral frame 1 (ePWM, HRPWM, eCAP, eQEP, DAC,CMPSS, and SDFM) instead of CLA on page 1-122	Select to enable the DMA to access peripheral frame 1	
Enable DMA to peripheral frame 2 (SPI and McBSP) instead of CLA on page 1-122	Select to enable the DMA to access peripheral frame 2	
Enable FastRTS on page 1-122	Enables use of optimized floating point math functions from C28x FPU fastRTS library	enabled
Remap ePWMs for DMA access (Requires silicon revision A and above) on page 1-122	Select to remap ePWMs registers for DMA access	
Configure CLA program and data memory on page 1-122	Enable this option to configure LSRAM memory for CLA program or data.	`Disabled`

Parameter	Description	Default Value
Maximum LSRAM size for CLA program (in KW) on page 1-122	Select the maximum LSRAM size that is available for CLA program in KiloWords.	
Maximum LSRAM size for CLA data (in KW) on page 1-122	Select the maximum LSRAM size that is available allowed for CLA data in KiloWords.	
Available LSRAM size for CPU (in KW) on page 1-122	Displays the remaining available LSRAM size for CPU in KiloWords.	

Clocking

Parameter	Description	Default Value
Desired CPU Clock in MHz on page 1-125	Specify the desired CPU clock frequency (CLKIN).	
Use internal oscillator on page 1-125	Use the internal zero pin oscillator on the CPU.	`enabled`
Oscillator clock (OSCCLK) frequency in MHz on page 1-125	Oscillator frequency used in the processor.	
Auto set PLL based on OSCCLK and CPU clock on page 1-125	PLL values in PLLCR, DIVSEL, and **Achievable SYSCLKOUT in MHz** are automatically calculated based on the CPU clock entered on the board.	
PLL control register (PLLCR) on page 1-125	If you select **Auto set PLL based on OSCCLK and CPU clock**, the auto-calculated control register value matches the specified CPU clock value, based on the oscillator clock frequency.	
PLL output divider (ODIV) on page 1-125	Calculates SYSCLKOUT = ((OSCCLK×SYSPLLMULT)/ ODIV)/SYSDIVSEL.	
Clock divider (DIVSEL) on page 1-125	If you select **Auto set PLL based on OSCCLK and CPU clock**, the auto-calculated control register value matches the specified CPU clock value, based on the oscillator clock frequency.	
Achievable SYSCLKOUT in MHz = (OSCCLK×PLLCR)/DIVSEL on page 1-125	The auto-calculated feedback value that matches the **Desired C28x CPU clock in MHz** value, based on the values of OSCCLK, PLLCR, and DIVSEL.	
Set the 'Achievable SYSCLKOUT in MHz = (OSCCLK*SYSPLLMULT)/ SYSDIVSEL' value calculated in CPU1 on page 1-125	Available only for CPU2 of dual C28x core processors. Value of this parameter must be same as the value of the parameter **Achievable SYSCLKOUT in MHz = (OSCCLK*PLLCR)/ DIVSEL** (auto calculated).	

Parameter	Description	Default Value
Select the 'Low-Speed Peripheral Clock Prescaler (LSPCLK)' option used in CPU1 on page 1-125	Available only for CPU2 of dual C28x core processors. Value of this parameter must be same as the value of the parameter **Low-Speed Peripheral Clock Prescaler (LSPCLK)** specified in CPU1.	
Low-Speed Peripheral Clock Prescaler (LSPCLK) on page 1-125	Prescaler value used to calculate LSPCLK based on SYSCLKOUT.	
Low-Speed Peripheral Clock (LSPCLK) in MHz on page 1-125	The LSPCLK value calculated using the SYSCLKOUT and LSPCLK Prescaler values.	
High-Speed Peripheral Clock Prescaler (HSPCLK) on page 1-125	Prescaler value used to calculate HSPCLK based on SYSCLKOUT.	
High-Speed Peripheral Clock (HSPCLK) in MHz on page 1-125	The HSPCLK value calculated using the SYSCLKOUT and HSPCLK Prescaler values.	
Analog Subsystem Clock Prescaler (ASYSCLK) on page 1-125	Prescaler value used to calculate ASYSCLK based on SYSCLKOUT.	
Analog Subsystem Clock (ASYSCLK) in MHz on page 1-125	The ASYSCLK value calculated using the SYSCLKOUT and ASYSCLK Prescaler values.	
Connectivity Manager (ARM Cortex-M) clock source on page 1-125	Select the clock source for ARM Cortex-M core.	`System PLL`
Connectivity Manager (ARM Cortex-M) clock divider on page 1-125	Select the divider for the ARM Cortex-M core clock.	`4`
Connectivity Manager (ARM Cortex-M) clock in MHz on page 1-125	The calculated value of the clock frequency (in MHz) supplied to ARM Cortex-M core.	`100`

ADC_x

Parameter	Description	Default Value
Select the CPU core which controls ADC_x module on page 1-119	The CPU core that controls the ADC module.	`Auto`
ADC clock prescaler (ADCCLK) on page 1-119	The ADCCLK divider for the c2802x, c2803x, c2806x, F28M3x, F2807x, or F2837x processor.	`SYSCLKOUT/5.0`
ADC clock frequency in MHz on page 1-119	The clock frequency for ADC, which is auto generated based on the value you select in **ADC clock prescaler (ADCCLK)**.	`40`
ADC overlap of sample and conversion (ADC#NONOVERLAP) on page 1-119	Enable or disable overlap of sample and conversion.	
ADC clock prescaler (ADCLKPS) on page 1-119	The HSPCLK is divided by ADCLKPS (a 4-bit value) as the first step in deriving the core clock speed of the ADC.	`3`
ADC Core clock prescaler (CPS) on page 1-119	After dividing the HSPCLK speed by the **ADC clock prescaler (ADCLKPS)** value, divides the result by 2.	`1`
ADC Module clock (ADCCLK = HSPCLK/ADCLKPS×2)/(CPS +1)) in MHz on page 1-119	The ADC module clock, which indicates the ADC operating clock speed.	
Acquisition window prescaler (ACQ_PS) on page 1-119	Determine the width of the sampling or acquisition period. A higher value indicates a wider sampling period.	`4`
Acquisition window size ((ACQ_PS+1)/ADCCLK) in micro seconds/channel on page 1-119	Determine the duration for which the sampling switch is closed.	
Offset on page 1-119	Specify the offset value.	
Use external reference 2.048V on page 1-119	Allows using a 2.048 V external voltage reference.	
Use external reference on page 1-119	Allows using an external voltage reference.	

Parameter	Description	Default Value
Continuous mode on page 1-119	When the ADC generates an end of conversion (EOC) signal, an ADCINT# interrupt is generated. The interrupt indicates whether the previous interrupt flag has been acknowledged.	
ADC offset correction (OFFSET_TRIM: –256 to 255) on page 1-119	The 280x ADC supports offset correction using a 9-bit value that it adds or subtracts before the results are available in the ADC result registers.	`0`
VREFHI, VREFLO on page 1-119	When you disable the **Use external reference 2.048V** or **External reference** option, the ADC logic uses a fixed 0–3.3 V input range, and VREFHI and VREFLO are disabled. To interpret the ADC input as a ratiometric signal, select the **External reference** option. Then, set values for the high-voltage reference (VREFHI) and the low voltage reference (VREFLO).	
INT pulse control on page 1-119	Set the time when the ADC sets ADCINTFLG ADCINTx relative to the SOC and EOC pulses.	
SOC high priority on page 1-119	Enable SOC high priority mode.	`All in round robin mode`
XINT2SOC external pin on page 1-119	The pin to which the ADC sends the XINT2SOC pulse.	
ADCEXTSOC external pin on page 1-119	The GPIO pin from which ADC receives the ADCEXTSOC pulse.	`GPIO#`
ADCEXTSOC Input X-BAR on page 1-119	Indicates the input of X-BAR for ADC external SOC.	`Input#`

COMP

Parameter	Description	Default Value
Comparator x (COMPx) pin assignment on page 1-129	Assign COMP pin to a GPIO pin.	

DAC

Parameter	Description	Default Value
DACx reference voltage on page 1-128	Select the reference voltage for the DAC channel A, B, or C.	`ADC reference voltage (VREFHIA/VREFHIB)`
DACx synchronization signal on page 1-128	Select the synchronization signal to load the value from the writable shadow register into the active register.	`SYSCLK`

eCAN_x

Parameter	Description	Default Value
CAN module clock frequency (= SYSCLKOUT) in MHz on page 1-136	The clock for the enhanced CAN module.	`200`
CAN module clock frequency (=SYSCLKOUT/2) in MHz on page 1-136	The clock for the enhanced CAN module.	
Baud rate prescaler (BRP: 2 to 256)/Baud rate prescaler (BRP: 1 to 1024) on page 1-136	Scale the bit rate using this value.	`20`
Time segment 1 (TSEG1) on page 1-136	Set the value of time segment 1. This value, with **TSEG2** and **Baud rate prescaler**, determines the length of a bit on the eCAN bus.	
Time segment 2 (TSEG2) on page 1-136	Set the value of time segment 2. This value, with **TSEG1** and **Baud rate prescaler**, determines the length of a bit on the eCAN bus.	
Baud rate (CAN Module Clock/BRP/(TSEG1 + TSEG2 +1)) in bits/sec on page 1-136	CAN module communication speed represented in bits/ second.	
SBG on page 1-136	Set the message resynchronization triggering.	
SJW on page 1-136	Set the synchronization jump width, which determines how many units of TQ a bit can be shortened or lengthened by when resynchronizing.	
SAM on page 1-136	Number of samples used by the CAN module to determine the CAN bus level.	
Enhanced CAN Mode on page 1-136	Enable time stamping and usage of Mailbox Numbers 16 through 31 in the C2000 eCAN blocks.	
Self test mode on page 1-136	If you set this parameter to `True`, the eCAN module goes to loopback mode. The loopback mode sends a dummy acknowledge message back.	`False`
Pin assignment (Tx) on page 1-136	Assign the CAN transmit pin to use with the eCAN_B module.	
Pin assignment (Rx) on page 1-136	Assign the CAN receive pin to use with the eCAN_B module.	

eCAP

Parameter	Description	Default Value
ECAPx capture pin assignment on page 1-138	Indicates the GPIO pin used for eCAP in capture mode.	GPIO#
ECAPx Input X-BAR on page 1-138	Select input X-BAR for ECAP	INPUT#
ECAPx APWM pin assignment on page 1-138	The GPIO pin to which output of the eCAP in APWM mode is sent.	GPIO#
Output X-BAR on page 1-138	Indicates which Output X-BAR is used for the selected ECAP# APWM pin assignment parameter.	#
eCAPxSYNCIN source selection on page 1-138	Indicates the SYNC source select register for the ePWM SYNCOUT, eCAP SYNCOUT, INPUTXBAR and EtherCATSYNC.	The default eCAPxSYNCIN source selection value varies based on the processor selected.

ePWM

Parameter	Description	Default Value
EPWM clock divider (EPWMCLKDIV) on page 1-145	Select the ePWM clock divider.	`SYSCLKOUT/1`
Select the 'EPWM clock divider (EPWMCLKDIV)' option used for CPU1 on page 1-145	Available only for CPU2 of dual C28x core processors. Its value must be the same as the value of the parameter EPWM clock divider (EPWMCLKDIV) selected in CPU1.	
TZx Input X-BAR on page 1-145	Indicates the trip-zone input X-BAR.	`INPUT#`
TZx pin assignment on page 1-145	Indicates the GPIO pin to the trip-zone input x (TZx).	`None`
TRIP# MUX select on page 1-145	Select the TRIP Multiplexer(MUX).	`Disable all`
TRIP# MUX (MUX 0->31) on page 1-145	Indicates the inputs selected for each MUX so far.	`XXXXXXXXXXX`
Select MUX input on page 1-145	Select the input to the Multiplexer selected in `TRIP# MUX select`.	`X:Disable`
Reset TRIP# MUX on page 1-145	Option to reset the MUX inputs selected.	
Invert TRIP output on page 1-145	Inverts the TRIP output signal.	`off`
SYNCI Input X-BAR on page 1-145	Indicates the SYNCI input X-BAR.	`INPUT#`
SYNCI pin assignment on page 1-145	Indicates the GPIO pin used for the ePWM external sync pulse input (SYNCI).	`None`
SYNCO pin assignment on page 1-145	Assign the ePWM external sync pulse output (SYNCO) to a GPIO pin.	
EXTSYNCOUT source selection on page 1-145	Select the external SYNCOUT source for ePWM	`Default value varies based on the processor selected`
ePWMxSYNCIN source selection on page 1-145	Select the EPWMxSYNCIN Source Select Register (synchronization input pulse) for the ePWM	`Default value varies based on the processor selected`
PWM#x pin assignment on page 1-145	Assign the GPIO pin to the PWM#x module.	`GPIO#`
GPTRIP#SEL pin assignment(GPIO0~63) on page 1-145	Assign the ePWM trip-zone input to a GPIO pin.	

Parameter	Description	Default Value
PWM1SYNCI/ GPTRIP6SEL pin assignment on page 1-145	Assign the ePWM sync pulse input (SYNCI) to a GPIO pin.	
DCxHTRIPSEL (Enter Hex value between 0 and 0x6FFF) on page 1-145	Assign the **Digital Compare A** high trip input to a GPIO pin.	
DCxLTRIPSEL (Enter Hex value between 0 and 0x6FFF) on page 1-145	Assign the **Digital Compare A** low trip input to a GPIO pin.	

I2C

Parameter	Description	Default Value
Mode on page 1-159	Configure the I2C module as `Master` or `Slave`.	
Addressing format on page 1-159	In `Slave` mode, determines the addressing format of the I2C master and sets the I2C module to the same mode.	
Own address register on page 1-159	In `Slave` mode, enter the 7-bit (0–127) or 10-bit (0–1023) address that the I2C module uses.	
Bit count on page 1-159	In `Slave` mode, sets the number of bits in each data byte the I2C module transmits and receives.	
Module clock prescaler (IPSC: 0 to 255) on page 1-159	In `Master` mode, enter a value in the range 0–255, inclusive, to configure the model clock frequency.	
I2C Module clock frequency (SYSCLKOUT / (IPSC+1)) in Hz on page 1-159	Display the frequency the I2C module uses internally. To set this value, change the **Module clock prescaler**.	
I2C Master clock frequency (Module Clock Freq/(ICCL +ICCH+10)) in Hz on page 1-159	Display the master clock frequency.	
Master clock Low-time divider (ICCL: 1 to 65535) on page 1-159	In `Master` mode, determines the duration of the low state of the SCL on the I2C bus.	
Master clock High-time divider (ICCH: 1 to 65535) on page 1-159	In `Master` mode, determines the duration of the high state of the SCL on the I2C bus.	
Enable loopback on page 1-159	In `Master` mode, enables or disables digital loopback mode.	
SDA pin assignment on page 1-159	Select a GPIO pin as an I2C data bidirectional port.	
SCL pin assignment on page 1-159	Select a GPIO pin as an I2C clock bidirectional port.	
Enable Tx interrupt on page 1-159	This parameter corresponds to bit 5 (TXFFIENA) of the I2C Transmit FIFO Register (I2CFFTX).	

Parameter	Description	Default Value
Tx FIFO interrupt level on page 1-159	This parameter corresponds to bits 4–0 (TXFFIL4-0) of the I2C transmit FIFO register (I2CFFTX).	
Enable Rx interrupt on page 1-159	This parameter corresponds to bit 5 (RXFFIENA) of the I2C receive FIFO register (I2CFFRX).	
Rx FIFO interrupt level on page 1-159	This parameter corresponds to bit 4–0 (RXFFIL4-0) of the I2C receive FIFO register (I2CFFRX).	
Enable system interrupt on page 1-159	Select this parameter to configure the five basic I2C interrupt request parameters in the interrupt enable register (I2CIER).	
Enable AAS interrupt on page 1-159	Enable the addressed-as-slave interrupt bit.	
Enable SCD interrupt on page 1-159	Enable the stop condition detected interrupt bit.	
Enable ARDY interrupt on page 1-159	Enable the register-access-ready interrupt bit.	
Enable NACK interrupt on page 1-159	Enable the no acknowledgment interrupt bit.	
Enable AL interrupt on page 1-159	Enable the arbitration-lost interrupt bit.	

SCI_x

Parameter	Description	Default Value
Enable loopback on page 1-169	Enable the loopback function for self-test and diagnostics.	
Suspension mode on page 1-169	The type of suspension to use while debugging your program with Code Composer Studio.	
Number of stop bits on page 1-169	Specify the number of stop bits transmitted.	
Parity mode on page 1-169	The type of parity to use.	
Character length bits on page 1-169	Length in bits of each transmitted or received character.	8
Desired baud rate in bits/sec on page 1-169	Specify the desired baud rate.	
Baud rate prescaler (BRR = (SCIHBAUD << 8) | SCILBAUD)) on page 1-169	Scale the SCI baud rate using this value.	
Closest achievable baud rate (LSPCLK/(BRR+1)/8) in bits/sec on page 1-169	The closest achievable baud rate, calculated based on LSPCLK and BRR.	
Communication mode on page 1-169	Select the mode for transmitting and receiving data.	
Post transmit FIFO interrupt when data is transmitted on page 1-169	Posts interrupt when available data in transmit FIFO is less than or equal to interrupt level.	`off`
Transmit FIFO interrupt level on page 1-169	Level for triggering SCI transmit interrupt.	1
Post receive FIFO interrupt when data is received on page 1-169	Posts interrupt when available data in receive FIFO is greater than or equal to interrupt level.	`off`
Receive FIFO interrupt level on page 1-169	Level for triggering SCI receive interrupt.	1
Data byte order on page 1-169	Select an option to match the endianness of the data being moved.	
Pin assignment (Tx) on page 1-169	Assign the SCI transmit pin to use with the SCI module.	
Pin assignment (Rx) on page 1-169	Assign the SCI receive pin to use with the SCI module.	

SPI_x

Parameter	Description	Default Value
Mode on page 1-172	Set to `Master` or `Slave`.	
Desired baud rate in bits/sec on page 1-172	Specify the desired baud rate.	
Baud rate factor (SPIBRR: between 3 and 127) on page 1-172	The value used to calculate the baud rate.	
Closest achievable baud rate (LSPCLK/(SPIBRR+1)) in bits/sec on page 1-172	The closest achievable baud rate, calculated based on LSPCLK and SPIBRR.	
Suspension mode on page 1-172	The type of suspension to use while debugging your program with Code Composer Studio.	
Enable loopback on page 1-172	Enable the loopback function for self-test and diagnostics.	`off`
Enable 3-wire mode on page 1-172	Enables SPI communication over three pins instead of the normal four pins.	`off`
Enable Tx interrupt on page 1-172	Enable SPI transmit interrupt operation.	`off`
FIFO interrupt level (Tx) on page 1-172	Set level for transmit FIFO interrupt.	`0`
Enable Rx interrupt on page 1-172	Enable SPI receive interrupt operation.	`off`
Enable high speed mode on page 1-172	Enable high speed SPI mode for supported pins.	
FIFO interrupt level (Rx) on page 1-172	Set level for receive FIFO interrupt.	
FIFO transmit delay on page 1-172	FIFO transmit delay (in processor clock cycles) to pause between data transmissions.	
Peripheral in controller out pin assignment on page 1-172	Assign the SPI (SIMO) to a GPIO pin.	
Peripheral out controller in pin assignment on page 1-172	Assign the SPI value (SOMI) to a GPIO pin.	
CLK pin assignment on page 1-172	Assign the CLK pin to a GPIO pin.	
STE pin assignment on page 1-172	Assign the SPI value (STE) to a GPIO pin.	

eQEP

Parameter	Description	Default Value
EQEP#x pin assignment on page 1-154	Assign eQEP pin to a GPIO pin.	

Watchdog

Parameter	Description	Default Value
Enable watchdog on page 1-174	Enable the watchdog timer module.	
Counter clock on page 1-174	Set the watchdog timer period relative to OSCCLK/512.	
Timer period ((1/Counter clock) ×256) in seconds on page 1-174	Display the timer period in seconds. This value automatically updates when you change the **Counter clock** parameter.	
Time out event on page 1-174	Configure the watchdog to reset the processor or generate an interrupt when the software fails to reset the watchdog counter.	

GPIO

Parameter	Description	Default Value
GPIO# on page 1-155	Use the GPIO pins for digital input or output by connecting to one of the three peripheral I/O ports.	

DMA_ch#

Parameter	Description	Default Value
Enable DMA channel on page 1-130	Enable to edit the configuration of a specific DMA channel.	
Data size on page 1-130	Select the size of the data bit transfer.	
Interrupt source on page 1-130	Select the peripheral interrupt that triggers a DMA burst for the specified channel.	
SRC wrap on page 1-130	Specify the number of bursts before returning the current source address pointer to the **Source Begin Address** value.	
DST wrap on page 1-130	Specify the number of bursts before returning the current destination address pointer to the **Destination Begin Address** value.	
SRC Begin address on page 1-130	Set the starting address for the current source address pointer.	
DST Begin address on page 1-130	Set the starting address for the current destination address pointer.	
Burst on page 1-130	Specify the number of 16-bit words in a burst, from 1 to 32.	
Transfer on page 1-130	Specify the number of bursts in a transfer, from 1 to 65536.	
SRC Burst step on page 1-130	Increment or decrement the current address pointer by this number of 16-bit words before the next burst.	
DST Burst step on page 1-130	Increment or decrement the current address pointer by this number of 16-bit words before the next burst.	
SRC Transfer step on page 1-130	Increment or decrement the current address pointer by this number of 16-bit words before the next transfer.	
DST Transfer step on page 1-130	Increment or decrement the current address pointer by this number of 16-bit words before the next transfer.	

Parameter	Description	Default Value
SRC Wrap step on page 1-130	Increment or decrement the SRC_BEG_ADDR address pointer by this number of 16-bit words when a wrap event occurs.	
DST Wrap step on page 1-130	Increment or decrement the DST_BEG_ADDR address pointer by this number of 16-bit words when a wrap event occurs.	
Generate interrupt on page 1-130	Enable this parameter to have the DMA channel send an interrupt to the CPU through the Peripheral Interrupt Expansion (PIE) at the beginning or end of a data transfer.	
Enable one shot mode on page 1-130	Enable this parameter to have the DMA channel complete an entire *transfer* in response to an interrupt event trigger.	
Sync enable on page 1-130	Enable this parameter to reset the DMA wrap counter when the **Interrupt source** is set to SEQ1INT and sends the ADCSYNC signal to the DMA wrap counter.	
Enable continuous mode on page 1-130	Select this parameter to leave the DMA channel enabled upon completing a transfer. The channel waits for the next interrupt event trigger.	
Enable DST sync mode on page 1-130	Enabling this parameter resets the destination wrap counter (DST_WRAP_COUNT) when **Sync enable** is enabled and the DMA module receives the SEQ1INT interrupt/ADCSYNC signal.	
Set channel 1 to highest priority on page 1-130	Enable this option when DMA channel 1 is configured to handle high-bandwidth data, such as ADC data, and the other DMA channels are configured to handle lower-priority data.	

Parameter	Description	Default Value
Enable overflow interrupt on page 1-130	Enable this parameter to have the DMA channel send an interrupt to the CPU through the PIE if the DMA module receives a peripheral interrupt while a previous interrupt from the same peripheral is waiting to be serviced.	

EMIF#

Parameter	Description	Default Value
EMIF clock divider (EMIF1CLKDIV) on page 1-140	Clock divider for clock frequency generation.	`SYSCLKOUT/2`
Enable CS0 for Synchronous memory on page 1-140	Chip select (CS0) to interface with the SDRAM.	`off`
Enable CS# for Asynchronous memory on page 1-140	Chip select (CS2/CS3/CS4) to interface with the asynchronous RAM.	`off`
SDRAM Column address bits on page 1-140	Value of the column address bits or the required page size of the connected SDRAM.	8
Number of internal SDRAM banks on page 1-140	Number of memory banks inside the connected SDRAM.	3
SDRAM data bus width in bits on page 1-140	Data bus width of the connected SDRAM.	16
Refresh to active command delay cycles (T_RFC) on page 1-140	Minimum number of EM#CLK cycles from the refresh or load mode command to the refresh or activate command in the connected SDRAM.	3
Row precharge to Active command delay cycles (T_RP) on page 1-140	Minimum number of EM#CLK cycles required from the row precharge command to the activate or refresh command in the connected SDRAM.	1
Active to read or write command delay cycles (T_RCD) on page 1-140	Minimum number of EM#CLK cycles from the activate command to the read or write command in the connected SDRAM.	2
Last write to row precharge command delay cycles (T_WR) on page 1-140	Minimum number of EM#CLK cycles from the last write transfer or last data in command to the row precharge command in the connected SDRAM.	1
Active to precharge command delay cycles (T_RAS) on page 1-140	Minimum number of EM#CLK cycles from the activate command to the row precharge command in the connected SDRAM.	4

Parameter	Description	Default Value
Active to active command delay cycles (T_RC) on page 1-140	Minimum number of EM#CLK cycles from an activate command to the next activate command in the same bank in the connected SDRAM.	6
Active one bank to active another bank command delay cycles (T_RRD) on page 1-140	Minimum number of EM#CLK cycles from an activate command in one bank to an activate command in a different bank in the connected SDRAM.	1
Self-refresh exit to other command delay cycles (T_XSR) on page 1-140	Minimum number of EM#CLK cycles from the self refresh exit command to any other command in the connected SDRAM.	7
SDRAM refresh period (tRefreshPeriod) in ms on page 1-140	Defines the rate at which the connected SDRAM refreshes.	64
SDRAM CAS Latency on page 1-140	CAS latency required to access the connected SDRAM.	3
Asynchronous mode on page 1-140	Asynchronous mode for the connected asynchronous memory.	Normal
Asynchronous data bus width in bits on page 1-140	Data bus width of the connected asynchronous memory.	16
Read strobe setup cycles (R_SETUP) on page 1-140	Number of EM#CLK cycles from the EMIF chip select to the pin enable for asynchronous memory assert.	15
Read strobe duration cycles (R_STROBE) on page 1-140	Number of EM#CLK cycles during which the pin enable for the asynchronous memory is held active.	64
Read strobe hold cycles (R_HOLD) on page 1-140	Number of EM#CLK cycles during which the EMIF chip select is held active after pin enable for the asynchronous memory is deasserted.	7
Write strobe setup cycles (W_SETUP) on page 1-140	Number of EM#CLK cycles from the EMIF chip select to the write enable for the asynchronous memory assert.	15
Write strobe duration cycles (W_STROBE) on page 1-140	Number of EM#CLK cycles during which the write enable for the asynchronous memory is held active.	63

Parameter	Description	Default Value
Write strobe hold cycles (W_HOLD) on page 1-140	Number of EM#CLK cycles during which the EMIF chip select is held active after write enable for the asynchronous memory is deasserted.	7
Turn around cycles (TA) on page 1-140	Number of EM#CLK cycles between the end of one asynchronous memory access and the start of another asynchronous memory access.	3
Enable extended wait mode on page 1-140	Enable the extended wait option for the asynchronous memory.	`off`
Maximum extended wait cycles for Asynchronous memory (MAX_EXT_WAIT) [0–255] on page 1-140	EMIF waits for *(MAX_EXT_WAIT +1) * 16* clock cycles before the asynchronous cycle is terminated.	128
Pin polarity of extended wait on page 1-140	Make EMIF wait if the pin is low or high.	`High`
Enable wait rise interrupt on page 1-140	Get an interrupt based on the detection of a rising edge on the EM#WAIT pin.	`off`
Enable timeout interrupt on page 1-140	Get an interrupt when the EM#WAIT pin does not become inactive within the number of cycles defined in **Maximum extended wait cycles for Asynchronous memory (MAX_EXT_WAIT) [0–255]**.	`off`
Enable line trap interrupt on page 1-140	Get an interrupt when there is an invalid cache line size or illegal memory access.	`off`

LIN

Parameter	Description	Default Value
LIN Module clock frequency (LM_CLK = SYSCLKOUT/2) in MHz on page 1-163	Display the frequency of the LIN module clock in MHz.	
Enable loopback on page 1-163	Enable LIN loopback testing.	
Suspension mode on page 1-163	Use this option to configure how the LIN state machine behaves while you debug the program using an emulator.	`Free_run`
Parity mode on page 1-163	Use this option to configure parity checking.	`None`
Frame length bytes on page 1-163	Set the number of data bytes in the response field, from 1–8 bytes.	`8`
Baud rate prescaler (P: 0-16777215) on page 1-163	To set the LIN baud manually, enter a prescaler value from 0–16777215.	`15`
Baud rate fractional divider (M: 0–15) on page 1-163	To set the LIN baud manually, enter a fractional divider value from 0–15.	`4`
Baud rate (FLINCLK = LM_CLK/(16×(P+1+M/16)) in bits/sec on page 1-163	Display the baud rate.	
Communication mode on page 1-163	Enable or disable the LIN module from using the ID-field bits ID4 and ID5 for length control.	`ID4 and ID5 not used for length control`
Data byte order on page 1-163	Set the endianness of the LIN message data bytes.	`Little_Endian`
Data swap width on page 1-163	Set the width for data swap.	
Pin assignment (Tx) on page 1-163	Map the LINTX output to a specific GPIO pin.	`GPIO9`
Pin assignment (Rx) on page 1-163	Map the LINRX input to a specific GPIO pin.	`GPIO11`
LIN mode on page 1-163	Set the LIN module as a master or a slave.	`Slave`
ID filtering on page 1-163	Select the type of mask filtering comparison the LIN module performs.	`ID slave task byte`
ID byte on page 1-163	If you set **ID filtering** as `ID byte`, use this option to set the ID BYTE, also known as the “LIN mode message ID”.	`0x3A`

Parameter	Description	Default Value
ID slave task byte on page 1-163	If you set **ID filtering** to `ID slave task byte`, use this option to set the ID-SlaveTask BYTE.	`0x30`
Checksum type on page 1-163	Select the checksum type.	`Classic`
Enable multibuffer mode on page 1-163	When you select this check box, the LIN node uses transmit and receive buffers instead of just one register.	Selected
Enable baud rate adapt mode on page 1-163	This option is displayed when you set **LIN mode** to `Slave`.	Not selected
Inconsistent synch field error interrupt on page 1-163	If you enable this option, the slave node generates interrupts when it detects irregularities in the synch field.	`Disabled`
No response error interrupt on page 1-163	If you enable this option, the LIN module generates an interrupt if it does not receive a complete response from the master node within a timeout period.	`Disabled`
Timeout after 3 wakeup signals interrupt on page 1-163	When enabled, the slave node generates an interrupt when it sends three wakeup signals to the master node and does not receive a header in response.	`Disabled`
Timeout after wakeup signal interrupt on page 1-163	When enabled, the slave node generates an interrupt when it sends a wakeup signal to the master node and does not receive a header in response.	`Disabled`
Timeout interrupt on page 1-163	When enabled, the slave node generates an interrupt after 4 seconds of inactivity on the LIN bus.	`Disabled`
Wakeup interrupt on page 1-163	The LIN slave mode generates a wakeup interrupt based on a request or condition.	`Disabled`

External Interrupt

Parameter	Description	Default Value
XINT# Input X-BAR on page 1-179	Indicates the input X-BAR for external interrupt.	`Input#`
XINT# GPIO on page 1-179	The GPIO pin for external interrupt.	`0`
XINT# Polarity on page 1-179	Set the polarity for external interrupt.	`Falling edge`

External Mode

Parameter	Description	Default Value
Communication interface on page 1-180	Select the type of communication interface to run your model in external mode.	`XCP on Serial`
SCI module on page 1-180	Select the serial communication interface module.	`SCI_A`
Serial port in MATLAB preferences on page 1-180	Select the COM port used by the target hardware.	
Host Interface on page 1-180	Select the interface through which the host computer communicates to target hardware for signal monitoring and parameter tuning.	`Third party calibration tools`
CAN module on page 1-180	Select the CAN module to be used with external mode	`eCAN_A`
CAN ID Command on page 1-180	Enter the CAN ID Command for the CAN module.	`2`
CAN ID Response on page 1-180	Enter the CAN ID Response for the CAN module.	`3`
CAN vendor on page 1-180	Enter the CAN vendor for the CAN module.	`<empty>`
CAN device on page 1-180	Enter the device for the CAN module.	`<empty>`
CAN channel number on page 1-180	Enter the CAN channel number for the CAN module.	`<empty>`
Extended CAN ID on page 1-180	Select to use extended ID.	`off`
Rx mailbox number on page 1-180	Enter the Rx mailbox number for the CAN module.	`0`
Tx mailbox number on page 1-180	Enter the Tx mailbox number for the CAN module.	`1`
Verbose on page 1-180	Select to view the external mode execution progress and updates in the Diagnostic Viewer or in the MATLAB command window.	`off`
Set logging buffer size automatically on page 1-180	Select to automatically set the number of bytes to preallocate for the buffer in the hardware during simulation.	`on`
Maximum number of contiguous samples on page 1-180	Specify a value for maximum number of contiguous samples parameter.	`8`

Parameter	Description	Default Value
Use a dedicated timer to improve time stamp accuracy on page 1-180	Select to log data inside ISR at ISR trigger rate	`off`

Execution Profiling

Parameter	Description	Default Value
Number of profiling samples to collect on page 1-178	Enter the number of profiling samples to collect.	

SD Card Logging

Parameter	Description	Default Value
Enable MAT-file logging on SD card on page 1-184	Enables the MAT-file logging for SD card.	`off`
SPI module on page 1-184	Select the desired interface on which the SD card is connected to hardware board.	
SPI baud rate on page 1-184	Select the desired option for the SPI interface used by the SD card.	`Maximum achievable supported by the inserted SD Card`

CMPSS

Parameter	Description	Default Value
Configure CMPSS# on page 1-175	Configure the comparator subsystem (CMPSS).	`off`
Configure COMP# on page 1-175	Configure the COMPH or COMPL module.	`off`
Reload condition for RAMP reference value (RAMPLOADSEL) on page 1-175	Reload condition for RAMP reference value.	`Immediate (RAMPMAXREFA)`
Invert comparator output on page 1-175	Invert comparator output.	`off`
Enable latch clear by EPWMSYNCPER event on page 1-175	Enable latch clear by EPWMSYNCPER event.	`off`
Configure digital filter on page 1-175	Configure the digital filter for COMP#.	`off`
Sample clock prescale [0 to 1023] on page 1-175	Set the sample clock prescale for digital filter of COMP#.	`0`
Sample window size on page 1-175	Set the sample window size for digital filter of COMP#.	`0`
Threshold sample size on page 1-175	Set the threshold sample window size for digital filter of COMP#.	`0`
Comparator output type for EPWM X-BAR (CTRIP#SEL) on page 1-175	Select the comparator output type source for COMP#.	`Asynchronous output (ASYNCH)`
Comparator output type for OUTPUT X-BAR (CTRIPOUT#SEL) on page 1-175	Select the comparator output type source for COMP#.	`Asynchronous output (ASYNCH)`
DAC reference voltage on page 1-175	Select the DAC reference voltage for CMPSS.	`Internal reference voltage (VDDA)`
Reload condition for DAC value (SWLOADSEL) on page 1-175	Select the reload condition for DAC value.	`System clock (SYSCLK)`
EPWM peripheral synchronization event on page 1-175	Select the EPWM peripheral synchronization event.	`EPWM1SYNCPER`
EPWM blank window event on page 1-175	Select the EPWM for blanking window	`EPWM1BLANK`
Comparator hysteresis value on page 1-175	Set the amount of hysteresis on the comparator inputs.	`0`

SDFM#

Parameter	Description	Default Value
Configure filter# on page 1-185	Configure the filter channel for the SDFM module.	`off`
Data pin assignment (SD#_D#) on page 1-185	Select the data pin for the GPIO configuration.	`GPIO#`
Clock pin assignment (SD#_C#) on page 1-185	Select the clock pin for the GPIO configuration.	`GPIO#`
Modulator clock mode on page 1-185	Select the modulator clock mode.	`Same as the modulator data rate (MOD_0)`
Comparator filter type on page 1-185	Select the comparator filter type.	`Sinc1`
Comparator over sample ratio (COSR) [0-31] on page 1-185	Specify the comparator OSR value.	`0`
Comparator higher threshold (HLT#) [0-32767] on page 1-185	Specify the comparator higher threshold value to detect an over-value condition.	`0`
Comparator lower threshold (LLT#) [0-32767] on page 1-185	Specify the comparator lower threshold value to detect an under-value condition.	`0`
Comparator higher threshold (HLTZ) [0-32767] on page 1-185	Specify the comparator higher threshold value to detect over-value condition.	`0`
Data filter type on page 1-185	Select the data filter type.	`Sinc1`
Data over sampling ratio (DOSR) [0-255] on page 1-185	Specify the data OSR value.	`0`
Data filter FIFO depth on page 1-185	Specify the FIFO value for the data filter.	`0`
Enable data filter reset by PWM on page 1-185	Enable to reset the data filter by external PWM compare output.	`off`
ePWM module on page 1-185	Select the ePWM module for synchronization.	`PWM#SOCA`
Comparator event # (CEVT#) interrupt on page 1-185	Select the comparator event (CEVT#) interrupt.	`Disable`
Enable high level threshold crossing output (HLTZ) on page 1-185	Enable threshold crossing event detection.	`off`
Enable modulator clock failure interrupt on page 1-185	Enable the interrupt for modulator clock failure.	`off`
Enable data filter acknowledge interrupt on page 1-185	Enable the interrupt for new data acknowledgment.	`off`
Enable comparator higher threshold (HLT) on page 1-185	Enable to detect an over-value condition.	`off`

Parameter	Description	Default Value
Enable comparator lower threshold (LLT) on page 1-185	Enable to detect an under-value condition.	`off`
Synchronize SD Data with PLLCLK on page 1-185	Select to synchronize the data input to a filter with the PLL clock.	`off`
Synchronize SD Clock with PLLCLK on page 1-185	Select to synchronize the clock input to a filter with the PLL clock.	`off`

PIL

Parameter	Description	Default Value
Communication interface on page 1-183	Select the type of communication interface to run your model.	`Serial`
SCI module on page 1-183	Select the serial communication interface module.	`SCI_A`
Serial port in MATLAB preferences on page 1-183	Select the COM port used by the target hardware.	

Analog subsystem

Parameter	Description	Default Value
External references for VREFHIx on page 1-70	Allows using an external voltage reference.	`off`
VREFHIx on page 1-70	High voltage reference.	`3.3`

Overrun detection

Parameter	Description	Default Value
Enable overrun detection on page 1-71	Enable to notify when task overrun occurs.	`off`
Set/Clear/toggle GPIO on page 1-71	Enable to select GPIO action.	`on`
Digital output pin to set an overrun on page 1-71	Specify the GPIO number of a digital output.	`34`
GPIO set mode on page 1-71	Select the GPIO mode.	`Set`
Additional notification option on page 1-71	Select the additional option to notify when task overrun occurs.	`None`
PIE number on page 1-71	Specify the PIE number for the interrupt to trigger on overrun.	`1`
CPU number on page 1-71	Specify the CPU number for the interrupt to trigger on overrun.	`1`
Name of the function on page 1-71	Specify the name of the C function to call on overrun.	`C2000_OverunFunction`

INPUT X-BAR

Parameter	Description	Default Value
INPUT# pin assignment on page 1-73	Specify the GPIO pin for input X-BAR.	`None`

OUTPUT X-BAR

Parameter	Description	Default Value
OUTPUT# MUX select on page 1-76	Select the output multiplexer(MUX).	`Disable all`
Select MUX input on page 1-76	Select the input to the MUX selected for `OUTPUT# MUX select`.	`X:Disable`
OUTPUT# MUX (MUX 0->31) on page 1-76	Indicates the input signal selected for each output MUX.	`XXXXXXXXXXXXXXXXXXXXXXXX`
Reset OUTPUT# MUX on page 1-76	Option to reset the MUX inputs selected.	
OUTPUT# pin assignment on page 1-76	Select the GPIO pin for the output X-BAR MUX signals.	`GPIO#`
Enable OUTPUT# latch on page 1-76	Enables the output latch to drive the respective output X-BAR.	`off`
Invert OUTPUT# on page 1-76	Inverts the output X-BAR signal.	`off`

CLB X-BAR

Parameter	Description	Default Value
AUXSIG# MUX select on page 1-83	Select the MUX to map the signal to AUXSIG# MUX.	`Disable all`
Select MUX input on page 1-83	Select the input to the MUX selected for `AUXSIG# MUX select`.	`X:Disable`
AUXSIG# MUX (MUX 0->31) on page 1-83	Indicates the input signal selected for each AUXSIG MUX.	`XXXXXXXXXXXXXXXXXXXXXXXX`
Reset AUXSIG# MUX on page 1-83	Option to reset the MUX inputs selected.	
Invert AUXSIG# on page 1-83	Inverts the CLB X-BAR signal or Inverts the AUXSIG# signal.	`off`

CLB

Parameter	Description	Default Value
Enable CLB Tile # on page 1-93	Select this option to enable the CLB tile.	`off`
Tile # Name on page 1-93	Specify the tile name. This tile name will be used to generate the required function declaration and calling the function for CLB tile configuration file (clb_config.c and clb_config.h) generated using CLB tool. The tile names entered here should be an exact match with the tile name set in CLB tool to generate the file.	`TILE#`
IN# MUX selection on page 1-93	Configure the signal source type for the IN# mux. The type of signal can be global inputs, local inputs and GPREG.	`Global inputs`
Input on page 1-93	Configure the peripheral signal as input to the CLB tile.	`ePWM#A`
Input filtering on page 1-93	Configure the type of input filtering for the signal type Global inputs and Local inputs. This option will be disabled for GPREG as it is not applicable.	`No filtering`
Enable sync on page 1-93	Configure the synchronization option (SYNC) for the Global inputs and Local inputs IN# mux selection only. This option is not applicable for GPREG type and will be disabled for the same.	`off`
Route OUT# signal to on page 1-93	Option to enable routing the CLB output signal to the peripheral instead of the default peripheral signal. Each CLB output signal passes through an external multiplexer that intersects a specific peripheral signal. If the options in this parameter is enabled it will route the CLB output for the specific peripheral instead of the original peripheral signal.	

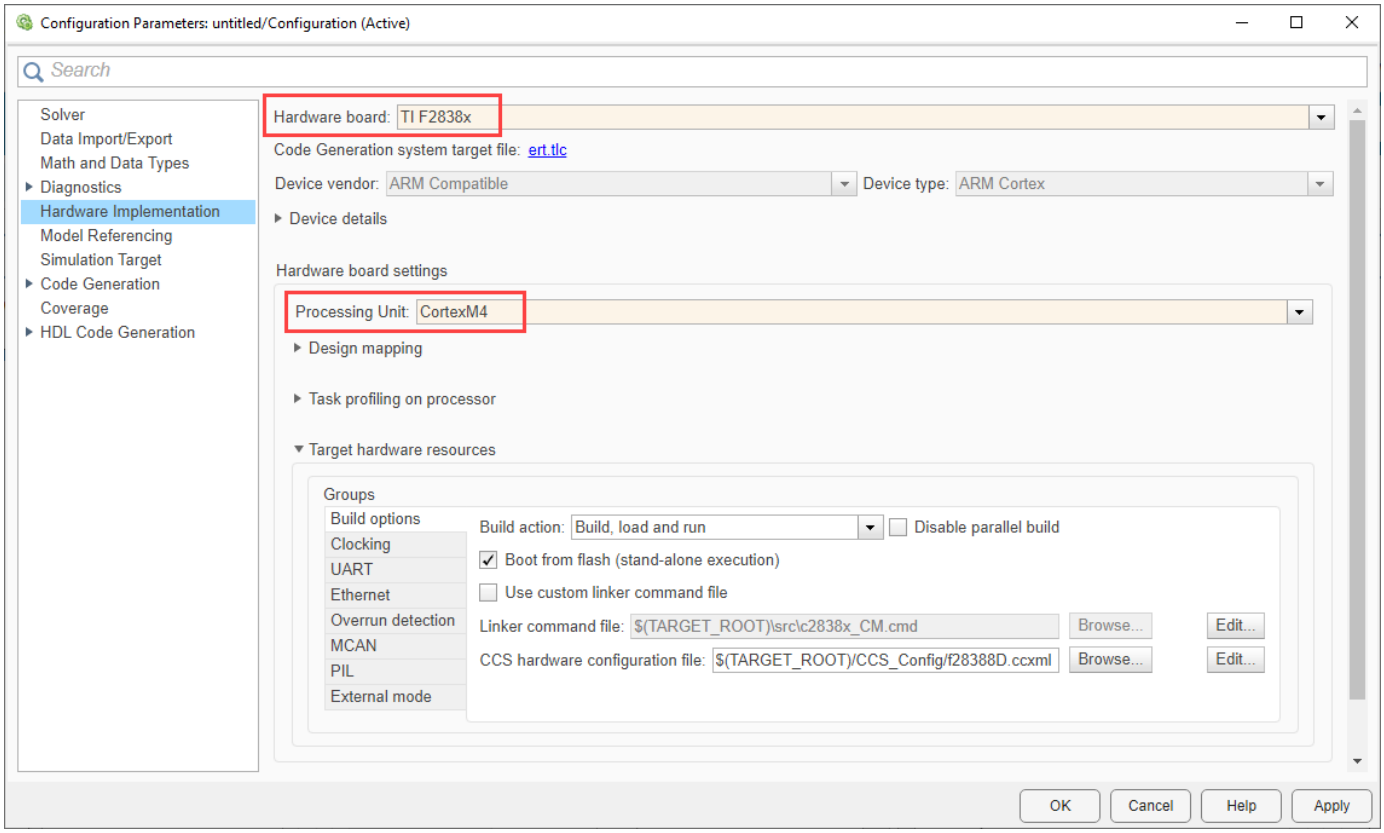
Parameter	Description	Default Value
CLB configuration header file (clb_config.h) on page 1-93	Provide the paths for the CLB configuration header file clb_config.h generated using CLB tool. This file holds the required function declarations and headers used to configure the CLB tile.	`clb_config.h`
CLB configuration source file (clb_config.c) on page 1-93	Provide the paths for the CLB configuration source file clb_config.c generated using CLB tool. This file holds the required function definitions used to configure the CLB tile.	`clb_config.c`
Browse on page 1-93	Click this button to browse the path for the file selection.	
Edit on page 1-93	Click this button to open the existing file for editing in MATLAB® editor.	

For more information on selecting a hardware board and general configuration settings, see “Hardware Implementation Pane”.

Model Configuration Parameters for Texas Instruments F2838x (ARM Cortex-M4)

To configure hardware parameters for Texas Instruments F2838x ARM Cortex-M4 processor:

1. In the Simulink Editor, select **Modeling** > **Model Settings**.
2. In the Configuration Parameter dialog box, click **Hardware Implementation**.



3. Set the **Hardware board** parameter to your TI F2838x and select the processing unit as **CortexM4**.
4. Click **Apply**.

Hardware Board Settings

For each hardware board you select, you can configure the board parameters according to your requirements.

Build Options

Parameter	Description	Default Value
Build action on page 1-112	Define how Embedded Coder responds when you build your model.	`Build, load, and run`
Device name on page 1-112	Select your device from the selected processor family.	`F28388D`
Disable parallel build on page 1-112	Select to compile the generated code and driver source codes in parallel order for faster build and deployment speed.	`off`
Boot From Flash (stand alone execution) on page 1-112	Specify if the application loads to the flash memory.	`enabled`
Use custom linker command file on page 1-112	Indicates that the custom linker command file must be used during the build action.	`off`
Linker command file on page 1-112	The path to the memory description file required during linking.	
CCS hardware configuration file on page 1-112	The Code Composer Studio file required for downloading the application on the hardware.	

Clocking

Parameter	Description	Default Value
Enter the 'Connectivity Manager (ARM Cortex-M) clock in MHz' value calculated in C28x CPU1 on page 1-114	Value of this parameter must be same as the value of the parameter 'Connectivity Manager (ARM Cortex-M) clock in MHz' (auto calculated in CPU1 model).	100

UART

Parameter	Description	Default Value
Enable UART Loopback on page 1-117	Select this check box to enable data transmission from Tx to Rx buffer.	`Not selected`
Desired Baud rate (in bits/sec) on page 1-117	Specify the desired baud rate of the data transmission.	`115200`
Closest Achievable Baud rate (in bits/sec) on page 1-117	The value in this parameter is calculated based on the desired baud rate that you specify and the system clock frequency.	`115207`
Number of stop bits on page 1-117	Select the number of stop bits used to indicate the end of a byte data transmission.	`1`
Parity mode on page 1-117	Select a parity mode that is added at the end of a binary data for error detection.	`None`
Pin assignment(Tx) on page 1-117	Select a GPIO pin as the UART pin for data transmission.	`GPIO84`
Pin assignment(Rx) on page 1-117	Select a GPIO pin as UART pin for data reception.	`GPIO85`
Enable receive interrupt on page 1-117	This parameter is enabled by default for updating DMA configuration after data receive.	`Selected`
Enable transmit interrupt on page 1-117	Select this parameter to trigger an ISR from an UART Transmit block. This will trigger UART interrupt when DMA copies any data to FIFO.	`Not selected`

Ethernet

Parameter	Description	Default Value
Enable DHCP for local IP address assignment on page 1-115	Select this parameter to configure the board to get an IP address from the local DHCP server on the network.	`Not selected`
Local IP address on page 1-115	Select this parameter to set the IP address of the board.	`192.168.1.8`
Subnet mask on page 1-115	Specify the subnet mask for the board.	`255.255.255.0`
Gateway on page 1-115	Set the serial gateway to the gateway required to access the target computer.	`192.168.1.1`
MAC address on page 1-115	Specify the media access control (MAC) address, the physical network address of the board.	`A8-63-F2-00-00-80`

Overrun detection

Parameter	Description	Default Value
Enable overrun detection on page 1-71	Enable to notify when task overrun occurs.	`off`
Set/Clear/toggle GPIO on page 1-71	Enable to select GPIO action.	`on`
Digital output pin to set an overrun on page 1-71	Specify the GPIO number of a digital output.	`34`
GPIO set mode on page 1-71	Select the GPIO mode.	`Set`
Additional notification option on page 1-71	Select the option to notify when task overrun occurs.	`None`
Interrupt number on page 1-71	Specify the interrupt number to trigger on overrun.	`1`
Name of the function on page 1-71	Specify the name of the C function to call on overrun.	`C2000_OverunFunction`

MCAN

Parameter	Description	Default Value
Protocol mode on page 1-96	Select the CAN type. Classic CAN or CAN-FD.	`CAN-FD`
MCAN module clock frequency (=connectivity manager (ARM Cortex-M) clock)in MHz on page 1-96	Displays the MCAN module clock frequency in MHz.	`100`
MCAN bit clock frequency (MCAN module clock freq/4) in MHz on page 1-96	Displays the MCAN bit clock frequency in MHz.	`25`
Nominal bit rate prescaler (NBRP: 1 to 512) on page 1-96	Nominal bit rate prescaler. The value by which the oscillator frequency is divided for generating the bit time quanta.	`1`
Nominal time segment 1 (NTSEG1: 2 to 256) on page 1-96	Nominal time segment before sample point.	`22`
Nominal time segment 2 (NTSEG2: 2 to 128) on page 1-96	Nominal time segment after sample point.	`2`
Closest achievable nominal baud rate (MCAN bit clock/ NBRP/(NTSEG1+NTSEG2)) in bits/sec on page 1-96	Closest achievable nominal MCAN baud rate in bits/sec.	`1000000`
Nominal re-synchronization jump width (NSJW: 1 to 128) on page 1-96	Nominal Resynchronization Jump Width (NSJW).	`1`
Enable bit rate switching on page 1-96	Enables bit rate switching between nominal bit rate and data bit rate.	`off`
Data bit rate prescaler (DBRP: 1 to 32) on page 1-96	Data Bit Rate Prescaler (DBRP). The value by which the oscillator frequency is divided for generating the bit time quanta.	`1`
Data time segment 1 (DTSEG1: 1 to 32) on page 1-96	Data time segment before sample point (DTSEG1).	`22`
Data time segment 2 (DTSEG2: 1 to 16) on page 1-96	Data time segment after sample point (DTSEG2).	`2`
Data baud rate (MCAN bit clock/DBRP/ (DTSEG1+DTSEG2)) in bits/sec on page 1-96	Closest achievable MCAN data baud rate calculated based on data parameters and given formula.	`1000000`

Parameter	Description	Default Value
Data re-synchronization jump width (DSJW: 1 to 16)) on page 1-96	Data resynchronization jump width (DSJW).	`1`
Mode on page 1-96	Select the operating mode for MCAN.	`Normal`
Pin assignment(Tx) on page 1-96	Select a GPIO pin for the MCAN data transmission.	`GPIO 31`
Pin assignment(Rx) on page 1-96	Select a GPIO pin for the MCAN data reception.	`GPIO 30`
Transmission mode on page 1-96	Select the mode of transmission.	`FIFO`
Enable blocking mode for Rx FIFO 0 on page 1-96	Enable blocking mode for FIFO 0 data reception.	`off`
Enable blocking mode for Rx FIFO 1 on page 1-96	Enable blocking mode for FIFO 1 data reception.	`off`
Update global filter configuration on page 1-96	Enable this parameter to update standard and extended filter IDs.	`off`
Reject remote frames standard on page 1-96	Rejects all remote frames with 11-bit standard IDs when enabled else the remote frames will be filtered as per the settings from Update standard filter elements.	`on`
Reject remote frames extended on page 1-96	Rejects all remote frames with 29-bit extended IDs when enabled else the remote frames will be filtered as per the settings from Update extended filter elements.	`on`
Non-matching frames extended on page 1-96	Defines how received messages with 11-bit standard IDs that do not match any element from **Update standard filter** elements are treated.	`Reject`
Non-matching frames standard on page 1-96	Defines how received messages with 29-bit extended IDs that do not match any element from **Update extended filter** elements are treated.	`Reject`
Update standard filter elements on page 1-96	Enable this parameter to update the standard 11bit ID filter elements parameters.	`off`
Select standard filter on page 1-96	Select the standard message ID filter elements.	`0`

Parameter	Description	Default Value
Filter # configuration on page 1-96	Select the standard filter element configuration.	`Disable filter element`
Filter # type (filter type will be ignored if filter configuration is stored into Rx buffer) on page 1-96	Select the standard filter type.	`Classic ID and mask filter (ID1 = filter, ID2 = mask)`
Filter # ID1 on page 1-96	Specify the standard Filter ID 1.	`0`
Filter # ID2 on page 1-96	Specify the standard Filter ID 2.	`0`
Update extended filter elements on page 1-96	Enable to update the extended filter elements.	`off`
Select extended filter on page 1-96	Select the extended message ID filter elements.	`0`
Filter # configuration on page 1-96	Select extended filter element configuration.	`Disable filter element`
Filter # type (filter type will be ignored if filter configuration is stored into Rx buffer) on page 1-96	Select the extended filter type.	`Classic ID and mask filter (ID1 = filter, ID2 = mask)`
Filter # ID1 on page 1-96	Specify the first ID of extended ID filter element.	`0`
Filter # ID2 on page 1-96	Specify the second ID of extended ID filter element.	`0`
Display configured extended and standard filters elements in command window on page 1-96	Click on **Display configured extended and standard filters elements** in command window button to view the configured standard and extended filter elements in MATLAB command window.	
Reset standard filters configurations on page 1-96	Click **Reset standard filters configurations** to reset the configured standard filter configurations.	
Reset extended filters configurations on page 1-96	Click **Reset extended filters configurations** to reset the configured extended filter configurations.	
Configure memory on page 1-96	Select to configure the memory and its parameters.	`off`
Maximum element size in TX FIFO (in bytes) on page 1-96	Select the maximum data size of CAN FD message in transmit FIFO.	`64`

Parameter	Description	Default Value
Maximum element size in RX FIFO 0 (in bytes) on page 1-96	Select the maximum data size of CAN FD message in receive FIFO 0.	`64`
Maximum element size in RX FIFO 1 (in bytes) on page 1-96	Select the maximum data size of CAN FD message in receive FIFO 1.	`64`
Maximum element size in RX buffer (in bytes) on page 1-96	Select the maximum data size of CAN FD message in receive buffer.	`64`
Number of elements in TX FIFO Queue on page 1-96	Select the number of elements (data + header CAN FD message) in transmit FIFO/ Queue.	`32`
Number of elements in RX FIFO 0 on page 1-96	Select the number of elements (data + header CAN FD message) in receive FIFO 0.	`Auto allocate`
Number of elements in RX FIFO 1 on page 1-96	Select the number of elements (data + header CAN FD message) in receive FIFO 1.	`Auto allocate`
Validate memory on page 1-96	Click **Validate memory** button to validate all the memory configurations.	
Configure receive interrupt sources on page 1-96	Select this option to display receive interrupt sources for configuration.	`off`
Configure RX buffer interrupt sources on page 1-96	Select this option to display buffer interrupt sources for configuration.	`off`
Dedicated RX buffer message on page 1-96	Select the dedicated interrupt line for receive buffer message.	`Disable`
High priority message on page 1-96	Select the dedicated interrupt line for high priority message.	`Disable`
Configure RX FIFO 0 interrupt sources on page 1-96	Select this option to display receive FIFO 0 interrupt sources for configuration.	`off`
RX FIFO 0 new message on page 1-96	Select the interrupt line for receive FIFO 0 new message.	`Disable`
RX FIFO 0 full on page 1-96	Select the interrupt line for receive FIFO 0 full.	`Disable`
RX FIFO 0 message lost on page 1-96	Select the interrupt line for receive FIFO 0 message lost.	`Disable`
RX FIFO 0 watermark on page 1-96	Select the interrupt line for receive FIFO 0 watermark.	`Disable`

Parameter	Description	Default Value
Configure RX FIFO 1 interrupt sources on page 1-96	Select this option to display receive FIFO 1 interrupt sources for configuration.	`off`
RX FIFO 1 new message on page 1-96	Select the interrupt line for receive FIFO 1 new message.	`Disable`
RX FIFO 1 full on page 1-96	Select the interrupt line for receive FIFO 1 full.	`Disable`
RX FIFO 1 message lost on page 1-96	Select the interrupt line for receive FIFO 1 message lost.	`Disable`
RX FIFO 1 watermark on page 1-96	Select the interrupt line for receive FIFO 1 watermark.	`Disable`
Configure transmit interrupt sources on page 1-96	Select this option to display transmit interrupt sources for configuration.	`off`
Configure TX FIFO interrupt sources on page 1-96	Enable to configure the transmit FIFO interrupt sources.	`off`
Transmission complete on page 1-96	Select the transmission interrupt line for transfer complete.	`Disable`
Transmission cancellation finish on page 1-96	Select the transmission interrupt line for transfer cancellation finish.	`Disable`
TX FIFO empty on page 1-96	Select the transmission interrupt line for TX FIFO empty.	`Disable`
Configure TX event FIFO interrupt sources on page 1-96	Select this option to display transmit event FIFO interrupt sources for configuration.	`off`
TX event FIFO new entry on page 1-96	Select the transmission interrupt line for TX event FIFO new entry.	`Disable`
TX event FIFO element lost on page 1-96	Select the transmission interrupt line for TX event FIFO element lost.	`Disable`
TX event FIFO full on page 1-96	Select the transmission interrupt line for TX event FIFO full.	`Disable`
TX event FIFO watermark on page 1-96	Select the transmission interrupt line for TX event FIFO watermark.	`Disable`
Configure other interrupt sources on page 1-96	Select this option to display other interrupt sources for configuration.	`off`

Parameter	Description	Default Value
Timestamp wraparound on page 1-96	Select the interrupt line for timestamp wraparound interrupt.	`Disable`
Timeout occurred on page 1-96	Select the interrupt line for timeout occurred interrupt.	`Disable`
Error logging overflow on page 1-96	Select the interrupt line for error logging overflow interrupt.	`Disable`
Warning status on page 1-96	Select the interrupt line for warning status interrupt.	`Disable`
Watchdog event on page 1-96	Select the interrupt line for watchdog event interrupt.	`Disable`
Data protocol error on page 1-96	Select the interrupt line for data protocol error interrupt.	`Disable`
Message RAM access failure on page 1-96	Select the interrupt line for message RAM access failure interrupt.	`Disable`
Bit error uncorrected on page 1-96	Select the interrupt line for bit error uncorrected interrupt.	`Disable`
Error passive status on page 1-96	Select the interrupt line for error passive status interrupt.	`Disable`
Bus off status on page 1-96	Select the interrupt line for bus off status interrupt.	`Disable`
Arbitration protocol error on page 1-96	Select the interrupt line for arbitration protocol error interrupt.	`Disable`
Reserved address access on page 1-96	Select the interrupt line for reserved address access interrupt.	`Disable`

PIL

Parameter	Description	Default Value
“PIL” on page 1-109	Select the type of communication interface to run your model.	`Serial`
Serial port in MATLAB preferences on page 1-109	Select the COM port used by the target hardware.	COM#
PIL Baud rate (UART Baud rate) on page 1-109	PIL baud rate used by the target. This is based on the baud rate that you specify in the **Desired Baud rate (in bits/sec)** parameter for UART0.	

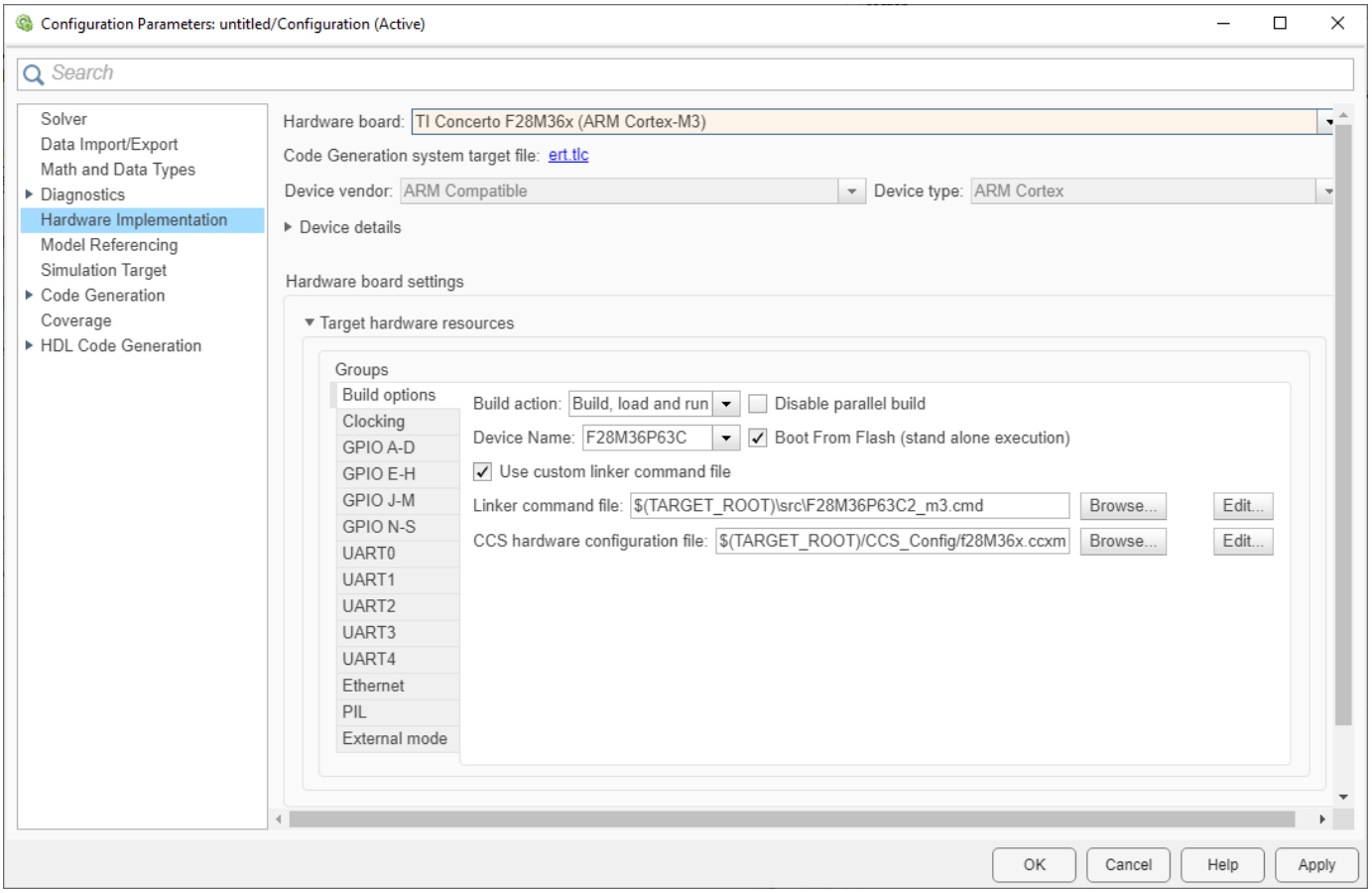
External Mode

Parameter	Description	Default Value
Communication interface on page 1-107	Select the type of communication interface to run your model in external mode.	`XCP on Serial`
Serial port in MATLAB preferences on page 1-107	Select the COM port used by the target hardware.	
Verbose on page 1-107	Select to view the external mode execution progress and updates in the Diagnostic Viewer or in the MATLAB command window.	`off`
Set logging buffer size automatically on page 1-107	Select to automatically set the number of bytes to preallocate for the buffer in the hardware during simulation.	`on`
Maximum number of contiguous samples on page 1-107	Specify the maximum number of contiguous samples to be packed in a single packet.	`8`

For more information on selecting a hardware blockset and general configuration settings, see “Hardware Implementation Pane”.

Model Configuration Parameters for Texas Instruments Concerto F28M3x (ARM Cortex-M3)

Hardware Implementation Pane Overview



1. In the Simulink Editor, select **Simulation** > **Model Configuration Parameters**.
2. In the Configuration Parameter dialog box, click **Hardware Implementation**.
3. Set the **Hardware board** parameter to a value such as `TI Concerto F28M36x (ARM Cortex-M3)`.
4. The parameter values under **Hardware board settings** are automatically populated to their default values.

   You can optionally adjust these parameters for your particular use case.
5. Click **Apply** to apply the changes.

For more information on selecting a hardware blockset and general configuration settings, see “Hardware Implementation Pane”.

Scheduler Options

Parameter	Description	Default Value
Base rate trigger on page 1-168	Set the static priority of the base rate task in the operating system.	`Timer 0`

C28x / ARM Cortex-M3 - Build options

C28x / ARM Cortex-M3 - Build options

Parameter	Description	Default Value
“Build action” on page 1-57	The option to specify how the build process should take place during code generation.	Build, load and run
“Device name” on page 1-57	The option to select a particular device from the selected processor family in the Target hardware parameter on the Code Generation pane.	F28M36P63C
“Disable parallel build” on page 1-57	Select to compile the generated code and driver source codes in parallel order for faster build and deployment speed.	`off`
“Boot From Flash (stand alone execution)” on page 1-57	The option to specify if the application has to load to the flash. If you do not select this option, the application loads to the RAM.	Selected
“Use custom linker command file” on page 1-57	The option to indicate that the custom linker command file must be used during the build action. Select this option, if you have your own custom linker file, which you can specify in Linker command file parameter. If you do not select this option, based on the device you have selected, a default custom linker command file will be used.	Selected
“Linker command file” on page 1-57	The path to memory description file that is required during linking. For each family of TI processor selected under ‘Target Hardware’, one linker command file will be selected automatically.	$(TARGET_ROOT)\src \c28M35H52C.cmd

Parameter	Description	Default Value
“CCS hardware configuration file” on page 1-58	The Code Composer Studio file required for downloading the application on the hardware. Select one of the .ccxml files from the folder ‘CCS_Config’ folder under blockset installation folder.	$(TARGET_ROOT)/CCS_Config/f28M35x.ccxml

M3x-Clocking

Clocking

Parameter	Description	Default Value
“Desired C28x CPU clock in MHz” on page 1-59	Specify the expected C28x CPU clock frequency and match the same in your C28x Model.	150
“Oscillator clock (OSCCLK) frequency in MHz” on page 1-59	Specify the frequency of the crystal oscillator used in the board.	20
“Auto set PLL based on OSCCLK and CPU clock” on page 1-59	The option that helps you to set the PLL control register value automatically.	Selected
“System PLL multiplier (SYSPLLMULT)[1–127.75]” on page 1-59	Specify the system PLL multiplier. You can specify a value in this parameter if **Auto set PLL based on OSCCLK and CPU clock** is not selected.	15
“System clock divider (SYSDIVSEL)” on page 1-59	If you select the **Auto set PLL based on OSCCLK and CPU clock** check box, the auto calculated clock divider value achieves the specified CPU Clock value based on the Oscillator clock frequency. Otherwise, you can select a value for Clock divider (SYSDIVSEL).	1
“Achievable C28x SYSCLK in MHz = (OSCCLK * SYSPLLMULT/ 2/ SYSDIVSEL)” on page 1-59	The auto calculated feedback value that matches most closely to the desired CPU Clock value on the board, based on the values of OSCCLK, SYSPLLMULT, and the SYSDIVSEL.	150
“M3 System clock divider (M3SSDIVSEL)” on page 1-59	Select a value from the options for M3 system clock divider.	2

Parameter	Description	Default Value
“M3 SYSCLK in MHz = (OSCCLK * SYSPLLMULT/ 2/ SYSDIVSEL/ M3SSDIVSEL)” on page 1-60	This is the achievable M3 system clock frequency.	75

M3x-GPIO A–D

Parameter	Description	Default Value
“Enable GPIO port A” on page 1-61	Select this option to enable GPIO port A.	Selected
“Show GPIOA settings for” on page 1-61	Select GPIO pins from port A for which you want to set the CPU core and the pin type.	Pin 0
“Select the CPU core which controls Pin #” on page 1-61	Select the CPU core for the selected GPIO pin.	
“Select the pin type for Pin 0” on page 1-61	Select the pull–up and the open–drain options for the selected GPIO pin.	

M3x-UART0–4

Parameter	Description	Default Value
“Enable UART Loopback” on page 1-62	Select this check box to enable data transmission from Tx to Rx buffer.	Not selected
“Enable M3 UART4 to C28 SCI-A Loopback” on page 1-62	Select this check box to enable data transmission from M3 UART4 to C28 SCI-A.	Not selected
“Desired Baud rate (in bits/sec)” on page 1-62	Specify the desired baud rate of the data transmission.	115200
“Closest Achievable Baud rate (in bits/sec)” on page 1-62	The value in this parameter is calculated based on the desired baud rate that you specify and the system clock frequency.	115207
“Number of stop bits” on page 1-62	Select the number of stop bits used to indicate the end of a byte data transmission. The options available:	1
“Parity mode” on page 1-62	Select a parity mode that is added at the end of a binary data for error detection.	None

Parameter	Description	Default Value
“Pin assignment(Tx)” on page 1-63	Select a GPIO pin as the UART pin for data transmission. By default, the GPIO29 is hardwired as the Tx GPIO to the FTDI chip.	PE5_GPIO29
“Pin assignment(Rx)” on page 1-63	Select a GPIO pin as UART pin for data reception.	PE4_GPIO28
“Enable Transmit Interrupt” on page 1-63	Select this check box to enable the transmit interrupt. This will trigger UART interrupt when DMA copies any data to FIFO.	Not selected
“Enable Receive Interrupt” on page 1-63	This check box by default is enabled for communication with external mode over serial.	Selected

M3x-Ethernet

Parameter	Description	Default Value
“Enable DHCP for local IP address assignment” on page 1-64	Select this check box to configure the board to get an IP address from the local DHCP server on the network.	Selected
“Local IP address” on page 1-64	Enter the IP address of the board.	192.168.1.10
“Subnet mask” on page 1-64	Enter the subnet mask for the board. A subnet mask divides an IP address into network address and a host address.	255.255.255.0
“Ethernet local host name” on page 1-64	Enter the local host name.	Concerto-M3
“MAC address” on page 1-64	Enter the MAC address.	A8-63-F2-80-90-80

M3x-PIL

Parameter	Description	Default Value
“PIL communication interface” on page 1-65	Select the communication interface for PIL. The available options are: `Serial` and `TCP/IP`.	Serial
“Serial port” on page 1-65	Enter the serial port used by the target hardware.	COM1

Parameter	Description	Default Value
“PIL Baud Rate (UART) Baud rate)” on page 1-65	This is the PIL baud rate used by the target. This is based on the baud rate that you specify in the **Desired Baud rate (in bits/sec)** parameter for UART0.	115207
“Ethernet port” on page 1-65	This is the Ethernet port used for PIL communication.	17725

External mode

Parameter	Description	Default Value
“Communication interface” on page 1-66	Use the ‘serial’ option to run your model in the External mode with serial communication.	Serial
“Serial port” on page 1-66	Enter the serial port used by the target hardware.	COM4
“Verbose” on page 1-66	Select this check box to view the External Mode execution progress and updates in the Diagnostic Viewer or in the MATLAB command window.	Not selected

SD Card Logging

Parameter	Description	Default Value
Enable MAT-file logging on SD card on page 1-184	Enables the MAT-file logging for SD card.	`off`
SPI module on page 1-184	Select the desired interface on which the SD card is connected to hardware board.	
SPI baud rate on page 1-184	Select the desired option for the SPI interface used by the SD card.	`Maximum achievable supported by the inserted SD Card`

For information on other configuration options, see “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2.

C28x / ARM Cortex-M3 - Build options

Use the build options to specify how the build process should take place during code generation.

Build action

The option to specify if you want only ‘build’ or ‘build, load, and run’ action during the build process. The `build, load and run` option is supported only for TI Code Composer Studio v5 (C2000), v6 (C2000), and v5(ARM)/v6(ARM) tool chain.

If you select `build, load, and run` option, you must provide the required CCS hardware configuration file.

Device name

The option to select a particular device from the selected processor family in the Target hardware parameter on the Code Generation pane.

Disable parallel build

- `on` – When you select this option, the blockset compiles generated code and driver source codes in sequential order.
- `off` – When you clear the selection, the blockset compiles generated code and driver source codes parallely. Parallel execution reduces the time taken to build the model.

Boot From Flash (stand alone execution)

The option to specify if the application has to load to the flash. If you do not select this option, the application loads to the RAM.

Use custom linker command file

The option to indicate that the custom linker command file must be used during the build action. Select this option, if you have your own custom linker file, which you can specify in Linker command file parameter. If you do not select this option, based on the device you have selected, a default custom linker command file will be used.

Linker command file

The path to memory description file that is required during linking. For each family of TI processor selected under ‘Target Hardware’, one linker command file will be selected automatically.

For different variant of processor, you can select from the ‘src’ folder inside the blockset installation path. You can also create custom linker command file and select the file path using **Browse**.

CCS hardware configuration file

The Code Composer Studio file required for downloading the application on the hardware. Select one of the .ccxml files from the folder ‘CCS_Config’ folder under blockset installation folder.

Instead, you can also create your own .ccxml file.

Select the file you created using **Browse**.

The .ccxml files provided with the blockset are as follows:

- f28M35x.ccxml – Texas Instruments XDS100v2 USB Emulator_0
- f28M36x.ccxml – Texas Instruments XDS100v2 USB Emulator_0

M3x-Clocking

Desired C28x CPU clock in MHz

Specify the expected C28x CPU clock frequency and match the same in your C28x Model. The C28x CLOCK is the same as PLLSYSCLK. The M3 Clock is a factor of M3SSDIVSEL divided by the PLLSYSCLK.

Oscillator clock (OSCCLK) frequency in MHz

Specify the frequency of the crystal oscillator used in the board. In case of Concerto the crystal oscillator is external to the processor.

Auto set PLL based on OSCCLK and CPU clock

The option that helps you to set the PLL control register value automatically. When you select this check box, the values in the SYSPLLMULT, SYSDIVSEL , and the Achievable C28x SYSCLK in MHz parameters are automatically calculated based on the **Desired C28x CPU Clock** value entered on the Board.

System PLL multiplier (SYSPLLMULT)[1–127.75]

Specify the system PLL multiplier. You can specify a value in this parameter if **Auto set PLL based on OSCCLK and CPU clock** is not selected. The PLL multiplier is a 9 bit field with 7 bits of the SYSPLLMULT register comprising of the integer portion and the remaining 2 bits for the fractional portion. You can enter a value in the range between 0 to 127.75 with multiples of 0.25 for fractional portion of the value.

System clock divider (SYSDIVSEL)

If you select the **Auto set PLL based on OSCCLK and CPU clock** check box, the auto calculated clock divider value achieves the specified CPU Clock value based on the Oscillator clock frequency. Otherwise, you can select a value for Clock divider (SYSDIVSEL).

Achievable C28x SYSCLK in MHz = (OSCCLK * SYSPLLMULT/ 2/ SYSDIVSEL)

The auto calculated feedback value that matches most closely to the desired CPU Clock value on the board, based on the values of OSCCLK, SYSPLLMULT, and the SYSDIVSEL.

M3 System clock divider (M3SSDIVSEL)

Select a value from the options for M3 system clock divider. The C28 CLKIN clock is divided by the selected value to generate the M3 CPU clock.

M3 SYSCLK in MHz = (OSCCLK * SYSPLLMULT/ 2/ SYSDIVSEL/ M3SSDIVSEL)

This is the achievable M3 system clock frequency. This is calculated based on the values of OSCCLK, SYSPLLMULT, SYSDIVSEL, and M3SSDIVSEL.

M3x-GPIO A–D

Enable GPIO port A

Select this option to enable GPIO port A.

Show GPIOA settings for

Select GPIO pins from port A for which you want to set the CPU core and the pin type.

Select the CPU core which controls Pin

- Select the CPU core for the selected GPIO pin.
  - Auto detect M3 usage, otherwise set to C28x — This default option detects, if you have used the selected GPIO pin for the M3 block in your model.
  - M3 — Select this option to assign the GPIO pin to the M3 CPU.
  - C28x — Select this option to assign the GPIO pin to the C28x CPU.

Select the pin type for Pin 0

Select the pull–up and the open–drain options for the selected GPIO pin.

Note The above parameter descriptions are also applicable for all the GPIO ports.

M3x-UART0–4

Enable UART Loopback

Select this check box to enable data transmission from Tx to Rx buffer. However, selecting this option does not ensure that the data is present in the GPIO MUX.

Enable M3 UART4 to C28 SCI-A Loopback

Select this check box to enable data transmission from M3 UART4 to C28 SCI-A. This option is available only for UART4 parameter.

Desired Baud rate (in bits/sec)

Specify the desired baud rate of the data transmission.

Closest Achievable Baud rate (in bits/sec)

The value in this parameter is calculated based on the desired baud rate that you specify and the system clock frequency. This baud rate is used for the data transfer.

Number of stop bits

Select the number of stop bits used to indicate the end of a byte data transmission. The options available:

- `1` – Select this option to indicate there is 1 stop bit at the end of a byte data transmission.
- `2` – Select this option to indicate there are 2 stop bits at the end of a byte data transmission.

Parity mode

- Select a parity mode that is added at the end of a binary data for error detection.

  The options available are:

  - `None` — Select this option when no parity is used. This is the default option.
  - `Odd` — Select this option to indicate that odd parity is used for data transmission.

    In odd parity mode, the parity bit is set to ‘1’ if the sum of bits with the value ‘1’ is even and the parity bit is set to ‘0’, if the sum of bits with the value ‘1’ is odd.
  - `Even` — Select this option to indicate that even parity is used for data transmission.

    In even parity mode, the parity bit is set to ‘1’, if the sum of bits with the value ‘1’ is odd and the parity bit is set to ‘0’, if the sum of bits with the value ‘1’ is even.
  - `One` — Select this option to indicate that the parity bit is always ‘1’.
  - `Zero` — Select this option to indicate that the parity bit is always ‘0’.

Pin assignment(Tx)

Select a GPIO pin as the UART pin for data transmission. By default, the GPIO29 is hardwired as the Tx GPIO to the FTDI chip.

Pin assignment(Rx)

Select a GPIO pin as UART pin for data reception. By default, the GPIO28 is hardwired as the Rx GPIO to the FTDI chip.

Enable Transmit Interrupt

Select this check box to enable the transmit interrupt. This will trigger UART interrupt when DMA copies any data to FIFO.

Enable Receive Interrupt

This check box by default is enabled for communication with external mode over serial.

M3x-Ethernet

Enable DHCP for local IP address assignment

Select this check box to configure the board to get an IP address from the local DHCP server on the network.

Local IP address

Enter the IP address of the board.

Subnet mask

Enter the subnet mask for the board. A subnet mask divides an IP address into network address and a host address.

Ethernet local host name

Enter the local host name.

MAC address

Enter the MAC address.

M3x-PIL

PIL communication interface

Select the communication interface for PIL. The available options are: `Serial` and `TCP/IP`.

Serial port

Enter the serial port used by the target hardware.

PIL Baud Rate (UART) Baud rate)

This is the PIL baud rate used by the target. This is based on the baud rate that you specify in the **Desired Baud rate (in bits/sec)** parameter for UART0.

Ethernet port

This is the Ethernet port used for PIL communication. This parameter appears only when you select `TCP/IP` option in **PIL communication interface** parameter.

External mode

Communication interface

Use the ‘serial’ option to run your model in the External mode with serial communication.

Serial port

Enter the serial port used by the target hardware.

Verbose

Select this check box to view the External Mode execution progress and updates in the Diagnostic Viewer or in the MATLAB command window.

Serial Configuration for External Mode and PIL

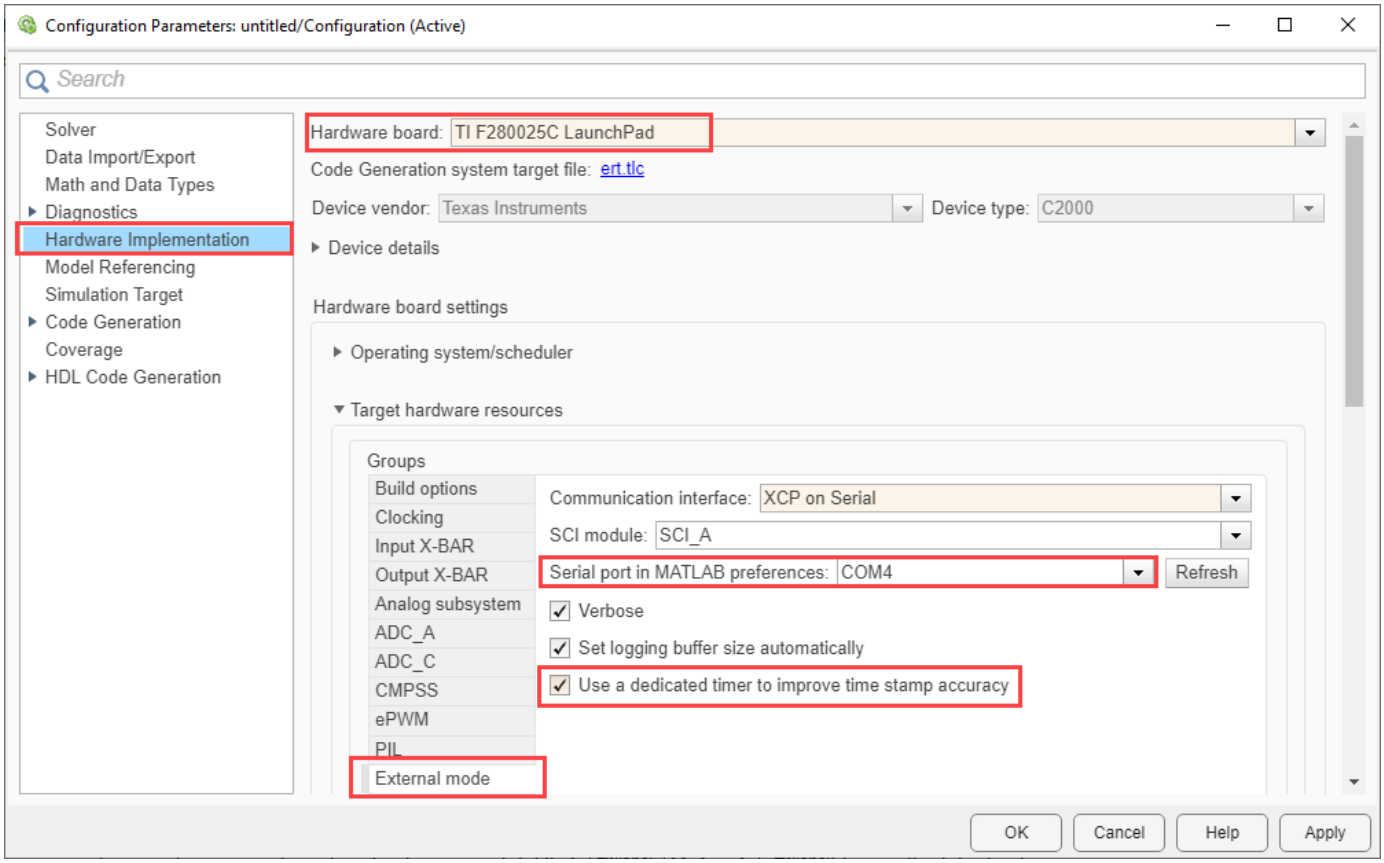
To prepare your model for external mode or PIL with serial communication:

1. In the Configuration Parameters dialog box, select **Model Configuration Parameters > Hardware Implementation**.
2. Select a C2000 processor from the Hardware Implementation > **Hardware board** drop-down list. For more, see
3. Under **Target Hardware Resources**, select **External mode/PIL**.
4. In **Communication interface** drop-down list, select `serial` for Classic External mode or `XCP on Serial` for External mode over XCP.

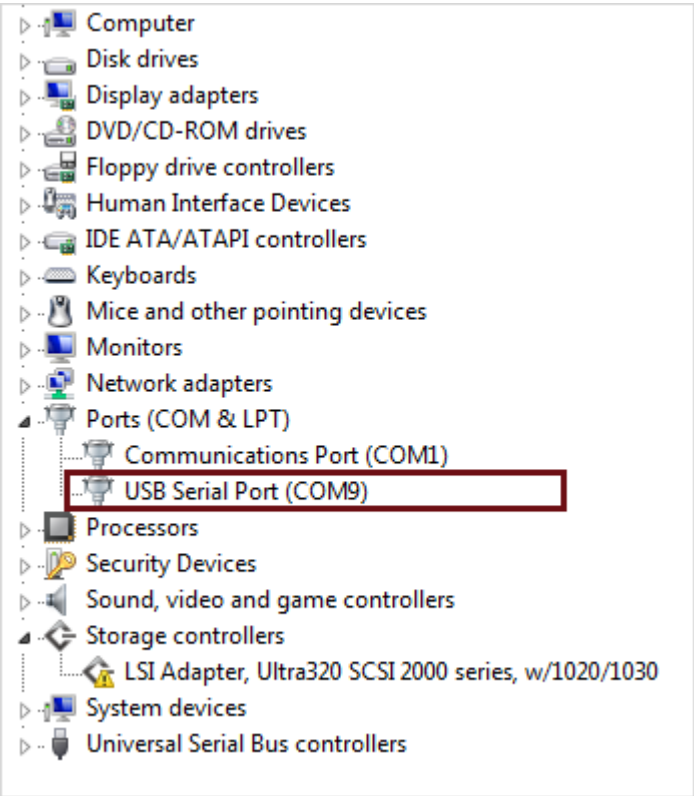
   Note For PIL, the communication interface is only with `serial`.
5. Select **SCI module** under **Target hardware resource** in **Model Configuration Parameters > Hardware Implementation > Hardware board**.
6. For the selected SCI module, configure the additional parameters **Desired baud rate in bits/sec**, **Pin assignment(Tx)** and **Pin assignment(Rx)**.

   Note

   - By default SCI_A module is selected for Controlcards and Launchpads. For custom boards, if you are using different serial module to connect FTDI then select the appropriate serial module.
   - The default baud for Controlcards is 115200 bits/sec. For better performance, you may increase the baud to a value that your hardware board permits.
7. Select the corresponding COM port that the target hardware uses from the drop-down. The COM port value will be saved as Serial port in MATLAB preference for a given target and not on a model. Click **Refresh** to see the latest value in MATLAB preference and updated list from device manager.



To see the list of available COM ports on your computer, select **Start > Control Panel > Device Manager > Ports (COM & LPT)**



You can also set the serial port in MATLAB preferences for the given hardware board using the MATLAB command:

```
codertarget.tic2000.setSerialPortPreferences(Hardware board,
 CPU value, Serial port)
```

Here CPU value is optional argument.

8 Make sure that the **Verbose** check box is selected to view the external mode execution progress and updates in the Diagnostic Viewer or in the Command Window.

9 Select **Use a dedicated timer to improve time stamp accuracy** parameter to log hardware time data inside ISR at ISR trigger rate and idle task at trigger rate for XCP External mode for Serial and CAN. For more information, see “External Mode” on page 1-180.

Your model is now ready to perform Monitor and Tune action (External Mode) over serial communication.

See Also

“Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

Analog subsystem

Analog subsystem: Analog subsystem is used to configure the flexible voltage references used by the analog modules i.e. ADC and DAC in the device. VREFHIx pin voltage can be driven in externally or can be generated by an internal bandgap voltage reference. The internal voltage reference range can be selected to be 0V to 3.3V or 0V to 2.5V

Ganged pins: Multiple VREFHI pins are ganged together in lower pin-count packages i.e. due to pin count restrictions of the device there is sharing of analog VREF pins among ADCs. In order to avoid potential pin contention between Internal vs External reference modes between ADCs, the reference voltage settings for the ganged pins should always be configured to the same setting. VREFHIB and VREFHIC are always ganged together for F28004x.

External reference for VREFHIx

By default, an internally generated band gap voltage reference supplies the ADC logic. However, depending on application requirements, you can enable the external reference so that the ADC logic uses an external voltage reference instead.

VREFHIx

The ADC logic uses internal bandgap voltage reference range to be 0V to 3.3V or 0V to 2.5V.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

Overrun detection

You can configure a Simulink model running on the target hardware to detect and notify you when a task overrun occurs. A task overrun occurs if the target hardware is still performing one instance of a task when the next instance of that task is scheduled to begin.

You can fix overruns by decreasing the frequency with which tasks are scheduled to run, and/or by reducing the number of tasks defined by your model.

If those solutions does not fix the task overrun condition, and you are using external mode or profiling, consider disabling external mode or profiling as both introduces additional code overhead.

Enable overrun detection

Enable to detect and notify when task overrun occurs.

Set/Clear/Toggle GPIO

Enable to select the General Purpose Input/Output (GPIO) action to notify on overrun.

Digital output pin to set an overrun

Specify the pin number of the digital output(GPIO) to notify on overrun.

Set GPIO mode

Select either `Set, Clear, or Toggle` as the GPIO action on overrun.

- **Set** - GPIO value will be set on overrun. Initial value is clear.
- **Clear** - GPIO value will be cleared on overrun. Initial value is set.
- **Toggle** - GPIO value toggles on overrun. Initial value is clear.

Additional notification option

Select to notify through any of the additional notification option on overrun: `None, Trigger interrupt, or Call user-defined function`.

PIE number

Specify the Peripherals Interrupt Expansion (PIE) number for the interrupt to trigger on overrun.

This parameter appears only for specific processors and when the parameter **Additional notification option** is set to `Trigger interrupt`.

CPU number

Specify the CPU number for the interrupt to trigger on overrun.

This parameter appears only for specific processors and when the parameter **Additional notification option** is set to `Trigger interrupt`.

Interrupt number

Specify the interrupt number to trigger on overrun.

This parameter is available only for F2838x (ARM Cortex-M4) processor and if the parameter **Additional notification option** is set to `Trigger interrupt`.

Name of the function

Specify the name of the custom C function to be called on overrun.

This parameter is available only if you set the parameter **Additional notification option** as `Call user-defined function`. For more information, refer “Model Configuration Parameters: Code Generation Custom Code” (Simulink Coder).

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2
- “Detect and Fix Task Overruns on Texas Instruments C2000 Hardware”

Input X-BAR

The crossbars (X-bars) provides flexibility to connect device inputs, outputs, and internal resources in a variety of configurations using Input X-BAR, Output X-BAR, and ePWM X-BAR.

The F2807x, F2837x, F2838x, F28002x, F28004x and F28003x processors support Input X-BAR. For more information, refer to TI Technical Reference Manual of there respective processors.

The Input X-BAR is used to route signals from a GPIO to many different peripherals blocks such as the ADC(s), eCAP(s), ePWM(s), and external interrupts. The Input X-BAR has access to every GPIO and can route each signal to any (or multiple) of the peripherals blocks.

Input X-Bar code is generated only when it is used by any peripheral in the model.

INPUT# pin assignment

Specify a valid GPIO number from which the signal will be passed to the corresponding destinations. When pin assignment is `None`, the corresponding Input X-Bar will not be configured.

Input X-BAR Destinations - F2838x

INPUT	DESTINATION
INPUT1	eCAPx, ePWM X-BAR, ePWM[TZ1,TRIP1], Output X-BAR, CLB X-BAR, Ethe
INPUT2	eCAPx, ePWM X-BAR, ePWM[TZ2,TRIP2], Output X-BAR, CLB X-BAR, Ethe
INPUT3	eCAPx, ePWM X-BAR, ePWM[TZ3,TRIP3], Output X-BAR, CLB X-BAR, Ethe
INPUT4	eCAPx, ePWM X-BAR, XINT1, Output X-BAR, CLB X-BAR, EtherCAT, ERAD
INPUT5	eCAPx, ePWM X-BAR, XINT2, ADCEXTSOC, EXTSYNCIN1, ePWM SYNC, e X-BAR, CLB X-BAR, EtherCAT, ERAD
INPUT6	eCAPx, ePWM X-BAR, XINT3, ePWM[TRIP6], EXTSYNCIN2, Output X-BAR SYNC, CLB X-BAR, EtherCAT, ERAD
INPUT7	eCAPx, ePWM X-BAR, CLB X-BAR, EtherCAT, ERAD, eCAP1, Capture Input
INPUT8	eCAPx, ePWM X-BAR, CLB X-BAR, EtherCAT, ERAD, eCAP2, Capture Input
INPUT9	eCAPx, ePWM X-BAR, CLB X-BAR, EtherCAT, ERAD, eCAP3, Capture Input
INPUT10	eCAPx, ePWM X-BAR, CLB X-BAR, EtherCAT, ERAD, eCAP4, Capture Input
INPUT11	eCAPx, ePWM X-BAR, CLB X-BAR, EtherCAT, ERAD, eCAP5, Capture Input
INPUT12	eCAPx, ePWM X-BAR, CLB X-BAR, EtherCAT, ERAD, eCAP6, Capture Input
INPUT13	eCAPx, ePWM X-BAR, XINT4, CLB X-BAR, EtherCAT, ERAD
INPUT14	eCAPx, ePWM X-BAR, XINT5, CLB X-BAR, EtherCAT, ERAD
INPUT15	eCAPx, EtherCAT, DCC Clock Source-1
INPUT16	eCAPx, EtherCAT, DCC Clock Source-0

Input X-BAR Destinations - F28004x

Input	Destinations
INPUT1	eCAPx, ePWM X-BAR, ePWM[TZ1,TRIP1], Output X-BAR
INPUT2	eCAPx, ePWM X-BAR, ePWM[TZ2,TRIP2], Output X-BAR
INPUT3	eCAPx, ePWM X-BAR, ePWM[TZ3,TRIP3], Output X-BAR
INPUT4	eCAPx, ePWM X-BAR, XINT1, Output X-BAR
INPUT5	eCAPx, ePWM X-BAR, XINT2, ADCEXTSOC, EXTSYNCIN1, Output X-BAR
INPUT6	eCAPx, ePWM X-BAR, XINT3, ePWM[TRIP6], EXTSYNCIN2, Output X-BAR
INPUT7	eCAPx, ePWM X-BAR
INPUT8	eCAPx, ePWM X-BAR
INPUT9	eCAPx, ePWM X-BAR
INPUT10	eCAPx, ePWM X-BAR
INPUT11	eCAPx, ePWM X-BAR
INPUT12	eCAPx, ePWM X-BAR
INPUT13	eCAPx, ePWM X-BAR, XINT4
INPUT14	eCAPx, ePWM X-BAR, XINT5
INPUT15	eCAPx
INPUT16	eCAPx

Input X-BAR Destinations - F2807x and F2837x

Input	Destinations
INPUT1	ePWM[TZ1,TRIP1], ePWM X-BAR, CLB X-BAR, Output X-BAR
INPUT2	ePWM[TZ2,TRIP2], ePWM X-BAR, CLB X-BAR, Output X-BAR
INPUT3	ePWM[TZ3,TRIP3], ePWM X-BAR, CLB X-BAR, Output X-BAR
INPUT4	XINT1, ePWM X-BAR, CLB X-BAR, Output X-BAR
INPUT5	XINT2, ADCEXTSOC, EXTSYNCIN1, ePWM X-BAR, CLB X-BAR, Output X-B
INPUT6	XINT3, ePWM[TRIP6], EXTSYNCIN2, ePWM X-BAR, CLB X-BAR, Output X-
INPUT7	ECAP1
INPUT8	ECAP2
INPUT9	ECAP3
INPUT10	ECAP4
INPUT11	ECAP5
INPUT12	ECAP6
INPUT13	XINT4
INPUT14	XINT5

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

Output X-BAR

The crossbars (X-BARs) provides flexibility to connect device inputs, outputs, and internal resources in a variety of configurations using Input X-BAR, Output X-BAR, and ePWM X-BAR.

The F2807x, F2837x, F2838x, F28002x, F28004x and F28003x processors support Output X-BAR. For more information, refer to TI Technical Reference Manual of there respective processors.

Output X-BAR takes signals from inside the device and brings them out to a GPIO. The Output X-BAR contains eight outputs. The signals which is passed to the GPIO comes from the Multiplexer(MUX). Each output has 32 MUX and you can select one signal per MUX.

OUPUT# MUX select

Select the MUX to map the signal to the MUX output. OUTPUT# MUX select value ranges based on the processor selected.

Selecting `Disable all` will indicate that all MUXes are disabled and the Output X-BAR# is not configured.

Note OUTPUT# MUX select will not have MUX entries whose inputs are all reserved.

Select MUX input

Select the signal to the MUX selected in `OUTPUT# MUX select`.

Select the input signals for the MUX which is sent to the GPIO. You can select one signal per MUX. The input signal to the MUX varies based on the MUX selected and processor.

The following table lists the OUTPUT MUX select and Select MUX input for C28x processor F2838x. The row headers 0-3 represent the **Select MUX input** and column headers 0-31 represent the **OUTPUT MUX select**.

Output X-BAR Mux Configuration Table - F2838x

Select MUX INPUT → / OUTPUT# MUX select ↓	0	1	2
0	CMPSS1.CTRIPOUTH	CMPSS1.CTRIPOUTH_OR_ CTRIPOUTL	ADCAEVT1
1	CMPSS1.CTRIPOUTL	INPUTXBAR1	CLB1_4.1
2	CMPSS2.CTRIPOUTH	CMPSS2.CTRIPOUTH_OR_ CTRIPOUTL	ADCAEVT2
3	CMPSS2.CTRIPOUTL	INPUTXBAR2	CLB1_5.1
4	CMPSS3.CTRIPOUTH	CMPSS3.CTRIPOUTH_OR_ CTRIPOUTL	ADCAEVT3
5	CMPSS3.CTRIPOUTL	INPUTXBAR3	CLB2_4.1
6	CMPSS4.CTRIPOUTH	CMPSS4.CTRIPOUTH_OR_ CTRIPOUTL	ADCAEVT4
7	CMPSS4.CTRIPOUTL	INPUTXBAR4	CLB2_5.1
8	CMPSS5.CTRIPOUTH	CMPSS5.CTRIPOUTH_OR_ CTRIPOUTL	ADCBEVT1
9	CMPSS5.CTRIPOUTL	INPUTXBAR5	CLB3_4.1
10	CMPSS6.CTRIPOUTH	CMPSS6.CTRIPOUTH_OR_ CTRIPOUTL	ADCBEVT2
11	CMPSS6.CTRIPOUTL	INPUTXBAR6	CLB3_5.1
12	CMPSS7.CTRIPOUTH	CMPSS7.CTRIPOUTH_OR_ CTRIPOUTL	ADCBEVT3
13	CMPSS7.CTRIPOUTL	ADCSOCA	CLB4_4.1
14	CMPSS8.CTRIPOUTH	CMPSS8.CTRIPOUTH_OR_ CTRIPOUTL	ADCBEVT4
15	CMPSS8.CTRIPOUTL	ADCSOCB	CLB4_5.1
16	SD1FLT1.COMPH	SD1FLT1.COMPH_OR_ COMPL	Reserved
17	SD1FLT1.COMPL	Reserved	CLB5_4.1
18	SD1FLT2.COMPH	SD1FLT2.COMPH_OR_ COMPL	Reserved
19	SD1FLT2.COMPL	Reserved	CLB5_5.1
20	SD1FLT3.COMPH	SD1FLT3.COMPH_OR_ COMPL	Reserved
21	SD1FLT3.COMPL	Reserved	CLB6_4.1
22	SD1FLT4.COMPH	SD1FLT4.COMPH_OR_ COMPL	Reserved
23	SD1FLT4.COMPL	INPUTXBAR10	CLB6_5.1
24	SD2FLT1.COMPH	SD2FLT1.COMPH_OR_ COMPL	Reserved
25	SD2FLT1.COMPL	Reserved	Reserved
26	SD2FLT2.COMPH	SD2FLT2.COMPH_OR_ COMPL	Reserved
27	SD2FLT2.COMPL	Reserved	ERRORSTS.
28	SD2FLT3.COMPH	SD2FLT3.COMPH_OR_ COMPL	XCLKOUT
29	SD2FLT3.COMPL	Reserved	Reserved
30	SD2FLT4.COMPH	SD2FLT4.COMPH_OR_ COMPL	Reserved

Select MUX INPUT → / OUTPUT# MUX select ↓	0	1	2
31	SD2FLT4.COMPL	Reserved	Reserved

Output X-BAR Mux Configuration Table - F28004x

Select MUX INPUT → / OUTPUT# MUX select ↓	0	1	2	3
0	CMPSS1.CTRIPOUTH	CMPSS1.CTRIPOUTH_OR_CTRIPOUTL	ADCAEVT1	ECAP1OUT
1	CMPSS1.CTRIPOUTL	INPUTXBAR1	CLB1_OUT4	ADCCEVT1
2	CMPSS2.CTRIPOUTH	CMPSS2.CTRIPOUTH_OR_CTRIPOUTL	ADCAEVT2	ECAP2OUT
3	CMPSS2.CTRIPOUTL	INPUTXBAR2	CLB1_OUT5	ADCCEVT2
4	CMPSS3.CTRIPOUTH	CMPSS3.CTRIPOUTH_OR_CTRIPOUTL	ADCAEVT3	ECAP3OUT
5	CMPSS3.CTRIPOUTL	INPUTXBAR3	CLB2_OUT4	ADCCEVT3
6	CMPSS4.CTRIPOUTH	CMPSS4.CTRIPOUTH_OR_CTRIPOUTL	ADCAEVT4	ECAP4OUT
7	CMPSS4.CTRIPOUTL	INPUTXBAR4	CLB2_OUT5	ADCCEVT4
8	CMPSS5.CTRIPOUTH	CMPSS5.CTRIPOUTH_OR_CTRIPOUTL	ADCBEVT1	ECAP5OUT
9	CMPSS5.CTRIPOUTL	INPUTXBAR5	CLB3_OUT4	Reserved
10	CMPSS6.CTRIPOUTH	CMPSS6.CTRIPOUTH_OR_CTRIPOUTL	ADCBEVT2	ECAP6OUT
11	CMPSS6.CTRIPOUTL	INPUTXBAR6	CLB3_OUT5	Reserved
12	CMPSS7.CTRIPOUTH	CMPSS7.CTRIPOUTH_OR_CTRIPOUTL	ADCBEVT3	ECAP7OUT
13	CMPSS7.CTRIPOUTL	ADCSOCAO	CLB4_OUT4	Reserved
14	Reserved	Reserved	ADCBEVT4	EXTSYNCOUT
15	Reserved	ADCSOCBO	CLB4_OUT5	Reserved
16	SD1FLT1.COMPH	SD1FLT1.COMPH_OR_COMPL	Reserved	Reserved
17	SD1FLT1.COMPL	Reserved	Reserved	CLAHALT

Select MUX INPUT → / OUTPUT# MUX select ↓	0	1	2	3
18	SD1FLT2.COMPH	SD1FLT2.COMPH_OR_COMPL	Reserved	Reserved
19	SD1FLT2.COMPL	Reserved	Reserved	Reserved
20	SD1FLT3.COMPH	SD1FLT3.COMPH_OR_COMPL	Reserved	Reserved
21	SD1FLT3.COMPL	Reserved	Reserved	Reserved
22	SD1FLT4.COMPH	SD1FLT4.COMPH_OR_COMPL	Reserved	Reserved
23	SD1FLT4.COMPL	Reserved	Reserved	Reserved

Output X-BAR Mux Configuration Table - F2807x and F2837x

Select MUX INPUT → / OUTPUT# MUX select ↓	0	1	2
0	CMPSS1.CTRIPOUTH	CMPSS1.CTRIPOUTH_OR_CTRIPOUTL	ADCAEV
1	CMPSS1.CTRIPOUTL	INPUTXBAR1	CLB1_O
2	CMPSS2.CTRIPOUTH	CMPSS2.CTRIPOUTH_OR_CTRIPOUTL	ADCAEV
3	CMPSS2.CTRIPOUTL	INPUTXBAR2	CLB1_O
4	CMPSS3.CTRIPOUTH	CMPSS3.CTRIPOUTH_OR_CTRIPOUTL	ADCAEV
5	CMPSS3.CTRIPOUTL	INPUTXBAR3	CLB2_O
6	CMPSS4.CTRIPOUTH	CMPSS4.CTRIPOUTH_OR_CTRIPOUTL	ADCAEV
7	CMPSS4.CTRIPOUTL	INPUTXBAR4	CLB2_O
8	CMPSS5.CTRIPOUTH	CMPSS5.CTRIPOUTH_OR_CTRIPOUTL	ADCBEV
9	CMPSS5.CTRIPOUTL	INPUTXBAR5	CLB3_O
10	CMPSS6.CTRIPOUTH	CMPSS6.CTRIPOUTH_OR_CTRIPOUTL	ADCBEV
11	CMPSS6.CTRIPOUTL	INPUTXBAR6	CLB3_O
12	CMPSS7.CTRIPOUTH	CMPSS7.CTRIPOUTH_OR_CTRIPOUTL	ADCBEV
13	CMPSS7.CTRIPOUTL	ADCSOCAO	CLB4_O
14	CMPSS8.CTRIPOUTH	CMPSS8.CTRIPOUTH_OR_CTRIPOUTL	ADCBEV
15	CMPSS8.CTRIPOUTL	ADCSOCBO	CLB4_O
16	SD1FLT1.COMPH	SD1FLT1.COMPH_OR_COMPL	Reserve
17	SD1FLT1.COMPL	Reserved	Reserve
18	SD1FLT2.COMPH	SD1FLT2.COMPH_OR_COMPL	Reserve
19	SD1FLT2.COMPL	Reserved	Reserve
20	SD1FLT3.COMPH	SD1FLT3.COMPH_OR_COMPL	Reserve
21	SD1FLT3.COMPL	Reserved	Reserve
22	SD1FLT4.COMPH	SD1FLT4.COMPH_OR_COMPL	Reserve
23	SD1FLT4.COMPL	Reserved	Reserve
24	SD2FLT1.COMPH	SD2FLT1.COMPH_OR_COMPL	Reserve
25	SD2FLT1.COMPL	Reserved	Reserve
26	SD2FLT2.COMPH	SD2FLT2.COMPH_OR_COMPL	Reserve
27	SD2FLT2.COMPL	Reserved	Reserve
28	SD2FLT3.COMPH	SD2FLT3.COMPH_OR_COMPL	Reserve
29	SD2FLT3.COMPL	Reserved	Reserve
30	SD2FLT4.COMPH	SD2FLT4.COMPH_OR_COMPL	Reserve

Select MUX INPUT → / OUTPUT# MUX select ↓	0	1	2
31	SD2FLT4.COMPL	Reserved	Reserve

Note Ensure the selected MUX input peripheral is enabled.

OUPUT# MUX (MUX 0 -> 31)

Indicates the input signal selected for each output# MUX. For example, XXXX1XXXXXXXXXXXXXXXXXXXXXXXX indicates that input signal 1 was selected for MUX 4. X indicates that the MUX is disabled and no signal from the MUX will be sent to the Output X-BAR output.

All the signals which are selected will be logically OR'd and sent to the output signal on the GPIO pin.

RESET OUPUT# MUX

Resets the signal selection for the MUX done so far.

Resets the **OUTPUT# MUX (MUX 0->31)** and **Select MUX input** inputs.

OUPUT# pin assignment

Select the GPIO pin to which the selected signals will be passed to. All signals from the MUXes which are enabled will be logically OR'd before being passed on to the respective OUTPUTx signal on the GPIO pin.

Enable OUTPUT# latch

Enable the output latch to latch the output signal on the GPIO pin. Latch signal has to be cleared manually.

Invert OUTPUT#

Select to invert the output signal to the GPIO pin.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

CLB X-BAR

The CLB X-BAR brings signals to the CLB modules. The CLB X-BAR has eight outputs which are routed to each CLB module.

AUXSIG# MUX select

Select the MUX to map the signal to the MUX AUXSIG#. AUXSIG# (# can take values 0 to 7)

You can select up to one signal per mux (maximum available up to of 31 muxes) for each AUXSIG# output. AUXSIG# MUX select values are based on the processor selected.

Selecting `Disable all` will indicate that all MUXes are disabled and the CLB X-BAR# is not configured.

Note AUXSIG# MUX select will not have MUX entries whose inputs are all reserved.

Select MUX input

Select the signal to the MUX selected in `AUXSIG# MUX select`. Ensure the selected MUX input peripheral is enabled and utilized.

Select the input signals for the MUX which is sent to the CLB. You can select one signal per MUX. The input signal to the MUX varies based on the MUX selected and processor.

The following table lists the AUXSIG# MUX select and Select MUX input for C28x processor F2838x. The row headers 0-3 represent the **Select MUX input** and column headers 0-31 represent the **AUXSIG# MUX select**.

CLB X-BAR Mux Configuration Table - F2838x

Select MUX INPUT → / AUXSIG# MUX select ↓	0	1	2
0	CMPSS1.CTRIPH	CMPSS1.CTRIPH_OR_CTRIP	ADCAEVT1
1	CMPSS1.CTRIPL	INPUTXBAR1	CLB1_OUT4
2	CMPSS2.CTRIPH	CMPSS2.CTRIPH_OR_CTRIP	ADCAEVT2
3	CMPSS2.CTRIPL	INPUTXBAR2	CLB1_OUT5
4	CMPSS3.CTRIPH	CMPSS3.CTRIPH_OR_CTRIP	ADCAEVT3
5	CMPSS3.CTRIPL	INPUTXBAR3	CLB2_OUT4
6	CMPSS4.CTRIPH	CMPSS4.CTRIPH_OR_CTRIP	ADCAEVT4
7	CMPSS4.CTRIPL	INPUTXBAR4	CLB2_OUT5
8	CMPSS5.CTRIPH	CMPSS5.CTRIPH_OR_CTRIP	ADCBEVT1
9	CMPSS5.CTRIPL	INPUTXBAR5	CLB3_OUT4
10	CMPSS6.CTRIPH	CMPSS6.CTRIPH_OR_CTRIP	ADCBEVT2
11	CMPSS6.CTRIPL	INPUTXBAR6	CLB3_OUT5
12	CMPSS7.CTRIPH	CMPSS7.CTRIPH_OR_CTRIP	ADCBEVT3
13	CMPSS7.CTRIPL	ADCSOCA	CLB4_OUT4
14	CMPSS8.CTRIPH	CMPSS7.CTRIPH_OR_CTRIP	ADCBEVT4
15	CMPSS8.CTRIPL	ADCSOCB	CLB4_OUT5
16	SD1FLT1.COMPH	SD1FLT1.COMPH_OR_ COMPL	SD1FLT1.CO
17	SD1FLT1.COMPL	INPUTXBAR7	CLB5_OUT4
18	SD1FLT2.COMPH	SD1FLT2.COMPH_OR_ COMPL	SD1FLT2.CO
19	SD1FLT2.COMPL	INPUTXBAR8	CLB5_OUT5
20	SD1FLT3.COMPH	SD1FLT3.COMPH_OR_ COMPL	SD1FLT3.CO
21	SD1FLT3.COMPL	INPUTXBAR9	CLB6_OUT4
22	SD1FLT4.COMPH	SD1FLT4.COMPH_OR_ COMPL	SD1FLT4.CO
23	SD1FLT4.COMPL	INPUTXBAR10	CLB6_OUT5
24	SD2FLT1.COMPH	SD2FLT1.COMPH_OR_ COMPL	SD2FLT1.CO
25	SD2FLT1.COMPL	INPUTXBAR11	MCANA.FEV
26	SD2FLT2.COMPH	SD2FLT2.COMPH_OR_ COMPL	SD2FLT2.CO
27	SD2FLT2.COMPL	INPUTXBAR12	MCANA.FEV
28	SD2FLT3.COMPH	SD2FLT3.COMPH_OR_ COMPL	SD2FLT3.CO
29	SD2FLT3.COMPL	INPUTXBAR13	MCANA.FEV
30	SD2FLT4.COMPH	SD2FLT4.COMPH_OR_ COMPL	SD2FLT4.CO

Select MUX INPUT → / AUXSIG# MUX select ↓	0	1	2
31	SD2FLT4.COMPL	INPUTXBAR14	EMAC.PPS0

CLB X-BAR Mux Configuration Table - F28003x

Select MUX INPUT → / AUXSIG# MUX select ↓	0	1	2
0	CMPSS1.CTRIPH	CMPSS1.CTRIPH_OR_CTRIP	ADCAEVT1
1	CMPSS1.CTRIPL	INPUTXBAR1	CLB1_OUT4
2	CMPSS2.CTRIPH	CMPSS2.CTRIPH_OR_CTRIP	ADCAEVT2
3	CMPSS2.CTRIPL	INPUTXBAR2	CLB1_OUT5
4	CMPSS3.CTRIPH	CMPSS3.CTRIPH_OR_CTRIP	ADCAEVT3
5	CMPSS3.CTRIPL	INPUTXBAR3	CLB2_OUT4
6	CMPSS4.CTRIPH	CMPSS4.CTRIPH_OR_CTRIP	ADCAEVT4
7	CMPSS4.CTRIPL	INPUTXBAR4	CLB2_OUT5
8	Reserved	Reserved	ADCBEVT1
9	Reserved	INPUTXBAR5	CLB3_OUT4
10	Reserved	Reserved	ADCBEVT2
11	Reserved	INPUTXBAR6	CLB3_OUT5
12	Reserved	Reserved	ADCBEVT3
13	Reserved	ADCSOCAO	CLB4_4
14	Reserved	Reserved	ADCBEVT4
15	Reserved	ADCSOCBO	CLB4_5
16	SD1FLT1.COMPH	SD1FLT1.COMPH_OR_ COMPL	SD1FLT1.CO
17	SD1FLT1.COMPL	INPUTXBAR7	Reserved
18	SD1FLT2.COMPH	SD1FLT2.COMPH_OR_ COMPL	SD1FLT2.CO
19	SD1FLT2.COMPL	INPUTXBAR8	Reserved
20	SD1FLT3.COMPH	SD1FLT3.COMPH_OR_ COMPL	SD1FLT3.CO
21	SD1FLT3.COMPL	INPUTXBAR9	Reserved
22	SD1FLT4.COMPH	SD1FLT4.COMPH_OR_ COMPL	SD1FLT4.CO
23	SD1FLT4.COMPL	INPUTXBAR10	Reserved
24	SD2FLT1.COMPH	SD2FLT1.COMPH_OR_ COMPL	SD2FLT1.CO
25	SD2FLT1.COMPL	INPUTXBAR11	MCANA.FEV
26	SD2FLT2.COMPH	SD2FLT2.COMPH_OR_ COMPL	SD2FLT2.CO
27	SD2FLT2.COMPL	INPUTXBAR12	MCANA.FEV
28	SD2FLT3.COMPH	SD2FLT3.COMPH_OR_ COMPL	SD2FLT3.CO
29	SD2FLT3.COMPL	INPUTXBAR13	MCANA.FEV
30	SD2FLT4.COMPH	SD2FLT4.COMPH_OR_ COMPL	SD2FLT4.CO

Select MUX INPUT → / AUXSIG# MUX select ↓	0	1	2
31	SD2FLT4.COMPL	INPUTXBAR14	ERRORSTS

CLB X-BAR Mux Configuration Table - F28004x

Select MUX INPUT → / AUXSIG# MUX select ↓	0	1	2	3
0	CMPSS1.CTRIPOUTH	CMPSS1.CTRIPOUTH_OR_CTRIPOUTL	ADCAEVT1	ECAP1OUT
1	CMPSS1.CTRIPOUTL	INPUTXBAR1	CLB1_OUT4	ADCCEVT1
2	CMPSS2.CTRIPOUTH	CMPSS2.CTRIPOUTH_OR_CTRIPOUTL	ADCAEVT2	ECAP2OUT
3	CMPSS2.CTRIPOUTL	INPUTXBAR2	CLB1_OUT5	ADCCEVT2
4	CMPSS3.CTRIPOUTH	CMPSS3.CTRIPOUTH_OR_CTRIPOUTL	ADCAEVT3	ECAP3OUT
5	CMPSS3.CTRIPOUTL	INPUTXBAR3	CLB2_OUT4	ADCCEVT3
6	CMPSS4.CTRIPOUTH	CMPSS4.CTRIPOUTH_OR_CTRIPOUTL	ADCAEVT4	ECAP4OUT
7	CMPSS4.CTRIPOUTL	INPUTXBAR4	CLB2_OUT5	ADCCEVT4
8	CMPSS5.CTRIPOUTH	CMPSS5.CTRIPOUTH_OR_CTRIPOUTL	ADCBEVT1	ECAP5OUT
9	CMPSS5.CTRIPOUTL	INPUTXBAR5	CLB3_OUT4	Reserved
10	CMPSS6.CTRIPOUTH	CMPSS6.CTRIPOUTH_OR_CTRIPOUTL	ADCBEVT2	ECAP6OUT
11	CMPSS6.CTRIPOUTL	INPUTXBAR6	CLB3_OUT5	Reserved
12	CMPSS7.CTRIPOUTH	CMPSS7.CTRIPOUTH_OR_CTRIPOUTL	ADCBEVT3	ECAP7OUT
13	CMPSS7.CTRIPOUTL	ADCSOCAO	CLB4_OUT4	Reserved
14	Reserved	Reserved	ADCBEVT4	EXTSYNCOUT
15	Reserved	ADCSOCBO	CLB4_OUT5	Reserved
16	SD1FLT1.COMPH	SD1FLT1.COMPH_OR_COMPL	Reserved	Reserved
17	SD1FLT1.COMPL	Reserved	Reserved	CLAHALT

Select MUX INPUT → AUXSIG# MUX select ↓	0	1	2	3
18	SD1FLT2.COMPH	SD1FLT2.COMPH_OR_COMPL	Reserved	Reserved
19	SD1FLT2.COMPL	Reserved	Reserved	Reserved
20	SD1FLT3.COMPH	SD1FLT3.COMPH_OR_COMPL	Reserved	Reserved
21	SD1FLT3.COMPL	Reserved	Reserved	Reserved
22	SD1FLT4.COMPH	SD1FLT4.COMPH_OR_COMPL	Reserved	Reserved
23	SD1FLT4.COMPL	Reserved	Reserved	Reserved

CLB X-BAR Mux Configuration Table - F28002x

Select MUX INPUT → / AUXSIG# MUX select ↓	0	1	2	3
0	CMPSS1.CTRIPOUTH	CMPSS1.CTRIPOUTH_OR_CTRIPOUTL	ADCAEVT1	ECAP1OUT
1	CMPSS1.CTRIPOUTL	INPUTXBAR1	CLB1_OUT4	ADCCEVT1
2	CMPSS2.CTRIPOUTH	CMPSS2.CTRIPOUTH_OR_CTRIPOUTL	ADCAEVT2	ECAP2OUT
3	CMPSS2.CTRIPOUTL	INPUTXBAR2	CLB1_OUT5	ADCCEVT2
4	CMPSS3.CTRIPOUTH	CMPSS3.CTRIPOUTH_OR_CTRIPOUTL	ADCAEVT3	ECAP3OUT
5	CMPSS3.CTRIPOUTL	INPUTXBAR3	CLB2_OUT4	ADCCEVT3
6	CMPSS4.CTRIPOUTH	CMPSS4.CTRIPOUTH_OR_CTRIPOUTL	ADCAEVT4	Reserved
7	CMPSS4.CTRIPOUTL	INPUTXBAR4	CLB2_OUT5	ADCCEVT4
9	Reserved	INPUTXBAR5	Reserved	Reserved
11	Reserved	INPUTXBAR6	Reserved	Reserved
13	Reserved	ADCSOCAO	Reserved	Reserved
14	Reserved	Reserved	Reserved	EXTSYNCOUT

CLB X-BAR Mux Configuration Table - F2807x/F2837xS/F2837xD

Select MUX INPUT → / AUXSIG# MUX select ↓	0	1	2	3
0	CMPSS1.CTRIPH	CMPSS1.CTRIPH_OR_CTRIPL	ADCAEVT1	ECAP1.OUT
1	CMPSS1.CTRIPL	INPUTXBAR1	CLB1_OUT4	ADCCEVT1
2	CMPSS2.CTRIPH	CMPSS2.CTRIPH_OR_CTRIPL	ADCAEVT2	ECAP2.OUT
3	CMPSS2.CTRIPL	INPUTXBAR2	CLB1_OUT5	ADCCEVT2
4	CMPSS3.CTRIPH	CMPSS3.CTRIPH_OR_CTRIPL	ADCAEVT3	ECAP3OUT
5	CMPSS3.CTRIPL	INPUTXBAR3	CLB2_OUT4	ADCCEVT3
6	CMPSS4.CTRIPH	CMPSS4.CTRIPH_OR_CTRIPL	ADCAEVT4	ECAP4.OUT
7	CMPSS4.CTRIPL	INPUTXBAR4	CLB2_OUT5	ADCCEVT4
8	CMPSS5.CTRIPH	CMPSS4.CTRIPH_OR_CTRIPL	ADCBEVT1	ECAP5.OUT
9	CMPSS5.CTRIPL	INPUTXBAR5	CLB3_OUT4	ADCCEVT1
10	CMPSS6.CTRIPH	CMPSS4.CTRIPH_OR_CTRIPL	ADCBEVT1	ECAP6.OUT
11	CMPSS6.CTRIPL	INPUTXBAR6	CLB3_OUT5	ADCCEVT2
12	CMPSS7.CTRIPH	CMPSS7.CTRIPH_OR_CTRIPL	ADCBEVT3	Reserved
13	CMPSS7.CTRIPL	ADCSOCAO	CLB4_OUT4	ADCDEVT3
14	CMPSS8.CTRIPH	CMPSS8.CTRIPH_OR_CTRIPL	ADCBEVT4	EXTSYNCOUT
15	CMPSS8.CTRIPL	ADCSOCB	CLB4_OUT5	ADCDEVT4
16	SD1FLT1.COMPH	SD1FLT1.COMPH_OR_COMPL	Reserved	Reserved
17	SD1FLT1.COMPL	Reserved	Reserved	Reserved
18	SD1FLT2.COMPH	SD1FLT2.COMPH_OR_COMPL	Reserved	Reserved
19	SD1FLT2.COMPL	Reserved	Reserved	Reserved
20	SD1FLT3.COMPH	SD1FLT3.COMPH_OR_COMPL	Reserved	Reserved
21	SD1FLT3.COMPL	Reserved	Reserved	Reserved

Select MUX INPUT → / AUXSIG# MUX select ↓	0	1	2	3
22	SD1FLT4.COMPH	SD1FLT4.COMPH_OR_COMPL	Reserved	Reserved
23	SD1FLT4.COMPL	Reserved	Reserved	Reserved
24	SD2FLT1.COMPH	SD2FLT1.COMPH_OR_COMPL	Reserved	Reserved
25	SD2FLT1.COMPL	Reserved	Reserved	Reserved
26	SD2FLT2.COMPH	SD2FLT2.COMPH_OR_COMPL	Reserved	Reserved
27	SD2FLT2.COMPL	Reserved	Reserved	Reserved
28	SD2FLT3.COMPH	SD2FLT3.COMPH_OR_COMPL	Reserved	Reserved
29	SD2FLT3.COMPL	Reserved	Reserved	Reserved
30	SD2FLT4.COMPH	SD2FLT4.COMPH_OR_COMPL	Reserved	Reserved
31	SD2FLT4.COMPL	Reserved	Reserved	Reserved

For F2837x and F2807x processors CLB clock comes from ePWM clock. And for F2837xD, if both CLB and ePWM are used then they should be in same CPU.

AUXSIG# MUX (MUX 0 -> 31)

Indicates the input signal selected for each output# MUX. For example, XXXX1XXXXXXXXXXXXXXXXXXXXXXXX indicates that input signal 1 was selected for MUX 4. X indicates that the MUX is disabled and no signal from the MUX will be sent to the CLB X-BAR.

All the signals which are selected will be logically OR'd before being passed on to the respective AUXSIG#x signal on the CLB.

RESET AUXSIG# MUX

Resets the signal selection for the MUX done so far.

Resets the **AUXSIG# MUX (MUX 0->31)** and **Select MUX input** inputs.

Invert AUXSIG#

Select to invert the auxiliary signal on the CLB.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

CLB

The configurable logic block (CLB) is a collection of configurable blocks that can be inter-connected using software to implement custom digital logic functions. The CLB is able to enhance existing peripherals through a set of crossbar interconnections, which provide a high level of connectivity to existing control peripherals such as enhanced pulse width modulators (ePWM), enhanced capture modules (eCAP), and enhanced quadrature encoder pulse modules (eQEP). The crossbars also allow the CLB to be connected to external GPIO pins. In this way, the CLB can be configured to interact with device peripherals to perform small logical functions such as simple PWM generators, or to implement custom serial data exchange protocols.

Note

- For F2838x processor the CLB tile clock is configured in SYNC mode and is derived as SYSCLKOUT/EPWMCLKDIV (Achievable SYSCLKOUT in MHz/EPWM clock divider) should be always less than 150MHz.
- For F28002x and F28004x processors the CLB tile clock is configured in SYNC mode and should be less than 100 MHz.
- For F2837x and F2807x processors CLB clock comes from ePWM clock. And for F2837xD, if both CLB and ePWM are used then they should be in same CPU.

Enable CLB Tile #

Select this option to enable the CLB tile. The number of CLB tile available based on the hardware board.

This option will enable you to configure the input and output signals to the CLB tile, the logic for the CLB tile can be implemented in TI Code Composer Studio CLB tool with system configuration file and the generated clb_config.h and clb_config.c file can be integrated with the parameters **CLB configuration header file (clb_config.h)** and **CLB configuration source file (clb_congif.c)**.

Tile # Name

Specify the tile name. Ensure that the tile name is same as the name mentioned in the CLB Tool.

Add the tile# name for the given CLB tile. This tile name will be used to generate the required function declaration and calling the function for CLB tile configuration. The tile names entered here should be an exact match with the tile name set in CLB tool system configuration file In TI CCS used to generate the clb_config.c and clb_config.h file. Mismatch may result in build failure or in incorrect configuration of CLB tiles. For example, if the tile name is **TILE1**, the function present in clb_config.c and clb_config.h is **initTILE1** and the function called during code generation is also **initTILE1**.

IN# mux selection

Configure the signal source type for the IN# mux for CLB# tile. The # corresponds to the values 0 to 7. The type of signal can be global inputs, local inputs and GPREG.

Ensure corresponding X-BAR configurations are done to route the signals properly for the CLB tile inputs

Input

Configure the peripheral signal as input to the CLB tile. Depending upon the signal type selected for IN# mux selection, different input signals can be configured. For GPREG IN# mux selection, the input values can be either 0 or 1.

Input filtering

Configure the type of input filtering for the signal type **Global inputs** and **Local inputs**. This option will be disabled for **GPREG** as it is not applicable. The values can be No filtering, Rising edge detect, falling edge detect and any edge detect.

Enable sync

Configure the synchronization option (SYNC) for the **Global inputs** and **Local inputs** IN# mux selection only. Enabling this option will synchronize the signal with respective to clock. This option is not applicable for **GPREG** type and will be disabled for the same.

Note For the signals which are specified as **ASYNC** in TI reference manuals, the input filter synchronizer must be enabled explicitly.

Route OUT# signal to

Use this option to route the CLB output signal to the peripheral instead of the default peripheral signal. Each CLB output signal passes through an external multiplexer that intersects a specific peripheral signal. If the options in this parameter is enabled it will route the CLB output for the specific peripheral instead of the original peripheral signal.

Each output signal will be replicated and can be routed to different peripheral.

For example,

1) In TI F2838x (C28x) and TI F28002x, for CLB tile1 Out0 is replicated as Out0_0, Out0_1 (also represented as Out8), Out0_2 (also represented as Out16) and Out0_3 (also represented as Out24). Out0_0 is routed to ePWM1A, OUT0_1 is routed to eQEP.QCLK, Out0_2 is routed to Global mux and Out0_3 is routed to SPI_A.CLKIN. So same output signal is routed to 4 different peripherals.

2) In TI Piccolo F28004x, for CLB tile1 Out0 is replicated as Out0_0, Out0_1 (also represented as Out8), Out0_2 (also represented as Out16) and Out0_3 (also represented as Out24). Out0_0 is routed to ePWM1A, OUT0_1 is routed to eQEP.QCLK, Out0_2 is routed to Global mux. So same output signal can be routed to 3 peripherals.

Similarly all the out# signals are replicated. these options are provided as check boxes with respective to the peripheral points where the CLB output signals are used as replacement to original peripheral signal.

Note Global Mux option refers to the Global Input signal mux to each CLB Tile.

CLB configuration header file (clb_config.h)

Provide the paths for the CLB configuration header file clb_config.h. This file holds the required function declarations and headers used to configure the CLB tile. This can be generated using system configuration in CLB tool using TI Code composer studio. Ensure the tile names selected in CLB tool matches with the tile name provided in the parameter **Tile# name**.

You can provide the file path relative to the model path.

CLB configuration source file (clb_config.c)

Provide the paths for the CLB configuration source file clb_config.c. This file holds the required function definitions used to configure the CLB tile. This can be generated using system configuration in CLB tool using TI Code composer studio. Ensure the tile names selected in CLB tool matches with the tile name provided in the parameter **Tile# name**.

You can provide the file path relative to the model path.

Browse

Click this button to browse the path for the file selection.

Edit

Click this button to open the existing file for editing in MATLAB editor.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

ARM Cortex-M4 - MCAN

The Controller Area Network (CAN) is a serial communications protocol that efficiently supports distributed real time control with a high level of security.

The MCAN module supports both Classic CAN and CAN FD (CAN with flexible data-rate) specifications. The CAN FD feature allows high throughput and increased payload per data frame. The Classic CAN and CAN FD devices can coexist on the same network without any conflict.

Protocol mode

Select the CAN type. CAN type can either be `Classic CAN` or `CAN-FD`.

MCAN module clock frequency (=connectivity manager (ARM Cortex-M) clock) in MHz

Displays the MCAN module clock frequency (=connectivity manager (ARM Cortex-M) clock) in MHz.

MCAN bit clock frequency (MCAN module clock freq/4) in MHz

Displays the MCAN bit clock frequency (MCAN module clock freq/4) in MHz.

Nominal bit rate prescaler (NBRP: 1 to 512)

Nominal Bit Rate Prescaler(NBRP). The value by which the oscillator frequency is divided for generating the bit time quanta. The bit time is built up from a multiple of this quanta. Valid values for the bit rate prescaler are 1 to 512.

Nominal time segment 1 (NTSEG1: 2 to 256)

Nominal Time Segment(NTGEG) before sample point. Valid values are 2 to 256.

Nominal time segment 2 (NTSEG2: 2 to 128)

Nominal Time Segment(NTSEG) after sample point. Valid values are 2 to 128.

Closest achievable nominal baud rate (MCAN bit clock/NBRP/(NTSEG1+NTSEG2)) in bits/sec

Closest achievable MCAN baud rate calculated based on parameters and given formula.

Nominal re-synchronization jump width (NSJW: 1 to 128)

Nominal Resynchronization Jump Width (NSJW). Valid values are 1 to 128.

Enable bit rate switching

Enables bit rate switching between nominal bit rate and data bit rate.

This parameter is available only for CAN-FD protocol mode.

Data bit rate prescaler (DBRP: 1 to 32)

Data Bit Rate Prescaler (DBRP). The value by which the oscillator frequency is divided for generating the bit time quanta. The bit time is built up from a multiple of this quanta. Valid values for the bit rate prescaler are 1 to 32.

This parameter is available only for **CAN-FD** protocol mode and if **Enable bit rate switching** parameter is enabled.

Data time segment 1 (DTSEG1: 1 to 32)

Data Time Segment (DTSEG1) before sample point . Valid values are 1 to 32.

This parameter is available only for **CAN-FD** protocol mode and if **Enable bit rate switching** parameter is enabled.

Data time segment 2 (DTSEG2: 1 to 16)

Data Time Segment (DTSEG2) after sample point (DTSEG2). Valid values are 1 to 16.

This parameter is available only for **CAN-FD** protocol mode and if **Enable bit rate switching** parameter is enabled.

Data baud rate (MCAN bit clock/DBRP/(DTSEG1+DTSEG2)) in bits/sec

Closest achievable MCAN data baud rate calculated based on data parameters and given formula.

This parameter is available only for **CAN-FD** protocol mode and if **Enable bit rate switching** parameter is enabled.

Data re-synchronization jump width (DSJW: 1 to 16)

Data Resynchronization Jump Width (DSJW). Valid values are 1 to 16.

This parameter is available only for **CAN-FD** protocol mode and if **Enable bit rate switching** parameter is enabled.

Mode

Select the operating mode for MCAN. Mode can be `Normal`, `Internal loopback` or `Bus monitoring`.

Pin assignment(Tx)

Select a GPIO pin for the MCAN data transmission.

Pin assignment(Rx)

Select a GPIO pin for the MCAN data reception.

Transmission mode

Select the mode of transmission. Transmission mode can either be `FIFO` or `Queue`.

Enable blocking mode for Rx FIFO 0

Enable blocking mode for FIFO 0 data reception.

Enable blocking mode for Rx FIFO 1

Enable blocking mode for FIFO 1 data reception.

Update global filter configuration

Enable this parameter to update standard and extended filter IDs.

Reject remote frames standard

Rejects all remote frames with 11-bit standard IDs when enabled else the remote frames will be filtered as per the settings from **Update standard filter elements**.

Reject remote frames extended

Rejects all remote frames with 29-bit extended IDs when enabled else the remote frames will be filtered as per the settings from **Update extended filter elements**.

Non-matching frames standard

Defines how received messages with 11-bit standard IDs that do not match any element from **Update standard filter elements** are treated. Non-matching frames standard can be:

- Accept in Rx FIFO 0

- Accept in Rx FIFO 1
- Reject

Non-matching frames extended

Defines how received messages with 29-bit extended IDs that do not match any element from**Update extended filter elements** are treated. Non-matching frames extended can be:

- Accept in Rx FIFO 0
- Accept in Rx FIFO 1
- Reject

Update standard filter elements

Select this parameter to update the standard 11bit ID filter elements parameters. Up to 128 filter elements can be configured for 11-bit standard IDs.

Select standard filter

Select the standard filter element number. Filter number ranges between 0 to 127.

All enabled filter elements are used for acceptance filtering of standard frames. Acceptance filtering stops at the first matching enabled filter element or when the end of the filter list is reached.

Filter # configuration

Select one of the following parameter for the standard filter element selected using **Select standard filter** parameter.

- Disable filter element
- Store in Rx FIFO 0 if filter matches
- Store in Rx FIFO 1 if filter matches
- Reject ID if filter matches
- Set priority if filter matches
- Set priority and store in FIFO 0 if filter matches
- Set priority and store in FIFO 1 if filter matches
- Store into Rx Buffer

Filter # type (filter type will be ignored if filter configuration is stored into Rx buffer)

Select one of the following parameter for the standard filter type selected using **Select standard filter** parameter.

- Range filter (from ID1 to ID2) - For range filter the ID2 >= ID1
- Dual filter - Two dedicated message IDs is provided. To filter one specific message ID, the ID2 = ID1
- Classic ID and mask filter (ID1 = filter, ID2 = mask) - A 0 bit at the filter mask (SFID2) will mask out the corresponding bit position of the configured Message ID filter (SFID1) and the value of the received Message ID at that bit position is not relevant for acceptance filtering. Only those bits of the received Message ID where the corresponding mask bits are 1 are relevant for acceptance filtering.

Filter # ID1

Specify the standard Filter ID 1.

Filter # ID2 (provide buffer number if filter configuration is stored into Rx buffer)

Specify the standard Filter ID 2. Provide the buffer number if filter configuration is stored into Rx buffer.

Update extended filter elements

Select this parameter to update the extended filter elements parameters.

Select extended filter

Select the extended filter element number. Filter number ranges between 0 to 63.

All enabled filter elements are used for acceptance filtering of extended frames. Acceptance filtering stops at the first matching enabled filter element or when the end of the filter list is reached.

Filter # configuration

Select one of the following parameters for the extended filter element selected using **Select extended filter** :

- Disable filter element
- Store in Rx FIFO 0 if filter matches
- Store in Rx FIFO 1 if filter matches
- Reject ID if filter matches
- Set priority if filter matches
- Set priority and store in FIFO 0 if filter matches
- Set priority and store in FIFO 1 if filter matches
- Store into Rx Buffer

Filter # type (filter type will be ignored if filter configuration is stored into Rx buffer)

Extended filter type.

Select one of the standard filter type:

- Range filter (from ID1 to ID2) - For range filter the ID2 >= ID1
- Dual filter - Two dedicated message IDs is provided. To filter one specific message ID, the ID2 = ID1
- Classic ID and mask filter (ID1 = filter, ID2 = mask) - A 0 bit at the filter mask (EFID2) will mask out the corresponding bit position of the configured Message ID filter (EFID1) and the value of the received Message ID at that bit position is not relevant for acceptance filtering. Only those bits of the received Message ID where the corresponding mask bits are 1 are relevant for acceptance filtering.

Filter # ID1

Specify the first ID of extended ID filter element.

Filter # ID2 (provide buffer number if filter configuration is stored into Rx buffer)

Specify the second ID of extended ID filter element. Provide buffer number if filter configuration is stored into Rx buffer.

Display configured extended and standard filters elements in command window

Click on **Display configured extended and standard filters elements in command window** button to view the configured standard and extended filter elements in MATLAB command window.

Reset standard filters configurations

Click **Reset standard filters configurations** to reset the configured standard filter configurations.

Reset extended filters configurations

Click **Reset extended filters configurations** to reset the configured extended filter configurations.

Configure memory

Select to configure the memory and its parameters. Select this parameter to configure CAN FD memory parameters. This is not applicable for classic CAN as memory parameters is automatically configured.

This parameter is available only for CAN FD protocol mode.

Maximum element size in TX FIFO (in bytes)

Select the maximum data size of CAN FD message in transmit FIFO.

This parameter is available only for CAN FD protocol mode and if **Update memory configuration** is selected.

Maximum element size in RX FIFO 0 (in bytes)

Select the maximum data size of CAN FD message in receive FIFO 0.

This parameter is available only for CAN FD protocol mode and if **Update memory configuration** is selected.

Maximum element size in RX FIFO 1 (in bytes)

Select the maximum data size of CAN FD message in receive FIFO 1.

This parameter is available only for CAN FD protocol mode and if **Update memory configuration** is selected.

Maximum element size in RX buffer (in bytes)

Select the maximum data size of CAN FD message in receive buffer.

This parameter is available only for CAN FD protocol mode and if **Update memory configuration** is selected.

Number of elements in TX FIFO/Queue

Select the number of elements (data + header CAN FD message) in transmit FIFO/Queue.

This parameter is available only for CAN FD protocol mode and if **Update memory configuration** is selected.

Number of elements in RX FIFO 0

Select the number of elements (data + header CAN FD message) in receive FIFO 0.

This parameter is available only for CAN FD protocol mode and if **Update memory configuration** is selected.

Number of elements in RX FIFO 1

Select the number of elements (data + header CAN FD message) in receive FIFO 1.

This parameter is available only for CAN FD protocol mode and if **Update memory configuration** is selected.

Validate memory

Click **Validate memory** button to validate all the memory parameters configured.

This parameter is available only for CAN FD protocol mode and if **Update memory configuration** is selected.

The following table explains the memory allocation.

Memory Allocation

Section	Element size (in bytes)	Number of Elements
Standard filter	4	Number of standard filters configured in **Update standard filter elements**.
Extended filter	8	Number of standard filters configured in **Update extended filter elements**.
Tx FIFO	Header(8) + value specified in **Maximum element size in TX FIFO (in bytes)**	Number of elements in TX FIFO/Queue.
Tx event FIFO	8	32
Rx buffer	Header(8) + value specified in **Maximum element size in RX buffer (in bytes)**	Maximum buffer number configured in **Update standard filter elements** or **Update extended filter elements** configuration + 1
Rx FIFO 0	Header(8) + value specified in **Maximum element size in RX FIFO 0 (in bytes)**	**Auto Allocate**: If parameter **Number of elements in RX FIFO 0** is set to Auto allocate then it will verify if **Rx FIFO 0** is used for matching or non-matching frames and assign remaining memory if it is available. In this case minimum available memory should be 1 element size of Rx FIFO 0. **Not Auto Allocate**: Number of elements in RX FIFO 0
Rx FIFO 1	Header(8) + value specified in **Maximum element size in RX FIFO 1 (in bytes)**	**Auto Allocate**: If parameter **Number of elements in RX FIFO 1** is set to Auto allocate then it will verify if Rx FIFO 1 is used for matching or non-matching frames and assign remaining memory if it is available. In this case minimum available memory should be 1 element size of Rx FIFO 1 . **Not Auto Allocate**: Number of elements in RX FIFO 1

Section	Element size (in bytes)	Number of Elements

Note

- If parameter **Number of elements in RX FIFO 0** or **Number of elements in RX FIFO 1** is set to auto allocate it verifies if Rx FIFO 0 or Rx FIFO 1 is used for matching or non-matching frames and assign remaining available memory to their respective FIFO.
- If only one FIFO is used then entire remaining memory is assigned it or it is distributed equally between both the FIFO's. In this case minimum available memory will be 1 element size of Rx FIFO 0 or Rx FIFO 1.

Configure receive interrupt sources

Select to configure the receive interrupt parameters such as buffer and FIFO messages.

Configure RX buffer interrupt sources

Select to configure the receive buffer interrupt.

To select this parameter, select the **Configure receive interrupt sources**.

Dedicated RX buffer message

Select the dedicated interrupt line for receive buffer message.

To select this parameter, select the **Configure RX buffer interrupt sources** .

High priority message

Select the dedicated interrupt line for high priority message.

To configure this parameter, select the **Configure receive buffer interrupt sources**.

Configure RX FIFO 0 interrupt sources

Enable to configure the receive FIFO 0 interrupt sources.

To select this parameter, select the **Configure receive interrupt sources**.

RX FIFO 0 new message

Select the interrupt line for receive FIFO 0 new message.

To configure this parameter, select the **Configure RX FIFO 0 interrupt**.

RX FIFO 0 full

Select the interrupt line for receive FIFO 0 full.

To configure this parameter, select the **Configure RX FIFO 0 interrupt**.

RX FIFO 0 message lost

Select the interrupt line for receive FIFO 0 message lost.

To configure this parameter, select the **Configure RX FIFO 0 interrupt**.

RX FIFO 0 watermark

Select the interrupt line for receive FIFO 0 watermark.

To configure this parameter, select the **Configure RX FIFO 0 interrupt**.

Configure RX FIFO 1 interrupt sources

Enable to configure the receive FIFO 1 interrupt sources.

To configure this parameter, select the **Configure RX FIFO 0 interrupt**.

RX FIFO 1 new message

Select the interrupt line for receive FIFO 1 new message.

To select this parameter, select the **Configure RX FIFO 1 interrupt sources**.

RX FIFO 1 full

Select the interrupt line for receive FIFO 1 full.

To select this parameter, select the **Configure RX FIFO 1 interrupt sources**.

RX FIFO 1 message lost

Select the interrupt line for receive FIFO 1 message lost.

To select this parameter, select the **Configure RX FIFO 1 interrupt sources**.

RX FIFO 1 watermark

Select the interrupt line for receive FIFO 1 watermark.

To select this parameter, select the **Configure RX FIFO 1 interrupt sources**.

Configure transmit interrupt sources

Select to configure the transmit interrupt parameters such as event and FIFO.

Configure TX FIFO interrupt sources

Select to configure the transmit FIFO interrupt sources.

To select this parameter, select the **Configure transmit interrupt sources**.

Transmission complete

Select the transmission interrupt line for transfer complete.

To select this parameter, select the **Configure TX FIFO interrupt sources** .

Transmission cancellation finish

Select the transmission interrupt line for transfer cancellation finish.

To select this parameter, select the **Configure TX FIFO interrupt sources** .

TX FIFO empty

Select the transmission interrupt line for TX FIFO empty.

To select this parameter, select the **Configure TX FIFO interrupt sources** .

Configure TX event FIFO interrupt sources

Select to configure the transmit FIFO interrupt sources.

To select this parameter, select the **Configure transmit interrupt sources**.

TX event FIFO new entry

Select the transmission interrupt line for TX event FIFO new entry.

To select this parameter, select the **Configure TX event FIFO interrupt sources** .

TX event FIFO element lost

Select the transmission interrupt line for TX event FIFO element lost.

To select this parameter, select the **Configure TX event FIFO interrupt sources** .

TX event FIFO full

Select the transmission interrupt line for TX event FIFO full.

To select this parameter, select the **Configure TX event FIFO interrupt sources** .

TX event FIFO watermark

Select the transmission interrupt line for TX event FIFO watermark.

To select this parameter, select the **Configure TX event FIFO interrupt sources** .

Configure other interrupt sources

Select to configure other interrupt sources.

Timestamp wraparound

Select the interrupt line for timestamp wraparound interrupt.

To select this parameter, select the **Configure other interrupt sources**.

Timeout occurred

Select the interrupt line for timeout occurred interrupt.

To select this parameter, select the **Configure other interrupt sources**.

Error logging overflow

Select the interrupt line for error logging overflow interrupt.

To select this parameter, select the **Configure other interrupt sources**.

Warning status

Select the interrupt line for warning status interrupt.

To select this parameter, select the **Configure other interrupt sources**.

Watchdog event

Select the interrupt line for watchdog event interrupt.

To select this parameter, select the **Configure other interrupt sources**.

Data protocol error

Select the interrupt line for data protocol error interrupt.

To select this parameter, select the **Configure other interrupt sources**.

Message RAM access failure

Select the interrupt line for message RAM access failure interrupt.

To select this parameter, select the **Configure other interrupt sources**.

Bit error uncorrected

Select the interrupt line for bit error uncorrected interrupt.

To select this parameter, select the **Configure other interrupt sources**.

Error passive status

Select the interrupt line for error passive status interrupt.

To select this parameter, select the **Configure other interrupt sources**.

Bus off status

Select the interrupt line for bus off status interrupt.

To select this parameter, select the **Configure other interrupt sources**.

Arbitration protocol error

Select the interrupt line for arbitration protocol error interrupt.

To select this parameter, select the **Configure other interrupt sources**.

Reserved address access

Select the interrupt line for reserved address access interrupt.

To select this parameter, select the **Configure other interrupt sources**.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

External Mode

Communication interface

Select the type of communication interface to run your model in external mode.

Default: `XCP on Serial`

- XCP on Serial
- XCP on TCP/IP

When you select **Communication interface** as `XCP on Serial`, consider the following limitations:

Note

- Your antivirus software or firewall might block UDP/TCP traffic. Configure the software to allow traffic from a specific IP port number.
- Due to RAM memory limitations on the F2838x(ARM Cortex-M4), loading application to RAM is not supported for this block.
- CPU Timer 2 of F2838x Arm Cortex-M core (Connectivity Manager) provides time base to lwIP stack. It is configured to trigger an interrupt every 1 ms. This timer should not be re-configured if Ethernet blocks are being used in the model. If the corresponding interrupt is armed through Hardware Interrupt block, it will run the interrupt handler every 1 ms.

Serial port in MATLAB preferences

Lists the COM port entries available in the device manager and saved COM port in MATLAB preferences of the target hardware. You can select the required COM port from the drop-down.

The COM port value will be saved as MATLAB preference for a given target instead of model. For example, if you choose a same target for a new model, the serial port saved in MATLAB preferences will be selected automatically.

Click refresh to see the latest value serial port value stored in MATLAB preference for the given hardware board and updated list of serial ports available from device manager.

You can also set the serial port in MATLAB preferences for the given hardware board using the MATLAB command:

```
codertarget.tic2000.setSerialPortPreferences(Hardware board, CPU value, Serial port)
```

Here CPU value is optional argument.

To know the COM port used by the target hardware on your computer, see “Serial Configuration for External Mode and PIL” on page 1-67.

Refresh

Lists the new COM port entries available in your device manager.

Click refresh to see the latest value serial port value stored in MATLAB preference for the given hardware board and updated list of serial ports available from device manager.

Verbose

Select this to view the external mode execution progress and updates in the Diagnostic Viewer or in the MATLAB command window.

Set logging buffer size automatically

Select this to automatically set the number of bytes to preallocate for the buffer in the hardware during simulation. By default, the **Set logging buffer size automatically** parameter is selected. If you clear this parameter, **Logging buffer size (in bytes)** parameter becomes available, where you can manually specify the memory buffer size for XCP-based External mode simulation.

Maximum number of contiguous samples

Specify the maximum number of contiguous samples to be packed in a single packet. Memory consumed on the target will increase with increasing value as contiguous needs to be stored on target before transmitting.

See Also

“Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2 | “Serial Configuration for External Mode and PIL” on page 1-67

PIL

PIL Communication interface

Select the type of Processor-In-Loop (PIL) communication interface to run your model in external mode.

Default: `Serial`

- TCP/IP

When you select **PIL Communication interface** as `Serial`, consider the following limitations:

Note

- Your antivirus software or firewall might block UDP/TCP traffic. Configure the software to allow traffic from a specific IP port number.
- Due to RAM memory limitations on the F2838x(ARM Cortex-M4), loading application to RAM is not supported for this block.
- CPU Timer 2 of F2838x Arm Cortex-M core (Connectivity Manager) provides time base to lwIP stack. It is configured to trigger an interrupt every 1 ms. This timer should not be re-configured if Ethernet blocks are being used in the model. If the corresponding interrupt is armed through Hardware Interrupt block, it will run the interrupt handler every 1 ms.

Serial port in MATLAB preferences

Lists the COM port entries available in the device manager and saved COM port in MATLAB preferences of the target hardware. You can select the required COM port from the drop-down.

The COM port value will be saved as MATLAB preference for a given target instead of model. For example, if you choose a same target for a new model, the serial port saved in MATLAB preferences will be selected automatically.

Click refresh to see the latest value serial port value stored in MATLAB preference for the given hardware board and updated list of serial ports available from device manager.

You can also set the serial port in MATLAB preferences for the given hardware board using the MATLAB command:

```
codertarget.tic2000.setSerialPortPreferences(Hardware board, CPU value, Serial port)
```

Here CPU value is optional argument.

To know the COM port used by the target hardware on your computer, see “Serial Configuration for External Mode and PIL” on page 1-67.

Refresh

Lists the new COM port entries available in your device manager.

Click refresh to see the latest value serial port value stored in MATLAB preference for the given hardware board and updated list of serial ports available from device manager.

PIL Baud Rate (UART) Baud rate

This is the PIL baud rate used by the target. This is based on the baud rate that you specify in the **Desired Baud rate (in bits/sec)** parameter for UART0.

See Also

“Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

Hardware Board Settings

Processing Unit

Choose the central processing unit (CPU) or control law accelerator (CLA) onto which to deploy the model block in the model. The top level model is set to `None` for multiprocessor models.

Settings

Default: `None`

See Also

“Multiprocessor Execution” (SoC Blockset) | “Run Multiprocessor Models in External Mode”

ARM Cortex-M4 - Build Options

Use the build options to specify how the build process takes place.

You can set the following parameters for build options:

Build action

Define how Embedded Coder responds when you build your model.

The `Build, load and run` option is supported for Texas Instruments Code Composer Studio CCS v4 and the later versions.

If you select the `Build, load and run` option, you must provide the required CCS hardware configuration file.

Disable parallel build

- on – When you select this option, the blockset compiles generated code and driver source codes in sequential order.
- off – When you clear the selection, the blockset compiles generated code and driver source codes parallely. Parallel execution reduces the time taken to build the model.

Device Name

Select a particular device from the selected processor family.

Boot From Flash (stand alone execution)

The option to specify if the application has to load to the flash. If you do not select this option, the application loads to the RAM.

Use custom linker command file

Select this option, if you have your own custom linker file, which you can specify in the Linker command file parameter. If you do not select this option, based on the device you have selected, a default custom linker command file is used.

Linker command file

For each family of TI processor selected under **Target hardware resources**, one linker command file is selected automatically.

For a different variant of the processor, you can select the variant from the ‘src’ folder in the blockset installation path. You can also create custom linker command file and select the file path using the **Browse** button.

The linker command file path provided can be absolute or relative. If the path provided is relative, the path must be selected with respect to the folder where the model is present or the code generation folder.

CCS hardware configuration file

The Code Composer Studio file required for downloading the application on the hardware. Select one of the .ccxml files from the folder ‘CCS_Config’ folder under blockset installation folder.

Alternately, can use Code Composer Studio to create the `ccxml` file. In Code Composer Studio, go to **File > New > Target Configuration File**. Select the file you created using the **Browse** button. You can also edit the `ccxml` file using the **Edit** button.

The `ccxml` files provided with C2000 Microcontroller Blockset are as follows:The `ccxml` files provided are as follows:

- f2838x.ccxml – TI F2838xD XDS100v2 USB Emulator
- f2838x.ccxml – TI F2838xS XDS100v2 USB Emulator

See Also

More About

- “Model Configuration Parameters for Texas Instruments F2838x (ARM Cortex-M4)” on page 1-40

ARM Cortex-M4 - Clocking

Enter the 'Connectivity Manager (ARM Cortex-M) clock in MHz' value calculated in C28x CPU1

Mention the value of Connectivity Manager (ARM Cortex-M) core clock in MHz configured by C28x CPU1. Value of this parameter must be same as the value of the parameter 'Connectivity Manager (ARM Cortex-M) clock in MHz' (auto calculated in the CPU1 model).

See Also

More About

- “Model Configuration Parameters for Texas Instruments F2838x (ARM Cortex-M4)” on page 1-40

ARM Cortex-M4 - Ethernet

Use the ethernet options to specify the host addresses.

You can set the following parameters for ethernet options:

Enable DHCP for local IP address assignment

Select this parameter to configure the board to get an IP address from the local DHCP server on the network.

You can get to know the dynamical assigned IP address through DHCP from the build log shown in the diagnostic viewer.

Local IP Address

Select this parameter to set the IP address of the board.

Set the board IP address according to these guidelines:

- The subnet address, typically the first 3 bytes of the board IP address, must be the same as the first 3 bytes of the host IP address.
- The last byte of the board IP address must be different from the last byte of the host IP address.
- The board IP address must not conflict with the IP addresses of other computers. For example, if the host IP address is 192.168.8.2, then you can use 192.168.8.3, if available.

Subnet mask

Specify the subnet mask for the board. The subnet mask is a mask that designates a logical subdivision of a network.

The value of the subnet mask must be the same for all devices on the network.

Gateway

Set the serial gateway to the gateway required to access the target computer.

For example, when you set this parameter to `255.255.255.255`, it means that you do not use a gateway to connect to your target computer. If you connect your computers with a crossover cable, leave this property as `255.255.255.255`.

- If you communicate with the target computer from within your LAN, you do not need to change this setting.
- If you communicate from a host located in a LAN different from your target computer (especially via the Internet), you must define a gateway and specify its IP address in this parameter.

MAC Address

Specify the media access control (MAC) address, the physical network address of the board.

Under most circumstances, you do not need to change the MAC address. If you connect more than one board to a single computer so that each address is unique, change the MAC address. You must have a separate network interface card (NIC) for each board.

To change the MAC address, specify an address that is different from the address that belongs to any other device attached to your computer. To obtain the MAC address for a specific board, refer to the label affixed to the board or consult its documentation.

The MAC address must be in the six octet format. For example, DE-AD-BE-EF-FE-ED

See Also

More About

- “Model Configuration Parameters for Texas Instruments F2838x (ARM Cortex-M4)” on page 1-40

ARM Cortex-M4 - UART

Enable UART Loopback

Select this check box to enable data transmission from Tx to Rx buffer. However, selecting this option does not ensure that the data is present in the GPIO MUX.

Desired Baud rate (in bits/sec)

Specify the desired baud rate of the data transmission.

Closest Achievable Baud rate (in bits/sec)

The value in this parameter is calculated based on the desired baud rate that you specify and the system clock frequency. This baud rate is used for the data transfer.

Number of stop bits

Select the number of stop bits used to indicate the end of a byte data transmission. The options available:

- `1` – Select this option to indicate there is 1 stop bit at the end of a byte data transmission.
- `2` – Select this option to indicate there are 2 stop bits at the end of a byte data transmission.

Note User must ensure that the number of stop bits are same for both transmit and receive.

Parity mode

- Select a parity mode that is added at the end of a binary data for error detection.

  The options available are:

  - `None` — Select this option when no parity is used. This is the default option.
  - `Odd` — Select this option to indicate that odd parity is used for data transmission.

    In odd parity mode, the parity bit is set to ‘1’ if the sum of bits with the value ‘1’ is even and the parity bit is set to ‘0’, if the sum of bits with the value ‘1’ is odd.
  - `Even` — Select this option to indicate that even parity is used for data transmission.

    In even parity mode, the parity bit is set to ‘1’, if the sum of bits with the value ‘1’ is odd and the parity bit is set to ‘0’, if the sum of bits with the value ‘1’ is even.
  - `One` — Select this option to indicate that the parity bit is always ‘1’.
  - `Zero` — Select this option to indicate that the parity bit is always ‘0’.

Pin assignment(Tx)

Select a GPIO pin as the UART pin for data transmission. An external USB-to-serial chip should be connected to this pin for serial communication with a PC.

Pin assignment(Rx)

Select a GPIO pin as UART pin for data reception. An external USB-to-serial chip should be connected to this pin for serial communication with a PC.

Enable Receive Interrupt

This check box is enabled by default for updating DMA configuration after data receive. This will trigger UART interrupt when DMA copies any data to FIFO.

Enable Transmit Interrupt

Select this parameter to trigger an ISR from an UART Transmit block. This will trigger UART interrupt when DMA copies any data to FIFO.

See Also

More About

- “Model Configuration Parameters for Texas Instruments F2838x (ARM Cortex-M4)” on page 1-40

C28x-ADC/C28x-ADC_A/C28x-ADC#

The high-speed peripheral clock (HSPCLK) or the system clock (SYSCLKOUT) controls the internal timing of ADC modules. The ADC derives the operating clock speed from the HSPCLK/SYSCLKOUT speed in several prescaler stages. For more information about configuring these scalers, see “Configuring Acquisition Window Width for ADC Blocks”.

You can set the following parameters for the ADC clock prescaler:

Select the CPU core which controls ADC_x module

This parameter is available only for the dual-core processor F2837xD with the **Build options > Select CPU** parameter set to `CPU1`.

The CPU core that controls the ADC module. When you select the `Auto` option for the ADC_x module in a model, the ADC_x module is assigned to the CPU1 core during code generation if the ADC_x block is present in the model, else it is assigned to the CPU2 core. If an ADC_x module is assigned to a CPU core, you cannot use that module in a model that runs in the other CPU core.

ADC clock prescaler (ADCCLK)

Select the ADCCLK divider. This is specific to a processor.

ADC clock frequency in MHz

The clock frequency for ADC, which is auto generated based on the value you select in **ADC clock prescaler (ADCCLK)**.

ADC overlap of sample and conversion (ADC#NONOVERLAP)

Enable or disable overlap of sample and conversion.

ADC clock prescaler (ADCLKPS)

The HSPCLK is divided by ADCLKPS (a 4-bit value) as the first step in deriving the core clock speed of the ADC. The default value is `3`.

ADC Core clock prescaler (CPS)

After dividing the HSPCLK speed by the **ADC clock prescaler (ADCLKPS)** value, divides the result by 2. The default value is `1`.

ADC Module clock (ADCCLK = HSPCLK/ADCLKPS*2)/(CPS+1)) in MHz

The ADC module clock, which indicates the ADC operating clock speed.

Acquisition window prescaler (ACQ_PS)

This value determines the width of the sampling or acquisition period. The higher the value, the wider is the sampling period. This value does not directly alter the core clock speed of the ADC. The default value is `4`.

Acquisition window size ((ACQ_PS+1)/ADCCLK) in micro seconds/channel

Acquisition window size determines the duration for which the sampling switch is closed. The width of SOC pulse is ADCTRL1[11:8] + 1 times the ADCLK period.

Offset

Refer to the individual TRM for specifying the ADC offset values.

Use external reference 2.048V External reference

By default, an internally generated band gap voltage reference supplies the ADC logic. However, depending on application requirements, you can enable the external reference so that the ADC logic uses an external voltage reference instead.

Continuous mode

When the ADC generates an end of conversion (EOC) signal, an ADCINT# interrupt that indicates whether the previous interrupt flag has been acknowledged or not is generated.

ADC offset correction (OFFSET_TRIM: –256 to 255)

The 280x ADC supports offset correction via a 9-bit value that it adds or subtracts before the results are available in the ADC result registers. Timing for results is not affected. The default value is `0`.

VREFHIVREFLO

When you disable the **Use external reference 2.048V** or **External reference** option, the ADC logic uses a fixed 0–3.3 volt input range, and **VREFHI** and **VREFLO** are disabled. To interpret the ADC input as a ratiometric signal, select the **External reference** option. Then, set values for the high voltage reference (**VREFHI**) and the low voltage reference (**VREFLO**). **VREFHI** uses the external ADCINA0 pin, and **VREFLO** uses the internal GND.

INT pulse control

Set the time when the ADC sets ADCINTFLG ADCINTx relative to the SOC and EOC pulses.

SOC high priority

Enables **SOC high priority mode**. In `all in round robin mode`, the default selection, the ADC services each SOC interrupt in a numerical sequence.

Choose one of the `high priority` selections to assign high priority to one or more of the SOCs. In this mode, the ADC operates in round robin mode until it receives a high priority SOC interrupt. The ADC finishes servicing the current SOC, services the high priority SOCs, and then returns to the next SOC in the round robin sequence.

For example, the ADC is servicing SOC8 when it receives a high priority interrupt on SOC1. The ADC completes servicing SOC8, services SOC1, and then services SOC9.

XINT2SOC external pin

The pin to which the ADC sends the XINT2SOC pulse.

ADCEXTSOC external pin

The GPIO pin from which ADC receives the ADCEXTSOC pulse.

Note

- For F2807x, F2837x, F28004x and F2838x processors the ADCEXTSOC external pin is disabled.
- This parameter is available only for specific processors.

ADCEXTSOC Input X-BAR

Indicates the input X-BAR for ADC external SOC.

Note

- For F2807x, F2837x, F28004x and F2838x processors the ADCEXTSOC Input X-BAR is disabled.

- This parameter is available only for specific processors.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-Build Options

Use the build options to specify how the build process takes place.

You can set the following parameters for build options:

Build action

Define how Embedded Coder responds when you build your model.

The `Build, load and run` option is supported for Texas Instruments Code Composer Studio CCS v4 and the later versions.

If you select the `Build, load and run` option, you must provide the required CCS hardware configuration file.

The TI Concerto F28M35x/F28M36x processors support only CCS v5 and the later versions. The TI Delfino F2807x/F2837x processors support only CCS v6 and the later versions.

Device Name

Select a particular device from the selected processor family.

Disable parallel build

- on – When you select this option, the blockset compiles generated code and driver source codes in sequential order.
- off – When you clear the selection, the blockset compiles generated code and driver source codes parallely. Parallel execution reduces the time taken to build the model.

Enable TMU

This option enables support for Trigonometric Math Unit (TMU). Relaxed floating-point mode also gets enabled as TMU hardware instructions are replaced only in relaxed floating point mode.

RTS library calls are replaced with the corresponding TMU hardware instructions for the following floating-point operations: floating point division, sqrt, sin, cos, atan, and atan2.

Note

- There are algorithmic differences between the TMU hardware instructions and the library routines, so the results of operations may differ slightly.
- This parameter is available only for specific C28x devices.

Boot From Flash (stand alone execution)

The option to specify if the application has to load to the flash. If you do not select this option, the application loads to the RAM.

Use custom linker command file

Select this option, if you have your own custom linker file, which you can specify in the Linker command file parameter. If you do not select this option, based on the device you have selected, a default custom linker command file is used.

Linker command file

For each family of TI processor selected under **Target hardware resources**, one linker command file is selected automatically.

For a different variant of the processor, you can select the variant from the 'src' folder in the blockset installation path. You can also create custom linker command file and select the file path using the **Browse** button.

The linker command file path provided can be absolute or relative. If the path provided is relative, the path must be selected with respect to the folder where the model is present or the code generation folder.

CCS hardware configuration file

In the C2000 Microcontroller Blockset installation folder, open CCS_Config and select one of the `ccxml` files.

Alternately, can use Code Composer Studio to create the `ccxml` file. In Code Composer Studio, go to **File > New > Target Configuration File**. Select the file you created using the **Browse** button. You can also edit the `ccxml` file using the **Edit** button.

The `ccxml` files provided with are as follows:

- f28027.ccxml—TI F28027 with Texas Instruments XDS100v1 USB Emulator
- f28035.ccxml—TI F28035 with Texas Instruments XDS100v1 USB Emulator
- f28069.ccxml—TI F28069 with Texas Instruments XDS100v1 USB Emulator
- f2808.ccxml—TI F2808 with Texas Instruments XDS100v1 USB Emulator
- f2808_eZdsp.ccxml—F2808 Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator
- f28044.ccxml—TI F28044 with Texas Instruments XDS100v1 USB Emulator
- f28335.ccxml—TI F28335 with Texas Instruments XDS100v1 USB Emulator
- f28335_eZdsp.ccxml—F28335 Spectrum Digital DSK-EVM-eZdsp onboard USB Emulator
- f2812_BH2000.ccxml—Blackhawk USB2000 Controller for F2812 eZDSP
- f28x_generic.ccxml—Generic Texas Instruments XDS100v1 USB Emulator
- f28x_ezdsp_generic.ccxml—Generic Spectrum Digital eZdsp onboard USB Emulator
- f28x_ezdsp_generic.ccxml—Generic Spectrum Digital eZdsp onboard USB Emulator
- f28377S.ccxml—TI F2837xS with Texas Instruments XDS100v2 USB Emulator
- f28075.ccxml—TI F2807x with Texas Instruments XDS100v2 USB Emulator
- f28377D.ccxml—TI F2837xD with Texas Instruments XDS100v2 USB Emulator
- f28379D.ccxml—TI F2839xD with Texas Instruments XDS100v2 USB Emulator
- f28004x.ccxml—TI F28004x with Texas Instruments XDS100v2 USB Emulator

The `ccxml` files provided with C2000 Microcontroller Blockset are as follows:

- f28M35x.ccxml – Texas Instruments XDS100v2 USB Emulator_0
- f28M36x.ccxml – Texas Instruments XDS100v2 USB Emulator_0

Enable DMA to access ePWM Registers instead of CLA

The option that you can select to enable the DMA to access ePWM registers instead of CLA. This option is available only for F2806x processors.

Enable DMA to peripheral frame 1 (ePWM, HRPWM, eCAP, eQEP, DAC, CMPSS, and SDFM) instead of CLA

The option that you can select to enable the DMA to access peripheral frame 1 (ePWM, HRPWM, eCAP, eQEP, DAC, CMPSS and SDFM) registers instead of CLA. This option is available only for F2837xD, F2837xS, F2807x processors.

Enable DMA to peripheral frame 2 (SPI and McBSP) instead of CLA

The option that you can select to enable the DMA to access peripheral frame 2 (SPI and McBSP) registers instead of CLA. This option is available only for F2837xD, F2837xS, F2807x processors.

Enable FastRTS

This option enables the use of optimized floating point math functions from C28x FPU fastRTS library instead of standard RTS library functions.

By using FastRTS library routines, you can achieve execution speeds considerable faster without rewriting existing code. This option is available only for F2806x, F2833x, F28M35x (C28x) and F28M35x (C28x) processors.

Remap ePWMs for DMA access (Requires silicon revision A and above)

The option that you can select to remap ePWMs registers for DMA access. This option is available only for F2833x processors.

Configure CLA program and data memory

Enable this option to configure LSRAM memory for CLA program or data.

Maximum LSRAM size for CLA program (in KW)

Select the maximum LSRAM size that you can allocate for CLA program in KiloWords.

Note Ensure memory allocation for CLA program and data is within the total available LSRAM memory of the selected processor.

Maximum LSRAM size for CLA data (in KW)

Select the maximum LSRAM size that you can allocate for CLA data in KiloWords (KW).

Note Ensure memory allocation for CLA program and data is within the total available LSRAM memory of the selected processor.

Available LSRAM size for CPU (in KW)

Displays the remaining available LSRAM size for CPU in KiloWords.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-Clocking

Use the clocking options to achieve the CPU clock rate specified on the board. The default clocking values run the CPU clock (CLKIN) at its maximum frequency. The parameters use the external oscillator frequency on the board (OSCCLK) that is recommended by the processor vendor.

For F2837xD and F2838xD dual-core processor, the clock settings are available only when you select the CPU1 option in the **Build options > Select CPU** parameter. When you select CPU2 option in the **Build options > Select CPU** parameter, set the CPU clock with the value available in the **Achievable SYSCLKOUT in MHz** parameter for the CPU1 model.

You can get feedback on the closest achievable SYSCLKOUT value with the specified oscillator clock frequency by selecting the **Auto set PLL based on OSCCLK and CPU clock** check box. Alternatively, you can manually specify the PLL value for the SYSCLKOUT value calculation.

Change the clocking values if:

- You want to change the CPU frequency.
- The external oscillator frequency differs from the value recommended by the manufacturer.

To determine the CPU frequency (CLKIN), use the following equation:

CLKIN = (OSCCLK × PLLCR) / (DIVSEL or CLKINDIV)

Where,

- CLKIN is the frequency at which the CPU operates, also known as the CPU clock.
- OSCCLK is the frequency of the oscillator.
- **PLLCR** is the PLL control register value.
- **CLKINDIV** is the clock in the divider.
- **DIVSEL** is the divider select.

The availability of the DIVSEL or CLKINDIV parameters changes depending on the processor that you select. If neither parameter is available, use the following equation:

CLKIN = (OSCCLK × PLLCR) / 2

You can set the following parameters for clocking:

Desired C28x CPU clock in MHz

Specify the desired CPU clock frequency (CLKIN). This value is taken automatically for **Achievable SYSCLKOUT in MHz = (OSCCLK×PLLCR)/DIVSEL**.

CPU Clock in MHz (C28SYSCLK/SYSCLKOUT)

Enter the value that you specified for **Desired C28x CPU clock in MHz**. This parameter is available only for TI Concerto F28M35x/ F28M36x processors. For more information, see the PLL-Based Clock Module section in the Texas Instruments *Reference Guide* for your processor.

Use internal oscillator

Use the internal zero pin oscillator on the CPU. This parameter is enabled by default.

Oscillator clock (OSCCLK) frequency in MHz

Oscillator frequency used in the processor. This parameter is not available for TI Concerto F28M35x/ F28M36x processors.

Note By default the clock source value is set to 20MHz. Ensure that the newer versions of TI F2838x control cards have clock source value set to 20MHz if they have different value.

Auto set PLL based on OSCCLK and CPU clock

PLL values in PLLCR, DIVSEL, and **Achievable SYSCLKOUT in MHz** are automatically calculated based on the CPU clock entered on the board. This parameter is not available for TI Concerto F28M35x/ F28M36x processors.

PLL control register (PLLCR)

If you select **Auto set PLL based on OSCCLK and CPU clock**, the auto calculated control register value achieves the specified CPU clock value, based on the oscillator clock frequency. Alternatively, you can select a value for **PLL control register (PLLCR)**. This parameter is not available for TI Concerto F28M35x/ F28M36x processors.

PLL output divider (ODIV)

Calculates SYSCLKOUT = ((OSCCLK×SYSPLLMULT)/ODIV)/SYSDIVSEL.

Clock divider (DIVSEL)

If you select **Auto set PLL based on OSCCLK and CPU clock**, the auto calculated control register value achieves the specified CPU clock value, based on the oscillator clock frequency. Alternatively, you can select a value for **Clock divider (DIVSEL)**. This parameter is not available for TI Concerto F28M35x/ F28M36x processors.

Achievable SYSCLKOUT in MHz = (OSCCLK×PLLCR)/DIVSEL

The auto calculated feedback value that matches the **Desired C28x CPU clock in MHz** value, based on the values of OSCCLK, PLLCR, and DIVSEL. This parameter is not available for TI Concerto F28M35x/ F28M36x processors.

Set the 'Achievable SYSCLKOUT in MHz = (OSCCLK*SYSPLLMULT)/SYSDIVSEL' value calculated in CPU1

Available only for CPU2 of dual C28x core processors. Value of this parameter must be same as the value of the parameter **Achievable SYSCLKOUT in MHz = (OSCCLK*PLLCR)/DIVSEL** (auto calculated).

Select the 'Low-Speed Peripheral Clock Prescaler (LSPCLK)' option used in CPU1

Available only for CPU2 of dual C28x core processors. Value of this parameter must be same as the value of the parameter **Low-Speed Peripheral Clock Prescaler (LSPCLK)** specified in CPU1.

Low-Speed Peripheral Clock Prescaler (LSPCLK)

The value using which LSPCLK is scaled. This value is based on SYSCLKOUT.

Low-Speed Peripheral Clock (LSPCLK) in MHz

The value is calculated based on LSPCLK Prescaler. Example: SPI uses a LSPCLK.

High-Speed Peripheral Clock Prescaler (HSPCLK)

The value using which HSPCLK is scaled. This value is based on SYSCLKOUT.

High-Speed Peripheral Clock (HSPCLK) in MHZ

The value is calculated based on HSPCLK Prescaler. Example: ADC uses a HSPCLK.

Analog Subsystem Clock Prescaler (ASYSCLK)

The value using which ASYSCLK is scaled. This value is based on SYSCLKOUT. This option is available only for TI Concerto F28M35x/ F28M36x processors.

Analog Subsystem Clock (ASYSCLK)

The value calculated using the SYSCLKOUT and ASYSCLK Prescaler values. This option is available only for TI Concerto F28M35x/ F28M36x processors.

Connectivity Manager (ARM Cortex-M) clock source

Select the clock source for ARM Cortex-M core. This feeds the CM clock divider. Currently, only `System PLL` is supported as the clock source. This option is available only for TI F2838x (C28x) processors.

Connectivity Manager (ARM Cortex-M) clock divider

Select the divider for the ARM Cortex-M core clock. This divider is acted upon the signal from clock source to get the ARM Cortex-M core clock. This option is available only for TI F2838x (C28x) processors.

Connectivity Manager (ARM Cortex-M) clock in MHz

The calculated value of the clock frequency (in MHz) supplied to ARM Cortex-M core. This option is available only for TI F2838x (C28x) processors.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-DAC

DACx reference voltage

Select the reference voltage for the DAC channel A, B, or C. In this case, x represents the DAC channel A, B, or C.

- ADC reference voltage (VREFHIA/VREFHIB) — The reference voltage used for the ADC. You can use this as reference voltage VREFHIA for DAC A, DAC B and VREFHIB for DAC C.
- External reference voltage through ADCINB0 (VDAC) — A separate external reference voltage for DAC. Ensure that you connect the ADCINB0 pin to the supply voltage.

DACx synchronization signal

Select the synchronization signal to load the value from the writable shadow register into the active register. In this case, x represents the DAC channel A, B, or C.

- SYSCLK — Loads the value from the writable shadow register DACVALS into the active register DACVALA on the next clock cycle.
- PWMSYNC1–12 (EPWMxSYNCPER) — Loads the value from the writable shadow register DACVALS into the active register DACVALA on the next PWM synchronization event.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-COMP

Assign COMP pins to GPIO pins.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-DMA_ch#

The Direct Memory Access (DMA) module transfers data directly between peripherals and memory using a dedicated bus, increasing overall system performance. In this case, # represents the DMA channel number.

You can individually enable and configure each DMA channel.

The DMA module services are event driven. Using the **Interrupt source** parameter, you can configure a wide range of peripheral interrupt event triggers. For more information, see the technical reference manual of your processor.

You can set the following parameters for DMA:

Enable DMA channel

Enable this parameter to edit the configuration of a specific DMA channel. This parameter does not have a corresponding bit or register.

Data size

Select the size of the data bit transfer.

The DMA read/write data buses are 32 bits wide. 32-bit transfers have twice the data throughput of a 16-bit transfer.

When providing DMA service to McBSP, set **Data size** to `16 bit`.

The following parameters are based on a 16-bit word size. If you set **Data size** to `32 bit`, double the value of the following parameters:

- Size: Burst
- Source: Burst step
- Source: Transfer step
- Source: Wrap step
- Destination: Burst step
- Destination: Transfer step
- Destination: Wrap step

Data size corresponds to bit 14 (DATASIZE) in the mode register (MODE).

Interrupt source

Select the peripheral interrupt that triggers a DMA burst for the specified channel.

Different C2000 processors have different interrupt trigger options that can be configured to trigger the DMA. Depending on the processor, the trigger sources include peripheral interrupts from ePWM, ADC, SPI, timer, and external interrupt. Some of these interrupt triggers such as TINT0 may require manual configuration. For external interrupt using GPIO, the configuration is done in the **External Interrupt** tab.

The **Interrupt source** parameter corresponds to bit 4–0 (PERINTSEL) in the mode register (MODE).

Burst size

Specify the number of 16-bit words in a burst, from 1 to 32. The DMA module must complete a burst before it can service the next channel.

Set the burst size for the peripheral DMA module services. For the ADC, the value equals the number of ADC registers used, up to 16. For multichannel buffered serial ports (McBSP), which lack FIFOs, the value is 1.

For RAM, the value can range from 1–32.

This parameter corresponds to bits 4–0 (BURSTSIZE) in the burst size register (BURST_SIZE).

Note This parameter is based on 16-bit word size. If you set **Data size** to `32 bit`, double the value of this parameter.

Transfer size

Specify the number of bursts in a transfer, from 1–65536.

This parameter corresponds to bits 15–0 (TRANSFERSIZE) in the transfer size register (TRANSFER_SIZE).

Source begin address

Set the starting address for the current source address pointer. The DMA module points to this address at the beginning of a transfer and returns to it as specified by the **SRC wrap** parameter.

This parameter corresponds to bits 21–0 (BEGADDR) in the active source begin register (SRC_BEG_ADDR).

Destination begin address

Set the starting address for the current destination address pointer. The DMA module points to this address at the beginning of a transfer and returns to it as specified by the **DST wrap** parameter.

This parameter corresponds to bits 21–0 (BEGADDR) in the active destination begin register (DST_BEG_ADDR).

Source burst step

Set the number of 16-bit words using which the current address pointer is incremented or decremented before the next burst. Enter a value from –4096 (decrement) to 4095 (increment).

To disable incrementing or decrementing the address pointer, set **Burst step** to 0. For example, because McBSP does not use FIFO, configure DMA to maintain the sequence of the McBSP data by moving each word of the data individually.

Accordingly, when you use DMA to transmit or receive McBSP data, set **Burst size** to 1 word and **Burst step** to 0.

This parameter corresponds to bits 15-0 (SRCBURSTSTEP) in the source burst step size register (SRC_BURST_STEP).

Note This parameter is based on 16-bit word size. If you set **Data size** to `32 bit`, double the value of this parameter.

Destination burst step

Set the number of 16-bit words using which the current address pointer is incremented or decremented before the next burst. Enter a value from –4096 (decrement) to 4095 (increment).

To disable incrementing or decrementing the address pointer, set **Burst step** to 0. For example, because McBSP does not use FIFO, configure DMA to maintain the sequence of the McBSP data by moving each word of the data individually. Accordingly, when you use DMA to transmit or receive McBSP data, set **Burst size** to 1 word and **Burst step** to 0.

This parameter corresponds to bits 15–0 (DSTBURSTSTEP) in the destination burst step size register (DST_BURST_STEP).

Note This parameter is based on 16-bit word size. If you set **Data size** to `32 bit`, double the value of this parameter.

Source transfer step

Set the number of 16-bit words using which the current address pointer is incremented or decremented before the next transfer. Enter a value from –4096 (decrement) to 4095 (increment).

To disable incrementing or decrementing the address pointer, set **Transfer step** to 0.

This parameter corresponds to bits 15–0 (SRCTRANSFERSTEP) source transfer step size register (SRC_TRANSFER_STEP).

If DMA is configured to perform memory wrapping (**SRC wrap** enabled), the corresponding source **Transfer step** does not alter the results.

Note This parameter is based on 16-bit word size. If you set **Data size** to `32 bit`, double the value of this parameter.

Destination transfer step

Set the number of 16-bit words using which the current address pointer is incremented or decremented before the next transfer. Enter a value from –4096 (decrement) to 4095 (increment).

To disable incrementing or decrementing the address pointer, set **Transfer step** to 0.

This parameter corresponds to bits 15–0 (DSTTRANSFERSTEP) destination transfer step size register (DST_TRANSFER_STEP).

If DMA is configured to perform memory wrapping (**DST wrap** enabled), the corresponding destination **Transfer step** does not alter the results.

Note This parameter is based on 16-bit word size. If you set **Data size** to `32 bit`, double the value of this parameter.

Source wrap size

Specify the number of bursts before returning the current source address pointer to the **Source Begin Address** value. To disable wrapping, enter a value that is greater than the **Transfer** value.

This parameter corresponds to bits 15-0 (SRC_WRAP_SIZE) in the source wrap size register (SRC_WRAP_SIZE).

Destination wrap size

Specify the number of bursts before returning the current destination address pointer to the **Destination Begin Address** value. To disable wrapping, enter a value that is greater than the **Transfer** value.

This parameter corresponds to bits 15-0 (DST_WRAP_SIZE) in the destination wrap size register (DST_WRAP_SIZE).

Source wrap step

Set the number of 16-bit words using which the SRC_BEG_ADDR address pointer is incremented or decremented when a wrap event occurs. Enter a value from –4096 (decrement) to 4095 (increment).

This parameter corresponds to bits 15–0 (WRAPSTEP) in the source wrap step size registers (SRC_WRAP_STEP).

Note This parameter is based on 16-bit word size. If you set **Data size** to `32 bit`, double the value of this parameter.

Destination wrap step

Set the number of 16-bit words using which the DST_BEG_ADDR address pointer is incremented or decremented when a wrap event occurs. Enter a value from –4096 (decrement) to 4095 (increment).

This parameter corresponds to bits 15–0 (WRAPSTEP) in the destination wrap step size registers (DST_WRAP_STEP).

Note This parameter is based on 16-bit word size. If you set **Data size** to `32 bit`, double the value of this parameter.

Set channel 1 to highest priority

This parameter is available only for DMA_ch1.

Enable this option when DMA channel 1 is configured to handle high-bandwidth data, such as ADC data, and the other DMA channels are configured to handle lower-priority data. When enabled, the DMA module services each enabled channel sequentially until it receives a trigger from channel 1. Upon receiving the trigger, DMA interrupts its service to the current channel at the end of the current word, services the channel 1 burst that generated the trigger, and then continues servicing the current channel at the beginning of the next word.

Disable this channel to give each DMA channel equal priority, or if DMA channel 1 is the only enabled channel. When disabled, the DMA module services each enabled channel sequentially.

This parameter corresponds to bit 0 (CH1PRIORITY) in the priority control register 1 (PRIORITYCTRL1).

Enable first DMA event to trigger the full transfer (one shot mode)

Enable this parameter to have the DMA channel complete an entire *transfer* in response to an interrupt event trigger.

This option allows a single DMA channel and peripheral to dominate resources, and may streamline processing, but it also creates the potential for resource conflicts and delays.

Disable this parameter to have DMA complete one *burst* per channel per interrupt.

This parameter appears only when **Set channel 1 to highest priority** is disabled.

Synchronize ADC interrupt event triggers to DMA wrap counter (sync mode)

Enable this parameter to reset the DMA wrap counter when the **Interrupt source** is set to SEQ1INT and sends the ADCSYNC signal to the DMA wrap counter. This way, the wrap counter and the ADC channels remain synchronized with each other.

If **Interrupt source** is not set to SEQ1INT, **Sync enable** does not alter the results.

This parameter corresponds to bit 12 (SYNCE) of the mode register (MODE).

Do not disable the DMA channel after the transfer is complete (continuous mode)

Select this parameter to leave the DMA channel enabled upon completing a transfer. The channel waits for the next interrupt event trigger.

Clear this parameter to disable the DMA channel upon completing a transfer. The DMA module disables the DMA channel by clearing the RUNSTS bit in the control register when it completes the transfer. To use the channel again, first reset the RUN bit in the control register.

Enable destination sync mode

Enabling this parameter resets the destination wrap counter (DST_WRAP_COUNT) when **Sync enable** is enabled and the DMA module receives the SEQ1INT interrupt/ADCSYNC signal.

Disabling this parameter resets the source wrap counter (SCR_WRAP_COUNT) when the DMA module receives the SEQ1INT interrupt/ADCSYNC signal.

This parameter is associated with bit 13 (SYNCSEL) in the mode register (MODE).

This parameter appears only when **Synchronize ADC interrupt event triggers to DMA wrap counter (sync mode)** is selected.

Generate interrupt

Enable this parameter to have the DMA channel send an interrupt to the CPU via the PIE at the beginning or end of a data transfer.

This parameter corresponds to bit 15 (CHINTE) and bit 9 (CHINTMODE) in the mode register (MODE).

Enable overflow interrupt

Enable this parameter to have the DMA channel send an interrupt to the CPU via PIE if the DMA module receives a peripheral interrupt while a previous interrupt from the same peripheral is waiting to be serviced.

This parameter is used for debugging during the development phase of a project.

The **Enable overflow interrupt** parameter corresponds to bit 7 (OVRINTE) of the mode register (MODE), and involves the overflow flag bit (OVRFLG) and peripheral interrupt trigger flag bit (PERINTFLG).

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-eCAN_A, C28x-eCAN_B

You can set the following parameters for the eCAN module:

CAN module clock frequency (= SYSCLKOUT) in MHz

The clock to the enhanced CAN module. The CAN module clock frequency is equal to SYSCLKOUT for processors such as c280x, c281x, c28044.

CAN module clock frequency (=SYSCLKOUT/2) in MHz

The clock to the enhanced CAN module. The CAN module clock frequency is equal to SYSCLKOUT/2 for processors such as piccolo, c2834x, c28x3x.

Baud rate prescaler (BRP: 2 to 256)/Baud rate prescaler (BRP: 1 to 1024)

The value using which bit rate is scaled.

Time segment 1 (TSEG1):

Set the value of time segment 1. This value, with **TSEG2** and **Baud rate prescaler**, determines the length of a bit on the eCAN bus. Valid values for **TSEG1** are from 1 through 16.

Time segment 2 (TSEG2):

Set the value of time segment 2. This value, with **TSEG1** and **Baud rate prescaler**, determines the length of a bit on the eCAN bus. Valid values for **TSEG2** are from 1 through 8.

Baud rate (CAN Module Clock/BRP/(TSEG1 + TSEG2 +1)) in bits/sec:

CAN module communication speed represented in bits/sec.

SBG

Set the message resynchronization triggering.

SJW

Set the synchronization jump width, which determines how many units of TQ a bit can be shortened or lengthened when resynchronizing. Where, TQ=Baud Rate Prescaler/CAN_CLK.

SAM

Number of samples used by the CAN module to determine the CAN bus level. Selecting `Sample_one_time` samples once at the sampling point. Selecting `Sample_three_times` samples once at the sampling point and twice before at a distance of TQ/2. The CAN module makes a majority decision from the three points.

Enhanced CAN Mode

Enable time-stamping and usage of **Mailbox Numbers** 16 through 31 in the C2000 eCAN blocks. Texas Instruments documentation refers to this as “HECC mode”.

Self test mode

If you set this parameter to `True`, the eCAN module goes to loopback mode. The loopback mode sends a “dummy” acknowledge message back. This mode does not need an acknowledge bit. The default is `False`.

Pin assignment (Tx)

Assign the CAN transmit pin to use with the `eCAN_B` module.

Pin assignment (Rx)

Assign the CAN receive pin to use with the `eCAN_B` module.

For more information about setting the timing parameters for the eCAN modules, see “Configuring Timing Parameters for CAN Blocks”.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-eCAP

Assign eCAP pins to GPIO pins.

ECAP# Input X-BAR

Select the input X-BAR for eCAP. This parameter is available only for specific processors.

ECAP# capture pin assignment

Indicates the GPIO pin used for eCAP in capture mode.

Note

- For F2807x, F2837x, F28004x, F28002x, and F2838x processors the ECAP# capture pin assignment is disabled.
- This parameter is available only for specific processors.

ECAP# APWM pin assignment

Assign eCAP APWM pins to GPIO pins. This parameter is available only for specific processors.

Note When configuring eCAP APWM pin assignment, the Output Xbar GPIO is also set in Output Xbar tab.

eCAPxSYNCIN source selection

Indicates the SYNC source select register for the ePWM SYNCOUT, eCAP SYNCOUT, INPUTXBAR and EtherCATSYNC. You can also set the **SYNCIN** to `Disabled`.

Note

- The default eCAPxSYNCIN source selection value varies based on the processor selected.
- For processors F28004x/F2837xD/F2837xS/F2807x, EXTSYNCIN1 and EXTSYNCIN2 are mapped to Input X-BAR 5 and Input X-BAR 6 respectively.

Output X-BAR

Indicates which Output X-BAR is used for the selected **ECAP# APWM pin assignment** parameter. This parameter is available only for specific processors.

Note

- For F2807x, F2837x, F28004x, F28002x, and F2838x processors the Output X-BAR is disabled.
- This parameter is available only for specific processors.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-EMIF

Use the external memory interface (EMIF) to connect the C2000 processor to an external synchronous or asynchronous memory.

For C2000 processors, the EMIF is supported for these memory devices:

- Synchronous memory interface — JESD21-C SDR SDRAM
- Asynchronous memory interface — SRAM, NOR Flash, or any external device

Based on the processors, the number of EMIF modules supported varies. When you configure the EMIF interface based on the memory used, the GPIO pins required for interacting with the memory through EMIF are also configured. You must ensure that these GPIO pins are not used with other peripherals or as input/output because these pins are not included in the existing conflict check.

The EMIF1 pin configuration for synchronous and asynchronous memory is:

- GPIO38 – GPIO52 (except GPIO42 and GPIO43) are configured as address pins A0 – A12.
- GPIO86 and GPIO87 are configured as address pins A13 and A14 only when asynchronous memory is selected. GPIO86 and GPIO87 are configured as row and column address select (RAS and CAS) when synchronous memory is selected.
- GPIO69 – GPIO85 are configured as data pins D15 – D0. GPIO53 – GPIO68 are configured as data pins D31 – D16 only for 32-bit memory configuration.
- GPIO88 – GPIO91 are configured as data mask pins DQM0 – DQM3. You can manually configure these pins using custom code as address pins A15 – A18 when the EMIF is configured only for 8-bit asynchronous memory.
- GPIO92 and GPIO93 are configured as banks BA1 and BA0.
- GPIO28 – GPIO37 are configured as chip select (CS2, CS3, and CS4), clock enable (SDCKE), clock (CLK), write enable (WE), read and write control (RNW), wait pin (WAIT), and enable pin (OE).

The EMIF2 pin configuration for synchronous and asynchronous memory is:

- GPIO98 – GPIO109 are configured as address pins A0 – A11.
- GPIO53 – GPIO68 are configured as data pins D15 – D0.

  An error message is displayed if the EMIF2 CS# is selected when the EMIF1 is configured for 32-bit data width because the same GPIO pins are used as D31 – D16 for the EMIF1 in 32-bit configuration.
- GPIO96 – GPIO97 are configured as data mask pins DQM0 – DQM1.
- GPIO111 and GPIO112 are configured as banks BA1 and BA0.
- GPIO110 and GPIO113 – GPIO121 are configured as chip select (CS0 and CS2), row and column address select (RAS and CAS), clock enable (SDCKE), clock (CLK), write enable (WE), read and write control (RNW), wait pin (WAIT), and enable pin (OE).

You can set these parameters for the EMIF:

EMIF clock divider (EMIF#CLKDIV)

Select the clock divider for the EMIF# module clock generation. In this case, # represents the number of the EMIF module. The EMIF clock frequency is based on SYSCLKOUT.

Enable CS0 for Synchronous memory

Select the chip select (CS0) to interface with the synchronous dynamic RAM (SDRAM). Synchronous memory supports the following memory sizes and addresses:

- EMIF1_CS0 — Data memory of size 256M × 16 with an address range of 0x80000000 to 0x8FFFFFFF
- EMIF2_CS0 — Data memory of size 3M × 16 with an address range of 0x90000000 to 0x91FFFFFF

Creation and usage of variables in SDRAM require the use of volatile qualifier and far attribute. Use `#pragma` to place the variables in SDRAM memory sections. Custom storage classes EM1_CS0_MEMORY and EM2_CS0_MEMORY are created in the signal object class `tic2000demospkg.Signal` to handle these requirements. You can use these custom storage classes to create variables using the Data Store Memory blocks.

Enable CS# for Asynchronous memory

Select the chip select (CS2/CS3/CS4) to interface with the asynchronous memory (SRAM / NOR Flash). Asynchronous memory supports the following memory sizes and addresses:

- EMIF1_CS2 — Data memory of size 2M × 16 with an address range of 0x00100000 to 0x002FFFFF
- EMIF1_CS3 — Data memory of size 512k × 16 with an address range of 0x00300000 to 0x0037FFFF
- EMIF1_CS4 — Data memory of size 393k × 16 with an address range of 0x00380000 to 0x003DFFFF
- EMIF2_CS2 — Data memory of size 4k × 16 with an address range of 0x00002000 to 0x00002FFF

Use *#pragma* to place the variables in asynchronous memory sections. Custom storage classes EM1_CS2_MEMORY, EM1_CS3_MEMORY, EM1_CS4_MEMORY, and EM2_CS2_MEMORY are created in the signal object class `tic2000demospkg.Signal` to handle these requirements. You can use these custom storage classes to create variables using the Data Store Memory blocks.

SDRAM column address bits

Select the value of the column address bits, thereby selecting the required page size of the connected SDRAM. Column address bits 8, 9, 10, and 11 corresponding to 256, 512, 1024, and 2048-word pages are supported.

The parameter **Page size = (2^column address bits)** is calculated based on the **SDRAM column address bits** parameter value.

Number of internal SDRAM banks

Select the number of memory banks inside the connected SDRAM. SDRAM with 1, 2, and 4 banks are supported.

SDRAM data bus width in bits

Select the data bus width of the connected SDRAM. Data bus widths of 16- and 32-bit are supported.

Refresh to active command delay cycles (T_RFC)

The minimum number of EM#CLK cycles from the refresh or load mode command to the refresh or activate command in the connected SDRAM. In this case, # represents 1 or 2. Some devices refer to this parameter as minimum auto refresh period.

The parameter **t_rfc in ns = (T_RFC+1)/fEM#CLK** is calculated based on the **Refresh to active command delay cycles (T_RFC)** parameter value.

Row precharge to active command delay cycles (T_RP)

The minimum number of EM#CLK cycles required from the row precharge command to the activate or refresh command in the connected SDRAM.

The parameter **t_rp in ns = (T_RP+1)/fEM#CLK** is calculated based on the **Row precharge to active command delay cycles (T_RP)** parameter value.

Active to read or write command delay cycles (T_RCD)

The minimum number of EM#CLK cycles from the activate command to the read or write command in the connected SDRAM.

The parameter **t_rcd in ns = (T_RCD+1)/fEM#CLK** is calculated based on the **Active to read or write command delay cycles (T_RCD)** parameter value.

Last write to row precharge command delay cycles (T_WR)

The minimum number of EM#CLK cycles from the last write transfer or last data in command to the row precharge command in the connected SDRAM.

The parameter **t_wr in ns = (T_WR+1)/fEM#CLK** is calculated based on the **Last write to row precharge command delay cycles (T_WR)** parameter value.

Active to precharge command delay cycles (T_RAS)

The minimum number of EM#CLK cycles from the activate command to the row precharge command in the connected SDRAM.

The parameter **t_ras in ns = (T_RAS+1)/fEM#CLK** is calculated based on the **Active to precharge command delay cycles (T_RAS)** parameter value.

Active to active command delay cycles (T_RC)

The minimum number of EM#CLK cycles from an activate command to the next activate command in the same bank in the connected SDRAM. This is also known as the minimum auto refresh period.

The parameter **t_rc in ns = (T_RC+1)/fEM#CLK** is calculated based on the **Active to active command delay cycles (T_RC)** parameter value.

Active one bank to active another bank command delay cycles (T_RRD)

The minimum number of EM#CLK cycles from an activate command in one bank to an activate command in a different bank in the connected SDRAM.

The parameter **t_rrd in ns = (T_RRD+1)/fEM#CLK** is calculated based on the **Active one bank to active another bank command delay cycles (T_RRD)** parameter value.

Self-refresh exit to other command delay cycles (T_XSR)

The minimum number of EM#CLK cycles from the self refresh exit command to any other command in the connected SDRAM.

The parameter **t_xsr in ns = (T_XSR+1)/fEM#CLK** is calculated based on the **Self-refresh exit to other command delay cycles (T_XSR)** parameter value.

SDRAM refresh period (tRefreshPeriod) in ms

REFRESH_RATE for SDRAM defines the rate at which the connected SDRAM refreshes. SDRAM refresh rate depends on the values of **SDRAM refresh period (tRefreshPeriod) in ms** and

SDRAM refresh cycle (ncycles). Enter the SDRAM refresh period and SDRAM refresh cycles from the SDRAM datasheet. The SDRAM refresh rate is calculated based on the formula *tRefreshPeriod * EMIF clock frequency / ncycles*.

SDRAM CAS Latency

Select the CAS latency required to access the connected SDRAM. SDRAM devices with CAS latencies of 2 and 3 are supported.

Asynchronous mode

Select the asynchronous mode for the connected asynchronous memory. These are the available modes:

- Normal — The byte enable will be active during the entire asynchronous cycle.
- Strobe — The byte enable will be active only during the strobe period of the access cycle mode.

Asynchronous data bus width in bits

Select the data bus width of the connected asynchronous memory. Asynchronous memory data bus width of 8-, 16-, and 32-bit are supported.

Read strobe setup cycles (R_SETUP)

The number of EM#CLK cycles from the EMIF chip select to the pin enable for asynchronous memory assert.

The parameter **t_r_setup in ns = (R_SETUP+1)/fEM#CLK** is calculated based on the **Read strobe setup cycles (R_SETUP)** parameter value.

Read strobe duration cycles (R_STROBE)

The number of EM#CLK cycles during which the pin enable for the asynchronous memory is held active.

The parameter **t_r_strobe in ns = (R_STROBE+1)/fEM#CLK** is calculated based on the **Read strobe duration cycles (R_STROBE)** parameter value.

Read strobe hold cycles (R_HOLD)

The number of EM#CLK cycles during which the EMIF chip select is held active after pin enable for the asynchronous memory is deasserted.

The parameter **t_r_hold in ns = (R_HOLD+1)/fEM#CLK** is calculated based on the **Read strobe hold cycles (R_HOLD)** parameter value.

Write strobe setup cycles (W_SETUP)

The number of EM#CLK cycles from the EMIF chip select to the write enable for the asynchronous memory assert.

The parameter **t_w_setup in ns = (W_SETUP+1)/fEM#CLK** is calculated based on the **Read strobe hold cycles (R_HOLD)** parameter value.

Write strobe duration cycles (W_STROBE)

The number of EM#CLK cycles during which the write enable for the asynchronous memory is held active.

The parameter **t_w_strobe in ns = (W_STROBE+1)/fEM#CLK** is calculated based on the **Write strobe duration cycles (W_STROBE)** parameter value.

Write strobe hold cycles (W_HOLD)

The number of EM#CLK cycles during which the EMIF chip select is held active after write enable for the asynchronous memory is deasserted.

The parameter **t_w_hold in ns = (W_HOLD+1)/fEM#CLK** is calculated based on the **Write strobe hold cycles (W_HOLD)** parameter value.

Turn around cycles (TA)

The number of EM#CLK cycles between the end of one asynchronous memory access and the start of another asynchronous memory access. This delay is not incurred between a read followed by a read or a write followed by a write to the same chip select.

Enable extended wait mode

Select this option to enable the extended wait option for the asynchronous memory. This option can be used if extended asynchronous wait cycles are required based on the EM#WAIT pin.

Maximum extended wait cycles for Asynchronous memory (MAX_EXT_WAIT) [0–255]

This option is enabled if extended wait for any of the asynchronous memory CS# is enabled. Based on the value entered, the EMIF waits for *(MAX_EXT_WAIT+1) * 16* clock cycles before the asynchronous cycle is terminated.

Pin polarity of extended wait

Select the option to make the EMIF wait if the pin is low or high. This option is enabled when the extended wait mode of any of the asynchronous memory CS2/CS3/CS4 is enabled.

Enable wait rise interrupt

Select this option to get an interrupt based on the detection of a rising edge on the EM#WAIT pin. This option is enabled when the extended wait mode of any of the asynchronous memory CS2/CS3/CS4 is enabled.

Enable timeout interrupt

Select this option to get an interrupt when the EM#WAIT pin does not become inactive within the number of cycles defined in **Maximum extended wait cycles for Asynchronous memory (MAX_EXT_WAIT) [0–255]**. This option is enabled when the extended wait mode of any of the asynchronous memory CS2/CS3/CS4 is enabled.

Enable line trap interrupt

Select this option to get an interrupt when there is an invalid cache line size or illegal memory access.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-ePWM

Assign ePWM signals to GPIO pins.

The ePWM X-BAR brings signals to the ePWM modules. Specifically, the ePWM X-BAR is connected to the Digital Compare (DC) submodule of each ePWM module for actions such as tripzones and syncing. You can set the following parameters for ePWM:

EPWM clock divider (EPWMCLKDIV)

Select the ePWM clock divider. This parameter is available only for F2807x, F2837x, F2838x processors.

Select the 'EPWM clock divider (EPWMCLKDIV)' option used for CPU1

Available only for CPU2 of dual C28x core processors. Its value must be the same as the value of the parameter EPWM clock divider (EPWMCLKDIV) selected in CPU1.

TZx Input X-BAR

Indicates the trip-zone input X-BAR.

Note This parameter appears only for specific processors.

TZx pin assignment

Assign the trip-zone input x (TZx) to a GPIO pin.

Note

- For F2807x, F2837x, F28004x and F2838x processors the `TZx pin assignment` is disabled.
- The TZ# pin assignments are available only for TI F280x processors.

TRIP# MUX select

Select the MUX to map the signal to the MUX output.

Selecting **Disable all** will indicate that all MUXes are disabled and the TRIP X-BAR# is not configured.

This parameter appears only for specific processors.

Select MUX input

Select the input to the MUX selected in `TRIP# MUX select`.

The F2807x, F2837x, F2838x, F28002x, F28004x and F28003x processors support ePWM X-BAR. For more information, refer to TI Technical Reference Manual of there respective processors.

Select the input signals for the MUX which is sent to the ePWM module. You can select one signal per MUX. The input signal to the MUX varies based on the MUX selected and processor.

The following table lists the TRIP MUX select and Select MUX input for C28x processor F2838x. The row headers 0-3 represent the **Select MUX input** and column headers 0-31 represent the **TRIP MUX select**.

ePWM X-BAR Mux Configuration Table - F2838x

Select MUX INPUT → / TRIP# MUX select ↓	0	1	2	3
0	CMPSS1.CTRIPH	CMPSS1.CTRIPH_OR_ CTRIPL	ADCAEVT1	ECAP1.OUT
1	CMPSS1.CTRIPL	INPUTXBAR1	CLB1_4.1	ADCCEVT1
2	CMPSS2.CTRIPH	CMPSS2.CTRIPH_OR_ CTRIPL	ADCAEVT2	ECAP2.OUT
3	CMPSS2.CTRIPL	INPUTXBAR2	CLB1_5.1	ADCCEVT2
4	CMPSS3.CTRIPH	CMPSS3.CTRIPH_OR_ CTRIPL	ADCAEVT3	ECAP3.OUT
5	CMPSS3.CTRIPL	INPUTXBAR3	CLB2_4.1	ADCCEVT3
6	CMPSS4.CTRIPH	CMPSS4.CTRIPH_OR_ CTRIPL	ADCAEVT4	ECAP4.OUT
7	CMPSS4.CTRIPL	INPUTXBAR4	CLB2_5.1	ADCCEVT4
8	CMPSS5.CTRIPH	CMPSS5.CTRIPH_OR_ CTRIPL	ADCBEVT1	ECAP5.OUT
9	CMPSS5.CTRIPL	INPUTXBAR5	CLB3_4.1	ADCDEVT1
10	CMPSS6.CTRIPH	CMPSS6.CTRIPH_OR_ CTRIPL	ADCBEVT2	ECAP6.OUT
11	CMPSS6.CTRIPL	INPUTXBAR6	CLB3_5.1	ADCDEVT2
12	CMPSS7.CTRIPH	CMPSS7.CTRIPH_OR_ CTRIPL	ADCBEVT3	ECAP7.OUT
13	CMPSS7.CTRIPL	ADCSOCA	CLB4_4.1	ADCDEVT3
14	CMPSS8.CTRIPH	CMPSS8.CTRIPH_OR_ CTRIPL	ADCBEVT4	EXTSYNCOUT
15	CMPSS8.CTRIPL	ADCSOCB	CLB4_5.1	ADCDEVT4
16	SD1FLT1.COMPH	SD1FLT1.COMPH_OR_ COMPL	Reserved	ERRORSTS.ERROR
17	SD1FLT1.COMPL	INPUTXBAR7	CLB5_4.1	CPU1.CLA1HALT
18	SD1FLT2.COMPH	SD1FLT2.COMPH_OR_ COMPL		ECATSYNC0
19	SD1FLT2.COMPL	INPUTXBAR8	CLB5_5.1	ECATSYNC1
20	SD1FLT3.COMPH	SD1FLT3.COMPH_OR_ COMPL	Reserved	Reserved
21	SD1FLT3.COMPL	INPUTXBAR9	CLB6_4.1	Reserved
22	SD1FLT4.COMPH	SD1FLT4.COMPH_OR_ COMPL	Reserved	Reserved

Select MUX INPUT → / TRIP# MUX select ↓	0	1	2	3
23	SD1FLT4.COMPL	INPUTXBAR10	CLB6_5.1	Reserved
24	SD2FLT1.COMPH	SD2FLT1.COMPH_OR_ COMPL	Reserved	Reserved
25	SD2FLT1.COMPL	INPUTXBAR11	MCANA.FEVT0	CLB7_4.1
26	SD2FLT2.COMPH	SD2FLT2.COMPH_OR_ COMPL	Reserved	Reserved
27	SD2FLT2.COMPL	INPUTXBAR12	MCANA.FEVT1	CLB7_5.1
28	SD2FLT3.COMPH	SD2FLT3.COMPH_OR_ COMPL	Reserved	Reserved
29	SD2FLT3.COMPL	INPUTXBAR13	MCANA.FEVT2	CLB8_4.1
30	SD2FLT4.COMPH	SD2FLT4.COMPH_OR_ COMPL	Reserved	Reserved
31	SD2FLT4.COMPL	INPUTXBAR14	Reserved	CLB8_5.1

ePWM X-BAR Mux Configuration Table - F28004x

Select MUX INPUT → / TRIP# MUX select ↓	0	*1*	2	3
0	CMPSS1.CTRIPH	CMPSS1.CTRIPH_OR_CTRIPL	ADCAEVT1	ECA
1	CMPSS1.CTRIPL	INPUTXBAR1	CLB1_OUT4	ADC
2	CMPSS2.CTRIPH	CMPSS2.CTRIPH_OR_CTRIPL	ADCAEVT2	ECA
3	CMPSS2.CTRIPL	INPUTXBAR2	CLB1_OUT5	ADC
4	CMPSS3.CTRIPH	CMPSS3.CTRIPH_OR_CTRIPL	ADCAEVT3	ECA
5	CMPSS3.CTRIPL	INPUTXBAR3	CLB2_OUT4	ADC
6	CMPSS4.CTRIPH	CMPSS4.CTRIPH_OR_CTRIPL	ADCAEVT4	ECA
7	CMPSS4.CTRIPL	INPUTXBAR4	CLB2_OUT5	ADC
8	CMPSS5.CTRIPH	CMPSS5.CTRIPH_OR_CTRIPL	ADCBEVT1	ECA
9	CMPSS5.CTRIPL	INPUTXBAR5	CLB3_OUT4	Rese
10	CMPSS6.CTRIPH	CMPSS6.CTRIPH_OR_CTRIPL	ADCBEVT2	ECA
11	CMPSS6.CTRIPL	INPUTXBAR6	CLB3_OUT5	Rese
12	CMPSS7.CTRIPH	CMPSS7.CTRIPH_OR_CTRIPL	ADCBEVT3	ECA
13	CMPSS7.CTRIPL	ADCSOCAO	CLB4_OUT4	Rese
14	Reserved	Reserved	ADCBEVT4	EXT
15	Reserved	ADCSOCBO	CLB4_OUT5	Rese
16	SD1FLT1.COMPH	SD1FLT1.COMPH_OR_COMPL	Reserved	Rese
17	SD1FLT1.COMPL	INPUT7	Reserved	CLA
18	SD1FLT2.COMPH	SD1FLT2.COMPH_OR_COMPL	Reserved	Rese
19	SD1FLT2.COMPL	INPUT8	Reserved	Rese
20	SD1FLT3.COMPH	SD1FLT3.COMPH_OR_COMPL	Reserved	Rese
21	SD1FLT3.COMPL	INPUT9	Reserved	Rese
22	SD1FLT4.COMPH	SD1FLT4.COMPH_OR_COMPL	Reserved	Rese
23	SD1FLT4.COMPL	INPUT10	Reserved	Rese
24	Reserved	Reserved	Reserved	Rese
25	Reserved	INPUT11	Reserved	Rese
26	Reserved	Reserved	Reserved	Rese
27	Reserved	INPUT12	Reserved	Rese
28	Reserved	Reserved	Reserved	Rese
29	Reserved	INPUT13	Reserved	Rese
30	Reserved	Reserved	Reserved	Rese

Select MUX INPUT → **TRIP# MUX select ↓**	**0**	***1***	**2**	**3**
31	Reserved	INPUT14	Reserved	Rese

ePWM X-BAR MUX Configuration Table - F2807x and F2837x

Select MUX INPUT → / TRIP# MUX select ↓	0	1	2	3
0	CMPSS1.CTRIPH	CMPSS1.CTRIPH_OR_CTRIPL	ADCAEVT1	EC
1	CMPSS1.CTRIPL	INPUTXBAR1	CLB1_OUT4	AD
2	CMPSS2.CTRIPH	CMPSS2.CTRIPH_OR_CTRIPL	ADCAEVT2	EC
3	CMPSS2.CTRIPL	INPUTXBAR2	CLB1_OUT5	AD
4	CMPSS3.CTRIPH	CMPSS3.CTRIPH_OR_CTRIPL	ADCAEVT3	EC
5	CMPSS3.CTRIPL	INPUTXBAR3	CLB2_OUT4	AD
6	CMPSS4.CTRIPH	CMPSS4.CTRIPH_OR_CTRIPL	ADCAEVT4	EC
7	CMPSS4.CTRIPL	INPUTXBAR4	CLB2_OUT5	AD
8	CMPSS5.CTRIPH	CMPSS5.CTRIPH_OR_CTRIPL	ADCBEVT1	EC
9	CMPSS5.CTRIPL	INPUTXBAR5	CLB3_OUT4	AD
10	CMPSS6.CTRIPH	CMPSS6.CTRIPH_OR_CTRIPL	ADCBEVT2	EC
11	CMPSS6.CTRIPL	INPUTXBAR6	CLB3_OUT5	AD
12	CMPSS7.CTRIPH	CMPSS7.CTRIPH_OR_CTRIPL	ADCBEVT3	
13	CMPSS7.CTRIPL	ADCSOCAO	CLB4_OUT4	AD
14	CMPSS8.CTRIPH	CMPSS8.CTRIPH_OR_CTRIPL	ADCBEVT4	EX
15	CMPSS8.CTRIPL	ADCSOCBO	CLB4_OUT5	AD
16	SD1FLT1.COMPH	SD1FLT1.COMPH_OR_COMPL	Reserved	Res
17	SD1FLT1.COMPL	Reserved	Reserved	Res
18	SD1FLT2.COMPH	SD1FLT2.COMPH_OR_COMPL	Reserved	Res
19	SD1FLT2.COMPL	Reserved	Reserved	Res
20	SD1FLT3.COMPH	SD1FLT3.COMPH_OR_COMPL	Reserved	Res
21	SD1FLT3.COMPL	Reserved	Reserved	Res
22	SD1FLT4.COMPH	SD1FLT4.COMPH_OR_COMPL	Reserved	Res
23	SD1FLT4.COMPL	Reserved	Reserved	Res
24	SD2FLT1.COMPH	SD2FLT1.COMPH_OR_COMPL	Reserved	Res
25	SD2FLT1.COMPL	Reserved	Reserved	Res
26	SD2FLT2.COMPH	SD2FLT2.COMPH_OR_COMPL	Reserved	Res
27	SD2FLT2.COMPL	Reserved	Reserved	Res
28	SD2FLT3.COMPH	SD2FLT3.COMPH_OR_COMPL	Reserved	Res

Select MUX INPUT → / TRIP# MUX select ↓	0	1	2	3
29	SD2FLT3.COMPL	Reserved	Reserved	Res
30	SD2FLT4.COMPH	SD2FLT4.COMPH_OR_COMPL	Reserved	Res
31	SD2FLT4.COMPL	Reserved	Reserved	Res

Note Ensure the selected MUX input peripheral is enabled.

TRIP# MUX (MUX 0->31)

Indicates the input signal selected for each MUX. For example, XXXX1XXXXXXXXXXXXXXXXXXXXXXXXX indicates that input signal 1 was selected for MUX 4. X indicates that the MUX is disabled and no signal from the MUX will be sent to the ePWM X-BAR output.

All the signals which are selected will be logically OR'd and sent to the TRIP# signal of ePWM.

This parameter appears only for specific processors.

RESET TRIP# MUX

Resets the signal selection for the MUX done so far.

Resets the **TRIP# MUX (MUX 0->31)** and **Select MUX input** inputs.

Invert TRIP# output

Enable to invert the TRIP output signal to the ePWM.

SYNCI Input X-BAR

Select the input X-BAR for SYNCI.

This parameter appears only for specific processors.

SYNCI pin assignment

Indicates the GPIO pin for the ePWM external input.

For F2807x, F2837x, F28004x and F2838x processors the SYNCI pin assignment is disabled as this is configured directly through Input X-BAR.

EXTSYNCOUT source selection

Indicates the external SYNCOUT source selection for the ePWM SYNCOUT and eCAP SYNCOUT.

Note

- The default EXTSYNCOUT source selection value varies based on the processor selected.

- For processors F2838x/F28003x/F28002x, EXTSYNCOUT option can be used to send synchronization output from eCAP in a model without ePWM blocks.

ePWMxSYNCIN source selection

Indicates the EPWMxSYNCIN Source Select Register (synchronization input pulse) for the ePWM SYNCOUT and eCAP SYNCOUT. You can also set the **SYNCIN** to `Disabled`.

Note

- The default ePWMxSYNCIN source selection value varies based on the processor selected.
- For processors F28004x/F2837xD/F2837xS/F2807x, EXTSYNCIN1 and EXTSYNCIN2 are mapped to Input X-BAR 5 and Input X-BAR 6 respectively.

SYNCO pin assignment

Assign the ePWM external sync pulse output (SYNCO) to a GPIO pin.

Note SYNCI and SYNCO pin assignments are available for TI F28044, TI F280x, TI Delfino F2833x, TI Delfino F2834x, TI Piccolo F2802x, TI Piccolo F2803x, TI Piccolo F2806 processors.

PWM#x pin assignment

Assign the GPIO pin to the PWM#x module.

GPTRIP#SEL pin assignment(GPIO0~63)

Assign the ePWM trip-zone input to a GPIO pin.

Note The GPTRIP#SEL pin assignment is available only for TI Concerto F28M35x/F28M36x processors.

PWM1SYNCI/ GPTRIP6SEL pin assignment

Assign the ePWM sync pulse input (SYNCI) to a GPIO pin.

Note The PWM1SYNCI/GPTRIP#SEL pin assignments are available only for TI Concerto F28M35x/F28M36x processors.

DCxHTRIPSEL (Enter Hex value between 0 and 0x6FFF) / DCBHTRIPSEL (Enter Hex value between 0 and 0x6FFF)

Assign the **Digital Compare A** high trip input to a GPIO pin.

Note DCxHTRIPSel pin assignment is available only for TI Concerto F28M35x/F28M36x processors.

DCxLTRIPSEL (Enter Hex value between 0 and 0x6FFF) / DCBLTRIPSEL (Enter Hex value between 0 and 0x6FFF)

Assign the **Digital Compare A** low trip input to a GPIO pin.

Note The DCxLTRIPSEL pin assignment is available only for TI Concerto F28M35x/F28M36x processors.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-eQEP

Assign eQEP pins to GPIO pins.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-GPIO

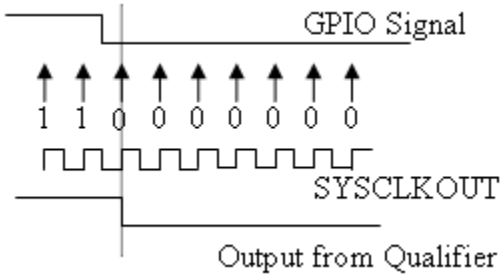
Use the GPIO pins for digital input or output by connecting to one of the three peripheral I/O ports.

The GPIO pins available for various processors are:

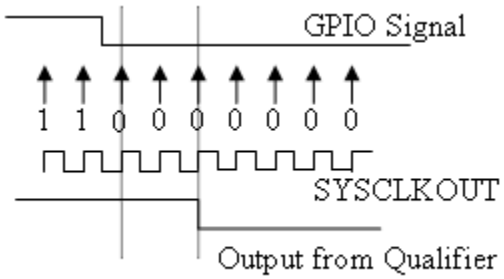
Processors	GPIO Pin Values
C281x	GPIOA, GPIOB, GPIOD, GPIOE, GPIOF, and GPIOG.
F2803x	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO32_39, and GPIO40_44.
F2805x	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO32_39, and GPIO40_42.
F2806x	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO32_39, GPIO40_44, GPIO50_55, and GPIO56_58.
F2823x, F2833x, and C2834x	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO32_39, GPIO40_47, GPIO48_55, and GPIO56_63.
C2801x, F2802x, F28044, F280x	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, and GPIO32_34.
F28M35x (C28x)	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO32_39, GPIO40_47, GPIO48_55, GPIO56_63, GPIO68_71, and GPIO128_135.
F28M36x (C28x)	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO40_47, GPIO48_55, GPIO56_63, GPIO64_71, GPIO72_79, GPIO80_87, GPIO88_95, GPIO96_103, GPIO104_111, GPIO112_119, GPIO120_127, GPIO128_135, and GPIO192_199.
F2807x	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO40_47, GPIO48_55, GPIO56_63, GPIO64_71, GPIO72_79, GPIO80_87, GPIO88_95, GPIO96_103, and GPIO128_135.
F2837xD	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO40_47, GPIO48_55, GPIO56_63, GPIO64_71, GPIO72_79, GPIO80_87, GPIO88_95, GPIO96_103, GPIO104_111, GPIO112_119, GPIO120_127, GPIO128_135, GPIO136_143, GPIO144_151, GPIO152_159, GPIO160_167, and GPIO168_175.
F2837xS	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO40_47, GPIO48_55, GPIO56_63, GPIO64_71, GPIO72_79, GPIO80_87, GPIO88_95, GPIO96_103, GPIO104_111, GPIO112_119, GPIO120_127, GPIO128_135, GPIO136_143, GPIO144_151, GPIO152_159, GPIO160_167, and GPIO168_175.
F2838x	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO40_47, GPIO48_55, GPIO56_63, GPIO64_71, GPIO72_79, GPIO80_87, GPIO88_95, GPIO96_103, GPIO104_111, GPIO112_119, GPIO120_127, GPIO128_135, GPIO136_143, GPIO144_151, GPIO152_159, GPIO160_167, and GPIO168_175.
F28004x	GPIO0_7, GPIO8_15, GPIO16_23, GPIO24_31, GPIO32_39, GPIO40_47, GPIO48_55, and GPIO56_63.

Each pin selected for input offers four signal qualification types:

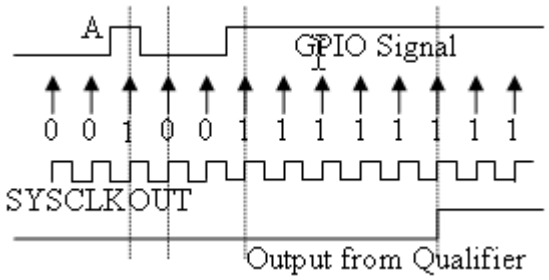
- `Synchronize to SYSCLKOUT only`—This setting is the default for all pins at reset. Using this qualification type, the input signal is synchronized to the system clock, SYSCLKOUT. The following figure shows the input signal measured on each tick of the system clock, and the resulting output from the qualifier.



- `Qualification using 3 samples`—This setting requires three consecutive cycles of the same value for the output value to change. The following figure shows that, in the third cycle, the GPIO value changes to `0`, but the qualifier output is still `1` because it waits for three consecutive cycles of the same GPIO value. The next three cycles have a value of `0`, and the output from the qualifier changes to `0` immediately after the third consecutive value is received.



- `Qualification using 6 samples`—This setting requires six consecutive cycles of the same GPIO input value for the output from the qualifier to change. In the following figure, glitch **A** does not alter the output signal. When the glitch occurs, the counting begins, but as the next measurement is low again, the count is ignored. The output signal does not change until six consecutive samples of the high signal are measured.

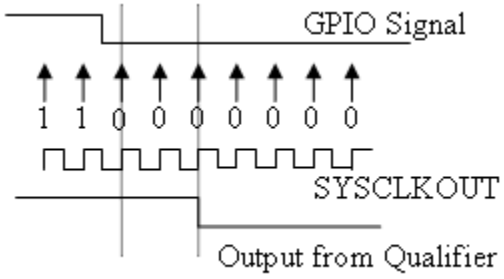


Note These GPIO settings are supported for the F2837xD dual core processor only when you select `CPU1` in **Build options >Select CPU**.

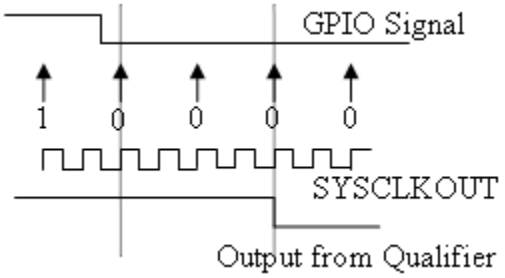
Qualification sampling period

Visible only when the `Qualification using # samples` option is selected. The qualification sampling period, with possible values of `0`–`255`, inclusive, calculates the frequency of the qualification samples or the number of system clock ticks per sample. The formula for calculating the qualification sampling frequency is *SYSCLKOUT/(2 * Qualification sampling period)*, except for zero. When **Qualification sampling period**=`0`, a sample is taken every SYSCLKOUT clock tick. For example, a setting of `0` means that a sample is taken on each SYSCLKOUT tick.

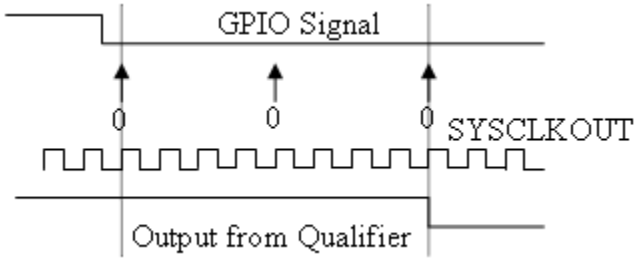
The following figure shows the SYSCLKOUT ticks, a sample taken every clock tick, and the **Qualification type** set to `Qualification using 3 samples`. In this case, the **Qualification sampling period**=`0`:



In the next figure **Qualification sampling period**=`1`. A sample is taken every two clock ticks, and the **Qualification type** is set to `Qualification using 3 samples`. The output signal changes much later than if **Qualification sampling period**=`0`.



In the following figure, **Qualification sampling period**=`2`. A sample is taken every four clock ticks, and the **Qualification type** is set to `Qualification using 3 samples`.



- `Asynchronous`—Using this qualification type, the signal is synchronized to an asynchronous event initiated by the software (CPU) via control register bits.

GPIOA, GPIOB, GPIOD, GPIOE input qualification sampling period

GPIO# Pull Up Disabled

Select this check box to disable the GPIO pull-up register. This option is available only for TI Concerto F28M35x/F28M36x processors.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-I2C

You can set the following parameters for I2C:

Mode

Configure the I2C module as `Master` or `Slave`.

If a module is an I2C master, it:

- Initiates communication with slave nodes by sending the slave address and requesting data transfer to or from the slave.
- Outputs the **Master clock frequency** on the serial clock line (SCL) line.

If a module is an I2C slave, it:

- Synchronizes itself with the serial clock line (SCL) line.
- Responds to communication requests from the master.

In `Slave` mode, you can configure the **Addressing format**, **Address register**, and **Bit count** parameters.

The **Mode** parameter corresponds to bit 10 (MST) of the I2C mode register (I2CMDR).

Addressing format

In `Slave` mode, determines the addressing format of the I2C master and sets the I2C module to the same mode:

- `7-Bit Addressing`—the normal address mode.
- `10-Bit Addressing`—the expanded address mode.
- `Free Data Format`—a mode that does not use addresses. (If you **Enable loopback**, the `Free data format` is not supported.)

The **Addressing format** parameter corresponds to bit 3 (FDF) and bit 8 (XA) of the I2C mode register (I2CMDR).

Own address register

In `Slave` mode, enter the 7-bit (0–127) or 10-bit (0–1023) address that the I2C module uses while it is a slave.

This parameter corresponds to bits 9–0 (OAR) of the I2C own address register (I2COAR).

Bit count

In `Slave` mode, sets the number of bits in each *data byte* the I2C module transmits and receives. This value must match that of the I2C master.

This parameter corresponds to bits 2–0 (BC) of the I2C mode register (I2CMDR).

Module clock prescaler (IPSC: 0 to 255)

In `Master` mode, configures the module clock frequency by entering a value 0–255, inclusive.

Module clock frequency = *I2C input clock frequency* / (*Module clock prescaler* + 1)

The I2C specifications require a module clock frequency between 7 MHz and 12 MHz.

The *I2C input clock frequency* depends on the DSP input clock frequency and the value of the PLL control register divider (PLLCR). For more information on setting the PLLCR, see the documentation for your digital signal controller.

The **Module clock prescaler (IPSC: 0 to 255)** corresponds to bits 7–0 (IPSC) of the I2C prescaler register (I2CPSC).

I2C Module clock frequency (SYSCLKOUT / (IPSC+1)) in Hz

Display the frequency the I2C module uses internally. To set this value, change the **Module clock prescaler**.

For more information about this value, see the “Formula for the Master Clock Period” section in the *TMS320x280x Inter-Integrated Circuit Module Reference Guide*, Literature Number: SPRU721, on the Texas Instruments website.

I2C Master clock frequency (Module Clock Freq/(ICCL+ICCH+10)) in Hz

Display the master clock frequency.

For more information about this value, see the “Clock Generation” section in the *TMS320x280x/ TMS320F28M35x/ TMS320F28M36x Inter-Integrated Circuit Module Reference Guide*, Literature Number: SPRU721/ SPRUH22F/ SPRUHE8B, available on the Texas Instruments website.

Master clock Low-time divider (ICCL: 1 to 65535)

In `Master` mode, the divider determines the duration of the low state of the serial clock line (SCL) on the I2C bus.

The low-time duration of the master clock = Tmod x (ICCL + d).

For more information, see the “Formula for the Master Clock Period” section in the *TMS320x280x/ TMS320F28M35x/ TMS320F28M36x Inter-Integrated Circuit Module Reference Guide*, Literature Number: SPRU721A/ SPRUH22F/ SPRUHE8B, available on the Texas Instruments website.

This parameter corresponds to bits 15–0 (ICCL) of the clock low-time divider register (I2CCLKL).

Master clock High-time divider (ICCH: 1 to 65535)

In `Master` mode, the divider determines the duration of the high state of the serial clock line (SCL) on the I2C bus.

The high-time duration of the master clock = Tmod x (ICCL + d).

For more information about this value, see the “Formula for the Master Clock Period” section in the *TMS320x280x/ TMS320F28M35x/ TMS320F28M36x Inter-Integrated Circuit Module Reference Guide*, Literature Number: SPRU721A, SPRUH22f, SPRUHE8B, available on the Texas Instruments website.

This parameter corresponds to bits 15–0 (ICCH) of the clock high-time divider register (I2CCLKH).

Enable loopback

In `Master` mode, enables or disables digital loopback mode. In digital loopback mode, I2CDXR transmits data over an internal path to I2CDRR, which receives the data after a configurable delay.

The delay, measured in DSP cycles, equals (I2C input clock frequency/module clock frequency) x 8.

While **Enable loopback** is enabled, free data format addressing is not supported.

This parameter corresponds to bit 6 (DLB) of the I2C mode register (I2CMDR).

SDA pin assignment

Select a GPIO pin as I2C data bidirectional port.

This parameter is not available for TI C2000 F280x, F28044, F2833x, and C2834x processors.

SCL pin assignment

Select a GPIO pin as I2C clock bidirectional port.

This parameter is not available for TI C2000 F280x, F28044, F2833x, and C2834x processors.

Enable Tx interrupt

This parameter corresponds to bit 5 (TXFFIENA) of the I2C transmit FIFO register (I2CFFTX).

Tx FIFO interrupt level

This parameter corresponds to bits 4–0 (TXFFIL4-0) of the I2C transmit FIFO register (I2CFFTX).

Enable Rx interrupt

This parameter corresponds to bit 5 (RXFFIENA) of the I2C receive FIFO register (I2CFFRX).

Rx FIFO interrupt level

This parameter corresponds to bit 4–0 (RXFFIL4-0) of the I2C receive FIFO register (I2CFFRX).

Enable system interrupt

Select this parameter to configure the five basic I2C interrupt request parameters in the interrupt enable register (I2CIER):

- Enable AAS interrupt
- Enable SCD interrupt
- Enable ARDY interrupt
- Enable NACK interrupt
- Enable AL interrupt

Enable AAS interrupt

Enable the addressed-as-slave interrupt.

When enabled, the I2C module generates an interrupt (AAS bit = 1) upon receiving one of the following:

- Its **Own address register** value
- A general call (all zeros)
- A data byte in free data format

When enabled, the I2C module clears the interrupt (AAS = 0) upon receiving one of the following:

- Multiple START conditions (7-bit addressing mode only)
- A slave address that is different from **Own address register** (10-bit addressing mode only)
- A NACK or a STOP condition

This parameter corresponds to bit 6 (AAS) of the interrupt enable register (I2CIER).

Enable SCD interrupt

Enable STOP condition detected interrupt.

When enabled, the I2C module generates an interrupt (SCD bit = 1) after the CPU detects a stop condition on the I2C bus.

When enabled, the I2C module clears the interrupt (SCD = 0) upon one of the following events:

- The CPU reads I2CISRC while it indicates a stop condition
- A reset of the I2C module
- Someone manually clears the interrupt

This parameter corresponds to bit 5 (SCD) of the interrupt enable register (I2CIER).

Enable ARDY interrupt

Enable register-access-ready interrupt enable bit.

When enabled, the I2C module generates an interrupt (ARDY bit = 1) after the previous address, data, and command values in the I2C module registers have been used. New values can be written to the I2C module registers.

This parameter corresponds to bit 2 (ARDY) of the interrupt enable register (I2CIER).

Enable NACK interrupt

Enable no acknowledgment interrupt enable bit.

When enabled, the I2C module generates an interrupt (NACK bit = 1) when the module operates as a transmitter in master or slave mode and receives a NACK condition.

This parameter corresponds to bit 1 (NACK) of the interrupt enable register (I2CIER).

Enable AL interrupt

Enable arbitration-lost interrupt.

When enabled, the I2C module generates an interrupt (AL bit = 1) when the I2C module operates as a master transmitter and looses an arbitration contest with another master transmitter.

This parameter corresponds to bit 0 (AL) of the interrupt enable register (I2CIER).

For more information about the I2C parameters, see the *TMS320x280x/ TMS320F28M35x/ TMS320F28M36x Inter-Integrated Circuit Module Reference Guide*, Literature Number: SPRU721A/ SPRUH22F/ SPRUHE8B available on the Texas Instruments website.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-LIN

You can configure the LIN Transmit and LIN Receive blocks within a model.

For detailed information on LIN module, see *TMS320F2803x Piccolo Local Interconnect Network (LIN) Module*, Literature Number SPRUGE2, available at the Texas Instruments website.

LIN Module clock frequency (LM_CLK = SYSCLKOUT/2) in MHz

Display the frequency of the LIN module clock in MHz.

Enable loopback

Enables LIN loopback testing. When this option is enabled, the LIN module does the following:

- Internally redirects the LINTX output to the LINRX input.
- Places the external LINTX pin in a high state.
- Places the external LINRX pin in a high impedance state.

The default is disabled.

Suspension mode

Use this option to configure how the LIN state machine behaves while you debug the program on an emulator. The available options are:

- `Hard_abort`—Halts the transmissions and counters when you enter the LIN debug mode. The transmissions and counters resume when you exit LIN debug mode.
- `Free_run`—Allows completion of the current transmit and receive functions when you enter the LIN debug mode.

Parity mode

Use this option to configure parity checking. The available options are:

- `None`—Disables parity.
- `Odd`—Sets the parity bit to one if you have an odd number of ones in your bytes, such as 00110010.
- `Even`—Sets the parity bit to one if you have an even number of ones in your bytes, such as 00110011.

The default is `None`.

For **ID parity error interrupt** in the LIN Receive block to generate interrupts, enable **Parity mode**.

Frame length bytes

Set the number of data bytes in the response field, from 1–8 bytes.

The default is `8` bytes.

Baud rate prescaler (P: 0-16777215)

To set the LIN baud rate manually, enter a prescaler value, from 0–16777215. Click **Apply** to update the **Baud rate** display.

The default is `15`.

For more information, see the “Baud Rate” topic in the Texas Instruments document, *TMS320F2803x Piccolo Local Interconnect Network (LIN) Module*, Literature Number SPRUGE2.

Baud rate fractional divider (M: 0–15)

To set the LIN baud rate manually, enter a fractional divider value, from 0–15. Click **Apply** to update the **Baud rate** display.

The default is `4`.

For more information, see the “Baud Rate” topic in the Texas Instruments document, *TMS320F2803x Piccolo Local Interconnect Network (LIN) Module*, Literature Number SPRUGE2.

Baud rate (FLINCLK = LM_CLK/(16*(P+1+M/16)) in bits/sec

Display the baud rate. For more information, see “Setting the LIN baud rate”.

Communication mode

Enable or disable the LIN module from using the ID-field bits ID4 and ID5 for length control.

The default is `ID4 and ID5 not used for length control`.

Data byte order

Set the “endianness” of the LIN message data bytes.

The default is `Little_Endian`.

Data swap width

Set the width for data swap. If you set **Data byte order** to `Big_Endian`, the only available option for **Data swap width** is `8_bits`.

Pin assignment (Tx)

Map the LINTX output to a specific GPIO pin.

The default is `GPIO9`.

Pin assignment (Rx)

Map the LINRX input to a specific GPIO pin.

The default is `GPIO11`.

LIN mode

Set the LIN module as a master or a slave. The default is `Slave`.

In master mode, the LIN node can transmit queries and commands to slaves. In slave mode, the LIN module responds to queries or commands from a master.

This option corresponds to the CLK_MASTER field in the SCI Global Control Register (SCIGCR1).

ID filtering

Select the type of mask filtering comparison the LIN module performs.

If you select `ID byte`, the module uses the RECID and ID-BYTE fields in the LINID register to detect a match. If you select this option and enter `0xFF` for LINMASK, the LIN module does not report matches.

If you select `ID slave task`, the module uses the RECID an ID-SlaveTask byte to detect a match. If you select this option and enter `0xFF` for LINMASK, the LIN module reports matches.

The default is `ID slave task byte`.

ID byte

If you set **ID filtering** as `ID byte`, use this option to set the ID BYTE, also known as the “LIN mode message ID”.

In master mode, the CPU writes this value to initiate a header transmission. In slave mode, the LIN module uses this value to perform message filtering.

The default is `0x3A`.

ID slave task byte

If you set **ID filtering** to `ID slave task byte`, use this option to set the ID-SlaveTask BYTE. The LIN node compares this byte with the received ID and determines whether to transmit or receive.

The default is `0x30`.

Checksum type

If you select `Classic`, the LIN node generates the checksum field from the data fields in the response.

If you select `Enhance`, the LIN node generates the checksum field from both the ID field in the header and data fields in the response. LIN 1.3 supports classic checksums only. LIN 2.0 supports both classic and enhanced checksums.

The default is `Classic`.

Enable multibuffer mode

When you select this check box, the LIN node uses transmit and receive buffers instead of just one register. This setting affects various other LIN registers, such as: checksums, framing errors, transmitter empty flags, receiver ready flags, and transmitter ready flags.

The default is selected.

Enable baud rate adapt mode

This option is displayed when you set **LIN mode** to `Slave`.

If you enable this option, the slave node automatically adjusts its baud rate to match that of the master node. For this feature to work, first set the **Baud rate prescaler** and **Baud rate fractional divider**.

If you disable this option, the LIN module sets a static baud rate based on the **Baud rate prescaler** and **Baud rate fractional divider**.

The default is not selected.

Inconsistent synch field error interrupt

This option is displayed when you set **LIN mode** to `Slave`.

If you enable this option, the slave node generates interrupts when it detects irregularities in the synch field. This option is only relevant if you enable **Enable adapt mode**.

The default is `Disabled`.

No response error interrupt

This option is displayed when you set **LIN mode** to `Slave`.

If you enable this option, the LIN module generates an interrupt if it does not receive a complete response from the master node within a timeout period.

The default is `Disabled`.

Timeout after 3 wakeup signals interrupt

This option is displayed when you set **LIN mode** to `Slave`.

When enabled, the slave node generates an interrupt when it sends three wakeup signals to the master node and does not receive a header in response. The slave waits 1.5 seconds before sending another series of wakeup signals.

This interrupt indicates that the master node is having a problem recovering from low-power or sleep mode.

The default is `Disabled`.

Timeout after wakeup signal interrupt

This option is displayed when you set **LIN mode** to `Slave`.

When enabled, the slave node generates an interrupt when it sends a wakeup signal to the master node and does not receive a header in response. The slave waits 150 milliseconds before sending another series of wakeup signals.

This interrupt indicates that the master node is delayed recovering from low-power or sleep mode.

The default is `Disabled`.

Timeout interrupt

This option is displayed when you set **LIN mode** to `Slave`.

When enabled, the slave node generates an interrupt after 4 seconds of inactivity on the LIN bus.

The default is `Disabled`.

Wakeup interrupt

This option is displayed when you set **LIN mode** to `Slave`.

When you enable this option:

- In low-power mode, a LIN slave node generates a wakeup interrupt when it detects the falling edge of a wake-up pulse or a low level on the LINRX pin.
- A LIN slave node that is “awake” generates a wakeup interrupt if it receives a request to enter low-power mode while it receives data.
- A LIN slave node that is “awake” does not generate a wakeup interrupt if it receives a wakeup pulse.

The default is `Disabled`.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-Scheduler Options

Use scheduler options to specify the base rate of your model. An interrupt can be used as a base rate trigger source for the scheduler. The scheduler options available are:

- `Timer 0`—The default option to schedule all synchronous rates present in your model with CPU Timer 0. When you select this option, the CPU Timer 0 is set according to the base rate of the model.
- `ADCINT1`—The option to schedule all synchronous rates present in your model with ADC interrupt 1 (ADCINT1). When you select this option, make sure that ADCINT1 triggers periodically at base rate used in the model.
- `ADCINT2`—The option to schedule all synchronous rates present in your model with ADC interrupt 2 (ADCINT2). When you select this option, make sure that ADCINT2 triggers periodically at base rate used in the model.
- `ADCx1_INT`—The option to schedule all synchronous rates present in your model with ADC interrupt (ADCx1_INT). When you select this option, make sure that ADCx1_INT triggers periodically at base rate used in the model.

  ADCx1_INT, where x can be A,B,C or D and varies depending on the processor selected.

Note The interrupt used to schedule the base rate trigger option in **Hardware implementation > Operating system/scheduler > Base rate trigger** must not be used to configure the Hardware Interrupt block. For example, if you configure **Timer 0** as `Base rate trigger`, you should not use CPU no =1 and PIE number =7 to trigger timer0 interrupt ISR from Hardware interrupt block in the model.

Warning Replacing the default scheduler interrupt source Timer 0 with ADCINT1 or ADCINT2 is an advanced setting. Ensure that you configure ADCINT1 or ADCINT2 to trigger periodically at the specified base rate. If the ADCINT1 or ADCINT2 does not trigger periodically or triggers at a different rate from the base rate, the model execution on the target is unpredictable.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-SCI_A, C28x-SCI_B, C28x-SCI_C, C28x_SCI_D

You can set the following parameters for serial communications interface (SCI):

Enable loopback

Enable the loopback function for self-test and diagnostics. When this function is enabled, a C28x DSP Tx pin is internally connected to its Rx pin, and the DSP can transmit data from its output port to its input port to check the integrity of the transmission.

Suspension mode

The type of suspension to be used while debugging your program with Code Composer Studio. When your program encounters a breakpoint, the suspension mode determines whether to perform the program instruction. The available options are:

- `Hard_abort`—Stops the program immediately.
- `Soft_abort`—Stops when the current receive/transmit sequence is complete.
- `Free_run`—Continues running regardless of the breakpoint.

Number of stop bits

Specify the number of stop bits transmitted.

Parity mode

The type of parity to be used. The available options are:

- `None`—Disables parity.
- `Odd`—Sets the parity bit to one if you have an odd number of ones in your bytes, such as 00110010.
- `Even`—Sets the parity bit to one if you have an even number of ones in your bytes, such as 00110011.

Character length bits

Length in bits of each transmitted or received character. The default value is 8.

Desired baud rate in bits/sec

Specify the desired baud rate.

Baud rate prescaler (BRR = (SCIHBAUD << 8) | SCILBAUD)

The value using which SCI baud rate is scaled. This value is based on LSPCLK.

Closest achievable baud rate (LSPCLK/(BRR+1)/8) in bits/sec

The closest achievable baud rate calculated based on LSPCLK and BRR.

Communication mode

Raw data is unformatted and sent whenever the transmitting side is ready to send, regardless of whether the receiving side is ready or not. Without a wait state, deadlock conditions do not occur and data transmission is asynchronous. With this mode, the receiving side could miss data, but if the data is noncritical, using raw data mode can avoid blocking processes.

When you select protocol mode, handshaking between the host and the processor occurs. The transmitting side sends $SND to indicate it is ready to transmit. The receiving side sends back $RDY to indicate it is ready to receive. The transmitting side then sends data and, when the transmission is completed, it sends a checksum.

Advantages of using protocol mode are:

- Avoids deadlock
- Determines whether data is received without errors (checksum)
- Determines whether data is received by the processor
- Determines time consistency; each side waits for its turn to send or receive

Note

- Deadlocks can occur if an SCI Transmit block tries to communicate with multiple SCI Receive blocks on different COM ports while both transmit and receive blocks are in blocking mode. Deadlocks cannot occur on the same COM port.
- Support for Communication mode as **Protocol** will be removed in a future release of MATLAB.
- Protocol mode is supported only for Data length option as Length via dialog.

Post transmit FIFO interrupt when data is transmitted

If this option is selected, an interrupt is posted when the available data in transmit FIFO is less than or equal to **Transmit FIFO interrupt level**, allowing the subsystem to perform any action. For example, you can use the C28x Hardware Interrupt block for triggering the SCI Transmit block to write data as soon as the FIFO is available for transmitting data.

If the option is not selected, the SCI Transmit block is in polling mode and checks if the FIFO is available. If the FIFO is not full, the block writes data into the FIFO. If the FIFO is full, in blocking mode, the block waits until the FIFO is available. In non-blocking mode, the block continues with the execution of the algorithm without waiting for the availability of the FIFO.

Transmit FIFO interrupt level

The Transmit FIFO generates an interrupt when the number of data bytes in the transmit FIFO is less than or equal to the value selected for this parameter. The transmits FIFO interrupt level ranges between 1 to 16.

To configure this parameter, select the **Post transmit FIFO interrupt when data is transmitted** parameter.

Post receive FIFO interrupt when data is transmitted

If this option is selected, an interrupt is posted when the data available in receive FIFO is greater than or equal to **Receive FIFO interrupt level**, allowing the subsystem to perform any action. For example, you can use the C28x Hardware Interrupt block for triggering the SCI Receive block to read the data as soon as it is received.

Enable receive FIFO interrupt option also enables `RXERRINTENA` (SCI receive error interrupt enable) interrupt. Enabling `RXERRINTENA` interrupt will trigger receive interrupt if RX ERROR bit is set due to errors in receiving the data.

If the option is not selected, the SCI Receive block is in polling mode and checks the FIFO for data. If data is present, the block reads and outputs the data. If data is not present, in blocking mode, the block waits until data is available. In non-blocking mode, the block continues with the execution of the algorithm without waiting for data.

Receive FIFO interrupt level

The receive FIFO generates an interrupt when the number of data bytes in the receive FIFO is greater than or equal to the value selected for this parameter. The receive FIFO interrupt level ranges between 1 to 16.

To configure this parameter, select the **Post receive FIFO interrupt when data is transmitted** parameter.

Data byte order

Select an option to match the endianness of the data being moved.

Pin assignment (Tx)

Assign the SCI transmit pin to be used with the SCI module.

Pin assignment (Rx)

Assign the SCI receive pin to be used with the SCI module.

Note All SCI modules are not available for all TI C2000 processors.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-SPI_A, C28x-SPI_B, C28x-SPI_C, C28x-SPI_D

You can set the following parameters for the serial peripheral interface (SPI):

Mode

Configure the SPI module as `Controller` or `Peripheral`.

Desired baud rate in bits/sec

Specify the desired baud.

Baud rate factor (SPIBRR: between 3 and 127)

The value used to calculate the baud. To set the **Baud rate factor**, search for “Baud Rate Determination” and “SPI Baud Rate Register (SPIBRR) Bit Descriptions” in *TMS320x28xx, 28xxx DSP Serial Peripheral Interface (SPI) Reference Guide*, Literature Number: SPRU059, available on the Texas Instruments website.

Closest achievable baud rate (LSPCLK/(SPIBRR+1)) in bits/sec

The closest achievable baud rate calculated based on LSPCLK and SPIBRR.

Suspension mode

The type of suspension to be used while debugging your program with Code Composer Studio. When your program encounters a breakpoint, the selected suspension mode determines whether to perform the program instruction. The available options are:

- `Hard_abort`—Stops the program immediately.
- `Soft_abort`—Stops when the current receive/transmit sequence is complete.
- `Free_run`—Continues running regardless of the breakpoint.

Enable loopback

Enable the loopback function for self-test and diagnostics. When this function is enabled, the Tx pin on a C28x DSP is internally connected to its Rx pin, and the DSP can transmit data from its output port to its input port to check the integrity of the transmission.

Enable 3-wire mode

Enable SPI communication over three pins instead of the normal four pins.

Enable Tx interrupt

Enable SPI transmit interrupt operation.

Enable high speed mode

Enable high speed SPI mode for supported pins.

This option is available for specific processors.

FIFO interrupt level (Tx)

Set level for transmit FIFO interrupt.

Enable Rx interrupt

Enable SPI receive interrupt operation.

FIFO interrupt level (Rx)

Set level for receive FIFO interrupt.

FIFO transmit delay

FIFO transmit delay (in processor clock cycles) to pause between data transmissions. Enter an integer.

Peripheral in controller out pin assignment

Assign the peripheral in controller out pin SPI value to a GPIO pin.

Peripheral out controller in pin assignment

Assign the peripheral out controller in pin SPI value to a GPIO pin.

CLK pin assignment

Assign the CLK pin to a GPIO pin.

Chip select (provided by SPI module) assignment

Assign the chip select (provided by SPI module) to a GPIO pin.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

C28x-Watchdog

When enabled, if the software fails to reset the watchdog counter within a specified interval, the watchdog resets the processor or generates an interrupt. This feature enables the processor to recover from faults.

For more information, see the *Data Manual* or *System Control and Interrupts Reference Guide* for your processor on the Texas Instruments website.

Enable watchdog

Enable the watchdog timer module.

This parameter corresponds to bit 6 (WDDIS) of the watchdog control register (WDCR) and bit 0 (WDOVERRIDE) of the system control and status register (SCSR).

Counter clock

Set the watchdog timer period relative to OSCCLK/512.

This parameter corresponds to bits 2–0 (WDPS) of the watchdog control register (WDCR).

Note Depending on the processor type, the default value of the watchdog clock (WDCLK) can be based on the internal oscillator (INTOSC1) or external oscillator (OSCCLK).

Timer period ((1/Counter clock)*256) in seconds

Display the timer period in seconds. This value automatically gets updated when you change the **Counter clock** parameter.

Time out event

Configure the watchdog to reset the processor or generate an interrupt when the software fails to reset the watchdog counter. The available options are:

- `Chip reset`—Generates a signal (WDRST) that resets the processor and disables the watchdog interrupt signal (WDINT).
- `Raise WD Interrupt`—Generates a watchdog interrupt signal (WDINT) and disables the reset processor signal (WDRST). The WDINT signal can be used to wake the device from an idle or standby low-power mode.

This parameter corresponds to bit 1 (WDENINT) of the system control and status register (SCSR).

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

CMPSS

The Comparator Subsystem consists of two modules, Comparator High (COMPH) and Comparator Low (COMPH). Each module generates a high digital output when the voltage on the first input pin (positive input) is greater than the voltage on the second input pin (negative input). And each module generates a low digital output when the voltage on the first input pin (positive input) is less than the voltage on the second input pin (negative input).

The second input pin can either be the external pin or the DAC module.

Configure CMPSS#

Configure the comparator subsystem (CMPSS).

Select this parameter to configure the comparator subsystem (CMPSS). If Configure CMPSS# check box is clear, then the CMPSS modules is configured with default values.

Note The number of modules available will vary for different processors.

Configure COMP#

Configure the COMPH or COMPL module.

Select this parameter to configure the COMPH or COMPL module. If configure COMP# check box is clear, then the COMPH# module is configured with default values.

Reload condition for RAMP reference value (RAMPLOADSEL)

Reload condition for RAMP reference value.

Determines whether **CMPSS Ramp Status Register (RAMPSTS)** is updated from **Immediate (RAMPMAXREFA)** or **Shadow (RAMPMAXREFS)** when CMPSS Comparator Status Register (COMPSTS) is triggered.

This parameter is available if you are using RAMP module as DAC source for second input pin of the comparator module.

Note This option is available only for COMPH module.

Invert comparator output

Invert comparator output.

Select this parameter to invert the output of COMP# module.

Enable latch clear by EPWMSYNCPER event

Enable latch clear by EPWMSYNCPER event.

Select this parameter to clear the comparator status latch output by `EPWMSYNCPER` event.

Configure digital filter

Configure the digital filter for COMP#.

Select this parameter to configure digital filter options sample clock, sample window size and threshold size. If the option is left unchecked, the comparator is configured with default values.

Sample clock prescale [0 to 1023]

Set the sample clock prescale for digital filter of COMP#.

It introduces the given number of system clock cycles between input samples.

Sample window size

Set the sample window size for digital filter of COMP#.

Number of samples monitored in digital filter window.

Threshold sample size

Set the threshold sample window size for digital filter of COMP#.

The THRESH samples of the opposite state must appear within the sample window for the output to change state.

Comparator output type for EPWM X-BAR (CTRIP#SEL)

Select the comparator output type source for COMP#.

The comparator output type for EPWM X-BAR (CTRIP#SEL):

- Asynchronous output (ASYNCH)
- Synchronous output (SYNCH)
- Digital output filter (COMP#STS)
- Latched output (COMP#LATCH)
- Asynchronous (ASYNCH) or Latched (COMP#LATCH) output

Comparator output type for OUTPUT X-BAR (CTRIPOUT#SEL)

Select the comparator output type source for COMP#.

The comparator output type for OUTPUT X-BAR (CTRIPOUT#SEL):

- Asynchronous output (ASYNCH)
- Synchronous output (SYNCH)
- Digital output filter (COMP#STS)
- Latched output (COMP#LATCH)
- Asynchronous (ASYNCH) or Latched (COMP#LATCH) output

DAC reference voltage

Select the DAC reference voltage for CMPSS.

Determines the voltage supply selected as the reference for comparator DAC. The reference voltage can either be **Internal reference voltage (VDDA)** or **External reference voltage through ADCINB0 (VDAC)**.

Reload condition for DAC value (SWLOADSEL)

Select the reload condition for DAC value.

Determines the DAC value selected for the software load select. The DAC value can either be **System clock (SYSCLK)** or **External event (EPWMSYNCPER)**.

EPWM peripheral synchronization event

Select the EPWM peripheral synchronization event.

Select the PWMSYNC event to clear the latched output or reload the RAMP values.

EPWM blank window event

Select the EPWM for blanking window

Determines the leading edge blanking of the comparator module.

Note This parameter appears only for specific processors.

Comparator hysteresis value

Set the amount of hysteresis on the comparator inputs.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

Execution profiling

Number of profiling samples to collect

Enter the number of profiling samples to collect. Using execution profiling, you can record the execution time, count the assembler instruction, and the high-level statement in the generated code.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

External Interrupt

External interrupts can be generated using GPIO pins.

XINT# Input X-BAR

Select the input X-BAR for external interrupt. In this case, # represents the number of the external interrupt. This parameter is available only for specific processors.

XINT# GPIO

Indicates the GPIO pins for triggering external interrupts. In this case, # represents the number of the external interrupt.

Note

- For F2807x, F2837x, F28004x and F2838x processors the XINT# GPIO is disabled.
- This parameter is available only for specific processors.

XINT# Polarity

Select the polarity of the signal in GPIO pin. Rising edge, Falling edge or Falling and rising edge are the polarity signals. In this case, # represents the number of the external interrupt.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

External Mode

Allows you to do external mode settings for your model.

Communication interface

Select the type of communication interface to run your model in external mode.

Default: `XCP on Serial`

- XCP on Serial
- XCP on CAN

SCI module

Select the serial communication interface module.

By default `SCI_A` module is selected for Controlcard and Launchpads. For custom boards, select other serial modules to connect to FTDI.

Serial port in MATLAB preferences

Lists the COM port entries available in the device manager and saved COM port in MATLAB preferences of the target hardware. You can select the required COM port from the drop-down.

The COM port value will be saved as MATLAB preference for a given target instead of model. For example, if you choose a same target for a new model, the serial port saved in MATLAB preferences will be selected automatically.

Click refresh to see the latest value serial port value stored in MATLAB preference for the given hardware board and updated list of serial ports available from device manager.

You can also set the serial port in MATLAB preferences for the given hardware board using the MATLAB command:

```
codertarget.tic2000.setSerialPortPreferences(Hardware board, CPU value, Serial port)
```

Here CPU value is optional argument.

To know the COM port used by the target hardware on your computer, see “Serial Configuration for External Mode and PIL” on page 1-67.

Host interface

Select the interface through which the host computer communicates to target hardware for signal monitoring and parameter tuning. This parameter supports only `Third party calibration tools` as host interface.

CAN module

Select the CAN module to be used with external mode.

CAN vendor

Enter the CAN vendor for the CAN module. Use the Vehicle Network Toolbox function `canChannelList()` to get values for **CAN vendor**. In the MATLAB command window, type `canChannelList()` and press enter.

CAN device

Enter the CAN device for the CAN module. Use the Vehicle Network Toolbox function `canChannelList()` to get values for **CAN device**. In the MATLAB command window, type `canChannelList()` and press enter.

CAN channel number

Enter the CAN channel number for the CAN module. Use the Vehicle Network Toolbox function `canChannelList()` to get values for **CAN channel number**. In the MATLAB command window, type `canChannelList()` and press enter.

CAN ID Command

Enter the CAN ID Command for the CAN module.

CAN ID Response

Enter the CAN ID Response for the CAN module.

Rx mailbox number (0-31)

Enter the Rx mailbox number for the CAN module. This parameter supports integers from `0` to `31`.

Tx mailbox number (0-31)

Enter the Tx mailbox number for the CAN module. This parameter supports integers from `0` to `31`.

Extended CAN ID

Select this if you want to use extended ID.

Refresh

Lists the new COM port entries available in your device manager.

Click refresh to see the latest value serial port value stored in MATLAB preference for the given hardware board and updated list of serial ports available from device manager.

Verbose

Select this to view the external mode execution progress and updates in the Diagnostic Viewer or in the MATLAB command window.

Set logging buffer size automatically

Select this to automatically set the number of bytes to preallocate for the buffer in the hardware during simulation. By default, the **Set logging buffer size automatically** parameter is selected. If you clear this parameter, **Logging buffer size (in bytes)** parameter becomes available, where you can manually specify the memory buffer size for XCP-based External mode simulation..

Maximum number of contiguous samples

Specify a value for maximum number of contiguous samples parameter. This parameter dictates the maximum number of samples that can be filled in a single packet and the memory is allocated accordingly.

Use a dedicated timer to improve time stamp accuracy

Select this parameter to log hardware time data inside ISR at ISR trigger rate and idle task at trigger rate for XCP External mode for Serial and CAN.

- `On` - Enabling this options results in data logging inside ISR at ISR trigger rate. You can analyze the log data in Dashboard blocks or Data Inspector but not in Scope or Display blocks.
- `Off` - Disabling this options results in data logging inside ISR at base rate and can analyze data in Scope or Display blocks.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2
- “Serial Configuration for External Mode and PIL” on page 1-67

PIL

Allows you to do profiling settings for your model.

Communication interface

Select the type of communication interface to run your model in external mode.

Default: `Serial`

SCI module

Select the serial communication interface module.

By default `SCI_A` module is selected for Controlcards and Launchpads. For custom boards, select other serial modules to connect to FTDI.

Serial port in MATLAB preferences

Lists the COM port entries available in the device manager and saved COM port in MATLAB preferences of the target hardware. You can select the required COM port from the drop-down.

The COM port value will be saved as MATLAB preference for a given target instead of model. For example, if you choose a same target for a new model, the serial port saved in MATLAB preferences will be selected automatically.

Click refresh to see the latest value serial port value stored in MATLAB preference for the given hardware board and updated list of serial ports available from device manager.

You can also set the serial port in MATLAB preferences for the given hardware board using the MATLAB command:

```
codertarget.tic2000.setSerialPortPreferences(Hardware board, CPU value, Serial port)
```

Here CPU value is optional argument.

To know the COM port used by the target hardware on your computer, see “Serial Configuration for External Mode and PIL” on page 1-67.

Refresh

Lists the new COM port entries available in your device manager.

Click refresh to see the latest value serial port value stored in MATLAB preference for the given hardware board and updated list of serial ports available from device manager.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

SD Card Logging

Use the SD card logging to log signals to SD card mounted on C2000 hardware.

Enable MAT-file logging on SD card

Enables the MAT-file logging for SD card.

SPI module

Select the desired interface on which the SD card is connected to hardware board.

C2000 hardware boards allow SD card to be interfaced through SPI. The SPI module will run in **Controller** mode. For the SPI module, **Clock polarity** will be automatically set to **Falling_edge**, **Clock phase** value will be automatically set to **No_delay** and **Data bits** will be automatically set to **8**. It is advisable not to use the SPI module selected for SD card to perform any other operations through SPI master write, SPI transmit and SPI receive blocks as these may create issues in data logging on SD card.

Note For the selected SPI module, ensure that:

- Select the appropriate GPIO pins for **Peripheral in controller out pin assignment**, **Peripheral out controller in pin assignment**, **CLK pin assignment** and **Chip select (provided by SPI module) pin assignment** in the corresponding **SPI_x** pane.
- GPIO pins are not used with other peripherals or as input/output because these pins are not included in the existing conflict check.

SPI baud rate

Select the desired option for the SPI interface used by the SD card.

The default is `Maximum achievable supported by the inserted SD Card`.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

SDFM

The sigma delta filter module (SDFM) is a four-channel digital filter designed specifically for current measurement and resolver position decoding in motor control applications. Each input channel can receive an independent delta-sigma (ΔΣ) modulator bitstream. The bitstreams are processed by four individually-programmable digital decimation filters.

The filter set includes a fast comparator (secondary filter) for immediate digital threshold comparisons for over-current and under-current monitoring and a primary data filter.

Configure filter #

Configure the filter channel for the SDFM module. Each SDFM module has four filter channels.

Data pin assignment (SD#_D#)

Select the data input (GPIO pin) for each filter channel.

Clock pin assignment (SD#_C#)

Select the clock input (GPIO pin) for each filter channel.

For F2838x processor, the GPIO value that you set in the Clock pin assignment (SD#_C#) option of one filter channel can also be used in other filter channels.

Modulator clock mode

The input control unit is configured to receive the modulated data in following four modes:

- **Same as the modulator data rate (MOD_0)** - The modulator clock runs with the modulator data rate. The modulator data is strobed at every rising edge of the modulator clock.
- **Half the modulator data rate (MOD_1)** - The modulator clock runs with half of the modulator data rate. The modulator data is strobed at every edge of the modulator clock.
- **Manchester encoded (MOD_2)** - The modulator clock is off and the modulator data is Manchester-encoded.
- **Twice the modulator data rate (MOD_3)** - The modulator clock runs with double the modulator data rate. The modulator data is strobed at every other positive modulator clock edge.

Comparator filter type

The comparator filter is configured to one of the four filter types: `SincFast`, `Sinc1`, `Sinc2`, and `Sinc3`.

The comparator filter is a lowpass filter that converts the input bitstream into digital data by digital filtering and decimation.

Comparator over sampling ratio (COSR) [0-31]

The comparator OSR settings can be configured to the value ranging `0` to `31` and are independent of the data filter.

Effective number of bits (ENOB) of the comparator filter depends upon the comparator filter type, COSR, and the sigma-delta modulator frequency.

Comparator higher threshold (HLT#) [0-32767]

Comparator higher threshold (HLT) is used to detect the over-value condition when `comparator data > = higher threshold register value (HLT)` and generate an SDFM interrupt when interrupts are enabled.

HLT lies in the range `0` and `32767`.

Comparator lower threshold (LLT#) [0-32767]

Comparator lower threshold (LLT) is used to detect the under-value condition when `comparator data < = lower threshold register value (LLT)` and generate an SDFM interrupt when interrupts are enabled.

LLT lies in the range `0` and `32767`.

Comparator higher threshold (HLTZ) [0-32767]

Comparator higher threshold (HLTZ) is used to detect threshold crossing event condition when `comparator data > = higher threshold register value (HLTZ)` and does not generate an SDFM interrupt when interrupts are enabled.

The Comparator higher threshold (HLTZ) has a range between `0` and `32767`.

Note This parameter is available only for specific processors.

Data filter type

The data filter is configured to one of the four filter types: `SincFast`, `Sinc1`, `Sinc2`, and `Sinc3`.

The data filter is a lowpass filter that converts the input bitstream into digital data by digital filtering and decimation.

Data over sampling ratio (DOSR)

The data OSR settings can be configured from `0` to `255` and is independent of the comparator filter.

Effective number of bits (ENOB) of the data filter depend upon the data filter type, DOSR, and the sigma-delta modulator frequency.

Data filter FIFO depth

Each data filter has a 16-level deep 32-bit FIFO.

FIFO enables the data filter unit to reduce interrupt overhead.

Note This parameter is available only for specific processors.

Enable data filter reset by ePWM

Enable to reset the data filter by external PWM compare output.

ePWM Module

Select the PWM module(PWM#SOCx) for synchronization.

This parameter is available only for specific processors.

Comparator event # (CEVT#) interrupt

Select the comparator event (CEVT#) interrupt.

This parameter is available only for specific processors.

Enable high level threshold crossing output (HLTZ)

Enable to detect an over-value condition.

HLTZ is used in conjunction with eCAP to measure the frequency or duty cycle of the threshold crossing events.

This parameter is available only for specific processors.

Enable modulator clock failure interrupt

Enable the interrupt for the modulator clock failure.

Enable data filter acknowledge interrupt

Enable the interrupt for new data acknowledgement.

When the primary filter is ready with new filter data, an acknowledgement (AFx) event is generated. For a few processors, data filter acknowledgement is from the FIFO ready data event.

Enable comparator lower threshold (LLT)

Will enable the SDFM interrupt to detect the under - value condition.

Enable comparator higher threshold (HLT)

Will enable the SDFM interrupt to detect the over- value condition.

Synchronize SD Data with PLLCLK

The data input to a filter can be synchronized with the PLL clock.

This parameter is available only for specific processors.

Synchronize SD Clock with PLLCLK

The clock input to a filter can be synchronized with the PLL clock.

This parameter is available only for specific processors.

See Also

More About

- “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2

c2000setup

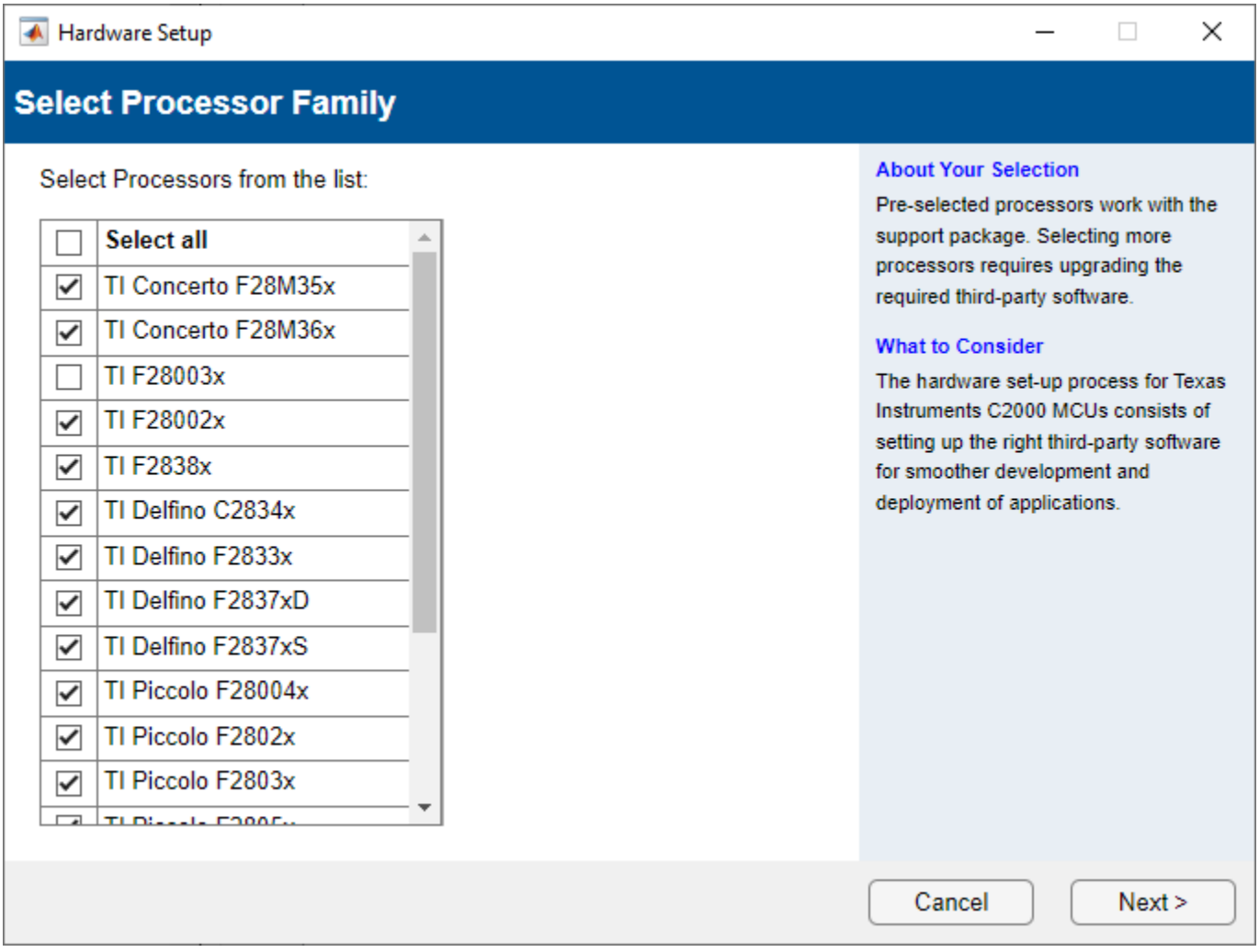
Launch C2000 Microcontroller Blockset hardware setup interface

Syntax

```
c2000setup
```

Description

`c2000setup` launches an interactive hardware setup interface to configure the connection to your C2000 Microcontroller Blockset hardware.



The Hardware Setup window provides instructions for configuring the C2000 Microcontroller Blockset to work with your hardware.

Follow the instructions on each page of the Hardware Setup window. When the hardware setup process completes, you can open the examples to get familiar with the product and its features.

Version History

Introduced in R2023a

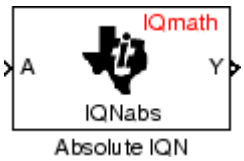
See Also

“Hardware Setup for Third-Party Tools” | “Supported Third-Party Tools for Texas Instruments C2000 Processors”

Blocks

C2000 Absolute IQN

Absolute value



Description

This block computes the absolute value of an IQ number input. The output is also an IQ number.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks® code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

c2000 Arctangent IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

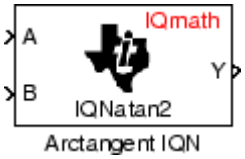
Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

C2000 Arctangent IQN

Four-quadrant arc tangent



Description

The Arctangent IQN block computes the four-quadrant arc tangent of the IQ number inputs and produces IQ number output.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The Texas Instruments function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Function

Type of arc tangent to calculate:

- `atan2` — Compute the four-quadrant arc tangent with output in radians with values from -pi to +pi.
- `atan2PU` — Compute the four-quadrant arc tangent per unit. If `atan2(B,A)` is greater than or equal to 0, `atan2PU(B,A) = atan2(B,A)/2*pi`. Otherwise, `atan2PU(B,A) = atan2(B,A)/2*pi+1`. The output is in per-unit radians with values from 0 to 2*pi radians.

Note The order of the inputs to the Arctangent IQN block correspond to the Texas Instruments convention, with argument 'A' at the top and 'B' at bottom.

References

For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

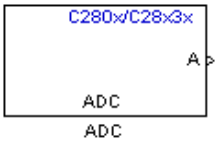
See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to

Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C280x/C2833x ADC

Analog-to-digital converter (ADC)



Libraries:
C2000 Microcontroller Blockset C280x

Description

The ADC block configures the ADC to perform analog-to-digital conversion of signals connected to the selected ADC input pins. The ADC block outputs digital values representing the analog input signal and stores the converted values in the result register of your digital signal processor. You use this block to capture and digitize analog signals from external sources such as signal generators, frequency generators, or audio devices. With the C2833x, you can configure the ADC to use the processor DMA module to move data directly to memory without using the CPU. This frees the CPU to perform other tasks and increases overall system capacity.

Ports

Output

Data — ADC data
vector

The output of the C281x ADC is a vector of `uint16` values. The output values are in the range 0 to 4095 because the C281x ADC is 12-bit converter.

Data Types: `uint16`

Parameters

ADC Control

Module — Select module to use
`A` (default) | `B` | `A and B`

Specify which DSP module to use:

- `A` — Displays the ADC channels in module A (ADCINA0 through ADCINA7).
- `B` — Displays the ADC channels in module B (ADCINB0 through ADCINB7).
- `A and B` — Displays the ADC channels in both modules A and B (ADCINA0 through ADCINA7 and ADCINB0 through ADCINB7)

Conversion mode — Select conversion type
`Sequential` (default) | `Simultaneous`

Type of sampling to use for the signals:

- `Sequential` — Samples the selected channels sequentially
- `Simultaneous` — Samples the corresponding channels of modules A and B at the same time

Start of conversion — Select start of conversion
`Software` (default) | `ePWMxA` | `ePWMxB` | `XINT1_ADCSOC`

Specify the type of signal that triggers the conversion:

Type of signal that triggers conversions to begin:

- `Software` — Signal from software. Conversion values are updated at each sample time.
- `ePWM#A` / `ePWM#B` / `ePWM#A_ePWM#B` — Start of conversion is controlled by user-defined PWM events.
- `XINT2_ADCSOC` — Start of conversion is controlled by the `XINT2_ADCSOC` external signal pin.

The choices available in **Start of conversion** depend on the **Module** setting. The following table summarizes the available choices. For each set of **Start of conversion** choices, the default is given first.

Module Setting	Start of Conversion Choices
`A`	`Software`, `ePWM#A`, `XINT2_ADCSOC`
`B`	`ePWM#B`, `Software`
`A and B`	`Software`, `ePWM#A`, `ePWM#B`, `ePWM#A_ePWM#B`, `XINT2_ADCSOC`

Sample time — Interval at which block reads data
`0.1` (default)

Time in seconds between consecutive sets of samples that are converted for the selected ADC channel(s). This is the rate at which values are read from the result registers. To execute this block asynchronously, set **Sample Time** to `-1`, check the **Post interrupt at the end of conversion** box.

To set different sample times for different groups of ADC channels, you must add separate C281x ADC blocks to your model and set the desired sample times for each block.

Data type — Select the output data type
`uint16` (default) | `double` | `single` | `int8` | `uint8` | `int16` | `int32` | `uint32`

Date type of the output data.

Post interrupt at end of conversion — Enable to post an asynchronous interrupt
`Off` (default) | `on`

Enable this check box to post an asynchronous interrupt at the end of each conversion. The interrupt is posted at the end of conversion.

To execute this block asynchronously, set **Sample Time** to `-1`.

Use DMA (with C28x3x) — Enable to post an asynchronous interrupt
`Off` (default) | `on`

Enable this check box to post an asynchronous interrupt at the end of each conversion. The interrupt is posted at the end of conversion.

To execute this block asynchronously, set **Sample Time** to `-1`.

Input channels

Conversions no. # — Select ADC channel for each conversion number
`ADCINA# and ADCINB#` (default) | `ADCINA#` | **…**

Specific ADC channel to associate with each conversion number.

In oversampling mode, a signal at a given ADC channel can be sampled multiple times during a single conversion sequence. To oversample, specify the same channel for more than one conversion. Converted samples are output as a single vector.

Use multiple output ports — Enable to output multiple ports
`on` (default) | `off`

If more than one ADC channel is used for conversion, you can use separate ports for each output and show the output ports on the block. If you use more than one channel and do not use multiple output ports, the data is output in a single vector.

More About

Modes

The ADC block supports ADC operation in dual and cascaded modes. In dual mode, either module `A` or module `B` can be used for the ADC block, and two ADC blocks are allowed in the model. In cascaded mode, both module `A` and module `B` are used for a single ADC block.

Version History

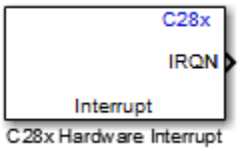
Introduced in R2016b

See Also

c280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x/F28002x/F28003x ePWM | “Configuring Acquisition Window Width for ADC Blocks”

C28x Hardware Interrupt

Interrupt Service Routine to handle hardware interrupt on C28x processors



Description

Execution scheduling models based on timer interrupts do not meet the requirements of some real-time applications to respond to external events. The C28x Hardware Interrupt block addresses this problem by allowing asynchronous processing of interrupts triggered by events managed by other blocks in the C28x DSP Chip Support Library.

When the C28x Hardware Interrupt block has an external interrupt selection, the selection enables interrupts on the selected general-purpose I/O pins. To configure these pins, see the **Configuration Parameters > Hardware Implementation > Hardware board settings > Target hardware resources > External Interrupt** pane. For more information, see “Model Configuration Parameters for Texas Instruments C2000 Processors” on page 1-2.

Vectorized Output

The output of this block is a function call. The size of the function call line equals the number of interrupts the block is set to handle. Each interrupt is represented by four parameters shown on the dialog box of the block. These parameters are a set of four vectors of equal length. Each interrupt is represented by one element from each parameter (four elements total), one from the same position in each of these vectors.

Each interrupt is described by:

- CPU interrupt numbers
- Peripheral Interrupts Expansion (PIE) interrupt numbers
- Task priorities
- Preemption flags

Each interrupt is described by a CPU interrupt number, a PIE interrupt number, a task priority, and a preemption flag.

The CPU and PIE interrupt numbers together uniquely specify a single interrupt for a single peripheral or peripheral module.

The following table lists the PIE and CPU interrupt numbers for the c28x processors F280x, F2802x, F2803x, F2805x, F2806x, F2833x, F28M35x, and F28M36x that support 12×8 interrupts. The row headers 1–12 represent the CPU values, and the column headers 1–8 represent the PIE values.

PIE and CPU Interrupt Numbers for F280x, F2802x, F2803x, F2805x, F2806x, F2833x, F28M35x, and F28M36x Processors

PIE ⇒ / CPU ⇓	1	2	3	4	5	6	7	8
1	SEQ1INT (ADC) / ADCINT1	SEQ2INT (ADC) / ADCINT2	Reserved	XINT1	XINT2	ADCINT / ADCINT9	TINT0 (TIMER 0)	WAKEINT (LPM/WD)
2	EPWM1_TZINT	EPWM2_TZINT	EPWM3_TZINT	EPWM4_TZINT	EPWM5_TZINT	EPWM6_TZINT	EPWM7_TZINT	EPWM8_TZINT
3	EPWM1_INT	EPWM2_INT	EPWM3_INT	EPWM4_INT	EPWM5_INT	EPWM6_INT	EPWM7_INT	EPWM8_INT
4	ECAP1_INT	ECAP2_INT	ECAP3_INT	ECAP4_INT	ECAP5_INT	ECAP6_INT	EPWM10_TZINT / HRCAP1_INT	EPWM9_TZINT / HRCAP2_INT
5	EQEP1_INT	EQEP2_INT	EQEP3_INT	HRCAP3_INT	HRCAP4_INT	Reserved	EPWM10_INT	EPWM9_INT
6	SPIRXINTA (SPI-A)	SPITXINTA (SPI-A)	SPIRXINTB (SPIB_RX) / MRINTB (McBSP-B)	SPITXINTB (SPIB_TX) / MXINTB (McBSP-B)	SPIRXINTC (SPI-C) / MRINTA (McBSP-A_RX)	SPITXINTC (SPI-C) / MXINTA (McBSP-A_TX)	SPIRXINTD (SPI-D) / EPWM12_TZINT	SPITXINTD (SPI-D) / EPWM11_TZINT
7	DINTCH1 (DMA1)	DINTCH2 (DMA2)	DINTCH3 (DMA3)	DINTCH4 (DMA4)	DINTCH5 (DMA5)	DINTCH6 (DMA6)	EPWM12_INT	EPWM11_INT
8	I2CINT1A	I2CINT2A	Reserved	Reserved	SCIRXINTC (SCI-C)	SCITXINTC (SCI-C)	Reserved	Reserved
9	SCIRXINTA (SCIA_RX)	SCITXINTA (SCIA_TX)	SCIRXINTB (SCIB_RX) / LINA_INT0	SCITXINTB (SCIB_TX) / LINA_INT1	ECAN0INTA (CANA_1)	ECAN1INTA (CANA_2)	ECAN0INTB (CANB_1)	ECAN1INTB (CANB_2)
10	EPWM9_TZINT / ADCINT1	EPWM10_TZINT / ADCINT2	EPWM11_TZINT / ADCINT3	EPWM12_TZINT / ADCINT4	EPWM13_TZINT / ADCINT5	EPWM14_TZINT / ADCINT6	EPWM15_TZINT / ADCINT7	EPWM16_TZINT / ADCINT8
11	CLA1_INT1 / EPWM9_INT7 / MTOCIPCINT1	CLA1_INT2 / EPWM10_INT / MTOCIPCINT2	CLA1_INT3 / EPWM11_INT / MTOCIPCINT3	CLA1_INT4 / EPWM12_INT / MTOCIPCINT4 /	CLA1_INT5 / EPWM13_INT	CLA1_INT6 / EPWM14_INT	CLA1_INT7 / EPWM15_INT	CLA1_INT8 / EPWM16_INT
12	XINT3	XINT4 / C28FLSINGERR	XINT5	XINT6 / C28RAMSINGERR	XINT7 / C28RAMACCVIOL	Reserved	LVF	LUF

The PIE and CPU interrupt numbers for the c28x processors F2807x, F2837xS, F2837xD, F2838x, F28004x, F28002x, and F28003x that support 12×16 interrupts are:

PIE and CPU Interrupt Numbers for F2807x, F2837xS, F2837xD, F2838x, F28004x, F28002x, and F28003x Processors

PIE ⇒ CPU ⇓	1	2	3	4	5	6	7	8
1	ADCA1	ADCB1	ADCC1	XINT1	XINT2	ADCD1	TIMER 0	WAKE / WDOG
2	EPWM1_TZ	EPWM2_TZ	EPWM3_TZ	EPWM4_TZ	EPWM5_TZ	EPWM6_ TZ	EPWM7_TZ	EPWM8_TZ
3	EPWM1	EPWM2	EPWM3	EPWM4	EPWM5	EPWM6	EPWM7	EPWM8
4	ECAP1	ECAP2	ECAP3	ECAP4	ECAP5	ECAP6	ECAP7	Reserved
5	EQEP1	EQEP2	EQEP3	Reserved	CLB1	CLB2	CLB3	CLB4
6	SPIA_RX	SPIA_TX	SPIB_RX	SPIB_TX	MCBSPA_RX	MCBSPA_TX	MCBSPB_RX	MCBSPB_TX
7	DMA_CH1	DMA_CH2	DMA_CH3	DMA_CH4	DMA_CH5	DMA_CH6	Reserved	Reserved
8	I2CA	I2CA_FIFO	I2CB	I2CB_FIFO	SCIC_RX	SCIC_TX	SCID_RX	SCID_TX
9	SCIA_RX	SCIA_TX	SCIB_RX	SCIB_TX	CANA_0	CANA_1	CANB_0	CANB_1
10	ADCA_EVT	ADCA2	ADCA3	ADCA4	ADCB_EVT	ADCB2	ADCB3	ADCB4
11	CLA1_1	CLA1_2	CLA1_3	CLA1_4	CLA1_5	CLA1_6	CLA1_7	CLA1_8
12	XINT3	XINT4	XINT5	MPOST	FMC.DONE	VCU	FPU_OVERFLOW	FPU_UNDERFLOW

PIE ⇒ CPU ⇓	9	10	11	12	13	14	15	16
1	I2CA	SYS_ERR	ECATSYNC0 (CPU1 only)	ECATINTn (CPU1 only)	IPC0/CIPC0	IPC1/CIPC1	IPC2/CIPC2	IPC3/CIPC3
2	EPWM9_TZ	EPWM10_ TZ	EPWM11_TZ	EPWM12_TZ	EPWM13_TZ	EPWM14_TZ	EPWM15_TZ	EPWM16_TZ
3	EPWM9	EPWM10	EPWM11	EPWM12	EPWM13	EPWM14	EPWM15	EPWM16
4	FSITXA_INT1	FSITXA_INT2	FSITXB_INT1	FSITXB_INT2	FSIRXA_INT1	FSIRXA_INT2	FSIRXB_INT1	FSIRXB_INT2
5	SD1 / SDFM1	SD2/SDFM1	ECATRSTINTn (CPU1 only)	ECATSYNC1 (CPU1 only)	SDFM1DR1	SDFM1DR2	SDFM1DR3	SDFM1DR4
6	SPIC_RX	SPIC_TX	SPID_RX	SPID_TX	SDFM2DR1	SDFM2DR2	SDFM2DR3	SDFM2DR4
7	FSIRXC_INT1	FSIRXC_INT2	FSIRXD_INT1	FSIRXD_INT2	FSIRXE_INT1	FSIRXE_INT2	FSIRXF_INT1	FSIRXF_INT2

PIE ⇒ CPU ⇓	9	10	11	12	13	14	15	16
8	LINA_0/ FSIRXG_IN T1	LINA_1/ FSIRXG_IN T2	FSIRXH_IN T1	FSIRXH_IN T2	PMBUSA/ CLB5	CLB6	UPPA (CPU1 only)/CLB7	CLB8
9	MCANSS_I NT0(CPU1 only)	MCANSS_I NT1 (CPU1 only)	MCANSS_E CC_CORR_P UL_INT (CPU1 only)	MCANSS_W AKE_AND_T S_PLS_INT (CPU1 only)	PMBUSA	CM_STATU S (CPU1 only)	USBA (CPU1 only)	Reserved
10	ADCC_EVT	ADCC2	ADCC3	ADCC4	ADCD_EVT	ADCD2	ADCD3	ADCD4
11	CMTOCPUx IPCINTR0	CMTOCPUx IPCINTR1	CMTOCPUx IPCINTR2	CMTOCPUx IPCINTR3	CMTOCPUx IPCINTR4	CMTOCPUx IPCINTR5	CMTOCPUx IPCINTR6	CMTOCPUx IPCINTR7
12	EMIF_ ERROR	RAM_CORR ECTABLE_E RROR/ ECAP6INT2	FLASH_CO RRECTABL E_ERROR/ ECAP7INT2	RAM_ACCE SS_VIOLATI ON	SYS_PLL_ SLIP/ CPUxCRC_I NT	AUX_PLL_S LIP// CLA1CRC_I NT	CLA OVER FLOW	CLA UNDERFLO W

The PIE and CPU interrupt numbers for the c281x processors are:

PIE and CPU Interrupt Numbers for C281x Processors

PIE ⇒ / CPU ⇓	1	2	3	4	5	6	7	8
1	PDPINTA (EV-A)	PDPINTB (EV-B)	Reserved	XINT1	XINT2	ADCINT (ADC)	TINT0 (TIMER 0)	WAKEINT (LPM/WD)
2	CMP1INT (EV-A)	CMP2INT (EV-A)	CMP3INT (EV-A)	T1PINT (EV-A)	T1CINT (EV-A)	T1UFINT (EV-A)	T1OFINT (EV-A)	Reserved
3	T2PINT (EV-A)	T2CINT (EV-A)	T2UFINT (EV-A)	T2OFINT (EV-A)	CAPINT1 (EV-A)	CAPINT2 (EV-A)	CAPINT3 (EV-A)	Reserved
4	CMP4INT (EV-B)	CMP5INT (EV-B)	CMP6INT (EV-B)	T3PINT (EV-B)	T3CINT (EV-B)	T3UFINT (EV-B)	T3OFINT (EV-B)	Reserved
5	T4PINT (EV-B)	T4CINT (EV-B)	T4UFINT (EV-B)	T4OFINT (EV-B)	CAPINT4 (EV-B)	CAPINT5 (EV-B)	CAPINT6 (EV-B)	Reserved
6	SPIRXINTA (SPI)	SPITXINTA (SPI)	Reserved	Reserved	MRINT (McBSP)	MXINT (McBSP)	Reserved	Reserved
7	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
8	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
9	SCIRXINTA (SCI-A)	SCITXINTA (SCI-A)	SCIRXINTB (SCI-B)	SCITXINTB (SCI-B)	ECAN0INT (CAN)	ECAN1INT (CAN)	Reserved	Reserved
10	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
11	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
12	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved

The task priority indicates the relative importance of the tasks associated with the asynchronous interrupts. The lowest value in this field represents the highest priority. The default priority value of the base rate task is 40, so the priority value for each asynchronously triggered task must be less than 40 (to configure them as higher-priority) for these tasks to preempt the base rate task.

The preemption flag determines whether a given interrupt is preemptable or not. Preemption overrides prioritization, if an interrupt triggers a higher-priority task while a lower-priority task is running, the execution of the lower-priority task can be suspended and resumed after the completion of the higher priority task, provided the lower-priority task is configured as preemptable.

Parameters

CPU interrupt numbers

Enter a vector of CPU interrupt numbers for the interrupts you want to process asynchronously.

PIE interrupt numbers

Enter a vector of PIE interrupt numbers for the interrupts you want to process asynchronously.

Simulink task priorities

Enter a vector of task priorities for the interrupts you want to process asynchronously.

See the discussion of this block's “Vectorized Output” on page 2-8 for an explanation of task priorities.

Preemption flags

Enter a vector of preemption flags for the interrupts you want to process asynchronously.

See the discussion of this block's “Vectorized Output” on page 2-8 for an explanation of preemption flags.

Enable simulation input

Select this check box if you want to be able to test asynchronous interrupt processing in the context of your Simulink software model.

Note Select this check box to enable you to test asynchronous interrupt processing behavior in Simulink software.

See Also

“ADC-PWM Synchronization Using ADC Interrupt”

“Asynchronous Scheduling”

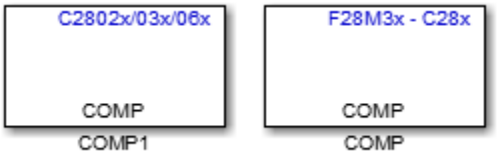
“Field-Oriented Control of PMSM with Quadrature Encoder Using C2000 Processors”

“Schedule a Multi-Rate Controller for a Permanent Magnet Synchronous Machine”

“Photovoltaic Inverter with MPPT Using Solar Explorer Kit”

C2802x/C2803x/C2806x/F28M3x COMP

Compare two input voltages on comparator pins



Description

Configure the comparator module to output the comparison result on the comparator pins of the processor.

Parameters

Comparator module

Select the comparator module to configure. Use only one block per module.

Second input

Select `COMPxB` to compare the voltage of **Input Pin A** with **Input Pin B**

Select `Internal DAC` to compare the voltage of **Input Pin A** with the output of a DAC reference located in the comparator. For more information, see the “DAC Reference” section of the *TMS320x2802x, 2803x Piccolo Analog-to-Digital Converter (ADC) and Comparator*.

The comparator source outputs 1, if **Input Pin A** has a value greater than **Input Pin B** or the 10-bit DAC reference. Otherwise, it outputs 0.

Invert comparator output

Select this check box to apply a logical NOT to the output of the comparator source. For example, when the comparator source outputs 1, the circuit inverts it to 0.

Synchronization

Select `Asynchronous` to pass the asynchronous version of the comparator output. Select `Synchronous` to pass the synchronous version of the comparator output. Selecting `Synchronous` enables the **Qualification period** option.

Qualification period

Qualify changes in the comparator output before passing them along. The `Passed through` setting passes changes in the comparator value along without qualifying them. The `consecutive clocks` settings pass changes in the comparator value along after receiving the specified number of consecutive samples with the same value. Use this setting to prevent intermittent and spurious changes in the comparator output.

Sample time

Specify the time interval between samples. To inherit sample time from the upstream block, set this parameter to -1.

References

TMS320x2802x, 2803x Piccolo Analog-to-Digital Converter (ADC) and Comparator, Literature Number: SPRUGE5, from the Texas Instruments Web site.

C2802x/C2803x/C2805x/C2806x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x/F28002x/F28003x ADC

Configure ADC to sample analog pins and output digital data

Libraries:
C2000 Microcontroller Blockset / C2802x
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C281x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x
C2000 Microcontroller Blockset / F28M35x / C28x
C2000 Microcontroller Blockset / F28M36x / C28x

Description

Configures the Type 3 to Type 5 ADC to output a constant stream of data collected from the ADC pins on the DSP. For more information on ADC types, refer to C2000 Real-Time Control Peripheral Reference Guide.

An ADC block allows for reading one ADC channel. Use multiple ADC blocks to read multiple ADC channels.

Ports

Output

Data — ADC data
vector

The output of the ADC is a vector of `uint16` values.

Data Types: `uint16`

Parameters

SOC Trigger

ADC Module — Select module to use
`A` (default) | `B` | `A and B`

Select ADC Module 1 or ADC Module 2 for conversion.

Select ADC Module A through D for the processors that support Type 4 ADC.

Note The **ADC Module** parameter is available only for Texas Instruments C2000 processors that support Type 3, Type 4, or Type 5 ADC.

ADC Resolution — Select 12 bit or 16 bit resolution
`A` (default) | `B` | `A and B`

Select 12-bit (Single-ended input) or 16-bit (Differential inputs) ADC resolution options.

In 12-bit mode, only single-ended input is supported. In 16-bit mode, the input voltage to the converter is sampled through a pair of input pins, that means the differential inputs between the two channels is converted.

Note

- This parameter is supported only for Texas Instruments C2000 F2837xD, Texas Instruments C2000 F2838x and Texas Instruments C2000 F2837xS processors.
- The 16-bit (Differential inputs) ADC mode is not enabled by default in most of the processors.

Sampling mode — Select sampling type
`Single` (default) | `Simultaneous`

Type of sampling to use for the signals:

- `Single` — Samples the selected channels sequentially,
- `Simultaneous` — Samples the corresponding channels of modules 1 and 2 at the same time. The hardware allows each signal of a pair to be sampled at the same time.

Note This parameter is supported only for Texas Instruments C2000 F28M3x processors.

SOC trigger number — Start of conversion number
`SOC0` (default) | `SOC1` | `SOC2` | **…**

Identify the start-of-conversion trigger by number. In single sampling mode, you can select an individual trigger. In simultaneous sampling mode, you can select triggers in pairs.

SOCx acquisition window — Length of the acquisition period
`7` (default) | `8` | `. . .`

Define the length of the acquisition period in ADC clock cycles. The value of this parameter depends on the SYSCLK and the minimum ADC sample time. The value of SOC acquisition window is subtracted by 1 and set to ACQPS field in ADC register. For more information, see the ADC Acquisition (Sample and Hold) Window section of the *TMS320x2802x, 2803x Piccolo Analog-to-Digital Converter (ADC) and Comparator Reference Guide*.

SOCx trigger source — Source that triggers the start of conversion
`Software` (default) | `EVA` | `External pin`

Select the source that triggers the start of conversion. The following types of inputs are available:

- `Software`.
- `CPU Timers 0/1/2 interrupts`.
- `XINT2 SOC`.
- `ePWMx SOCA and SOCB`.

If you set **SOCx trigger source** to `XINT2_XINT2SOC`, use the **Input5 pin assignment** parameter at **Hardware Implementation > Target hardware resources** to define the external GPIO pin that triggers the start of conversion.

Note The SOCx trigger source input **ePWMx SOCA and SOCB** range will vary according the processor selected.

ADCINT will trigger SOCx — Use ADC interrupt to trigger SOC
`No ADCINT` (default) | `ADCINT1` | `ADCINT2`

At the end of conversion, use the ADCINT1 or ADCINT2 interrupt to trigger a start of conversion (SOC). This loop creates a continuous sequence of conversions. The default selection, No ADCINT disables this parameter. To set the interrupt, select the Post interrupt at EOC trigger option, and choose the appropriate interrupt.

Sample time — Interval at which block reads data
`0.1` (default)

Time in seconds between consecutive sets of samples that are converted for the selected ADC channel(s). This is the rate at which values are read from the result registers. To execute this block asynchronously, set **Sample Time** to `-1`, check the **Post interrupt at the end of conversion** box.

Data type — Select the output data type
`uint16` (default) | `double` | `single` | `int8` | `uint8` | `int16` | `int32` | `uint32`

Date type of the output data.

Post interrupt at EOC trigger — Post interrupts when ADC triggers
`Off` (default) | `on`

Post interrupts when the ADC triggers EOC pulses. When you select this option, the dialog box displays the **Interrupt selection** and **ADCINT# continuous mode** options.

Note For new processors, the Interrupt selection provides option **ADCA#, ADCB#** and so on.

Interrupt selection — ADC interrupt selection
`ADCINT1` (default) | `ADCINT2`

Select which interrupt the ADC posts after triggering an EOC pulse.

Dependencies

To enable this parameter, select **Post interrupt at EOC trigger** parameter.

ADC# continuous mode — Generate ADC interrupt
`ADCINT1` (default) | `ADCINT2`

When the ADC generates an end of conversion (EOC) signal, generate an ADCINT# interrupt, whether the previous interrupt flag has been acknowledged or not.

Dependencies

To enable this parameter, select **Post interrupt at EOC trigger** parameter.

Input channels

Conversions channel — Select ADC channel to which this ADC conversion applies
`ADCINA0` (default) | `ADCINA1` | `ADCINA2` | **…**

Select the input channel to which this ADC conversion applies. For Type 4 ADC, if you select 16-bit (differential inputs) mode, the differential voltage between the two channels is converted.

References

TMS320x2802x, 2803x Piccolo Analog-to-Digital Converter (ADC) and Comparator, Literature Number: SPRUGE5, from the Texas Instruments Web site.

Version History

Introduced in R2016b

See Also

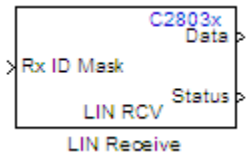
C28x Hardware Interrupt

Topics

“ADC-PWM Synchronization Using ADC Interrupt”
“Configuring Acquisition Window Width for ADC Blocks”

C2803x LIN Receive

Receive data via local interconnect network (LIN) module on target



Description

The Local Interconnect Network (LIN) bus implements a serial communications protocol for distributed automotive and industrial applications. In particular, LIN serves low cost applications that do not require the bandwidth or robustness provided by the CAN protocol.

The LIN Receive block configures the target to receive scalar or vector data from the LINRX or LINTX pins.

Each C2803x target has one LIN module. Your model can only contain one LIN Transmit and one LIN Receive block per module.

The C2803x LIN Transmit block takes three inputs:

- **ID**: Set the value of the LIN ID for the LIN transmit node.
- **TX ID Mask**: Set the value of the LIN ID mask for the LIN transmit node.
- **Data**: Connect this input to the data source.

For more information and examples, see:

- “Configuring LIN Communications”
- “LIN-Based Control of PWM Duty Cycle”

Note Many LIN-specific settings are located under **Peripherals > LIN** in Hardware Implementation -> Target Hardware Resources for your model. Verify that these settings meet the requirements of your application.

Parameters

Data type

Select the data type the LIN block outputs to the model. Available options are `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, or `uint32`. To interpret the data, the data type and data length must match those of the data input to transmitting LIN node.

The default value is `int16`.

Data length

Set the length of the data the LIN block outputs to the model. This value is measured in multiples of the **Data type**. For example, if **Data type** is `int16` and **Data length** is `int16`, the LIN block outputs the data to the model in lengths of

1 x int16

If you set the **Data length** to a value greater than 1, the block outputs the data as vectors.

To interpret the data, the data type and data length must match those of the data input to transmitting LIN node.

The default value is 1.

Note In a loopback configuration, the maximum data length cannot exceed 8 bytes. If the sum of the incoming and the outgoing data exceeds the hardware buffer length of the LIN module, the module discards incoming bytes of data.

Initial output

Set the initial value the DATA port outputs to the model before the LIN node has received data.

The default value is `0`.

Action taken when connection times out

Specify what the LIN block outputs on the DATA port in response to a connection time-out. The choices are:

- `Output the last received value` — the DATA port outputs the last data value the LIN node received.
- `Output custom value` — the DATA port outputs the value defined by **Output value when connection times out**.

The default value is `Output the last received value`.

If the LIN node has not received data, and you set this parameter to `Output the last received value`, the DATA port outputs the **Initial output** value.

Output value when connection times out

Specify the custom value the DATA port outputs when **Action taken when connection times out** is set to `Output custom value` and a connection timeout occurs.

Enable blocking mode

If you enable (select) this checkbox, the target application stops and waits for the LIN node to receive data before continuing. If you disable this option, the application continues running and does not wait for data to arrive.

The default value is disabled (deselected).

Verify checksum

If you enable (select) this option, the LIN node verifies the checksum it receives.

The default value is disabled (deselected).

Output receiving status

Enabling (selecting) this checkbox adds a `status` output to the LIN Receive block, as shown in the following figure.

The `status` output reports the following values for each message the LIN node receives:

- `0`: No error.
- `-1`: A time-out occurred while the block was waiting to receive data.
- `-2`: Unable to receive.
- Other status values represent the highest 8 bits of the SCI Flags Register. Convert these values from decimal to binary. Then determine the meaning of these values by referring to Table 14. SCI Flags Register (SCIFLR) Field Descriptions in *TMS320F2803x Piccolo Local Interconnect Network (LIN) Module*, Literature Number SPRUGE2, available at the Texas Instruments Web site.

Receive buffer interrupt

If you enable this option, the SCI node generates an interrupt after it receives a complete frame. The default value is `Disabled`.

Checksum error interrupt

If you enable this option, the LIN block generates an interrupt when the incoming message contains an invalid checksum.

The default value is `Disabled`.

The TXRX Error Detector Checksum Calculator verifies checksums for incoming messages. With the classic LIN implementation, the checksum only covers the data fields. For LIN 2.0–compliant messages, the checksum includes both the ID field and the data fields. If you enable this option, the Checksum Calculator generates interrupts when it detects checksum errors, such as those caused by LIN message collisions.

Framing error interrupt

If you enable this option, the LIN module generates interrupts when framing errors occur.

The default value is `Disabled`.

Overrun error interrupt

If you enable this option, the LIN module generates interrupt when overrun errors occur.

The default value is `Disabled`.

ID parity error interrupt

If you enable this option, the LIN module generates an ID-Parity interrupt when it receives an invalid ID.

The default value is `Disabled`.

If you enable this option, also enable **Parity mode** in Hardware Implementation -> Target Hardware Resources.

ID match interrupt

If you enable this option, the LIN module generates an interrupt when the LIN node validates the ID in messages it receives.

The default value is `Disabled`.

Sample time

Set the block's input sample time, T_s.

The default value is `0.1` seconds.

References

For detailed information on the LIN module, see *TMS320F2803x Piccolo Local Interconnect Network (LIN) Module*, Literature Number SPRUGE2, available at the Texas Instruments Web site.

See Also

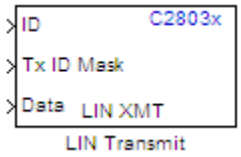
C2803x LIN Transmit (block reference)

“Configuring LIN Communications”

“LIN-Based Control of PWM Duty Cycle”

C2803x LIN Transmit

Transmit data from target via serial communications interface (SCI) to host



Description

The Local Interconnect Network (LIN) bus implements a serial communications protocol for distributed automotive and industrial applications. In particular, LIN serves low cost applications that do not require the bandwidth or robustness provided by the CAN protocol.

The C2803x LIN Transmit block takes three inputs:

- **ID**: Set the value of the LIN ID for the LIN transmit node.
- **TX ID Mask**: Set the value of the LIN ID mask for the LIN transmit node.
- **Data**: Connect this input to the data source.

Note Many LIN-specific settings are located under **Peripherals > LIN** in **Hardware Implementation > Target hardware resources** for your model. Verify that these settings meet the requirements of your application.

Parameters

Send checksum

Select this checkbox to include a checksum in the last data field of the checkbyte. LIN 2.0 implementations require this checksum.

The default value is unchecked (disabled).

Physical bus error interrupt

The LIN master node detects when the physical bus cannot convey a valid message. For example, if the bus had a short circuit to ground or to V_{BAT}. This raises a physical bus error flag in all of the LIN nodes on the network. If you enable **Physical bus error interrupt**, the LIN transmit node generates an interrupt in response to a physical bus error flag.

Bit error interrupt

If you enable this option, the LIN node compares the data it transmits and the data on the LIN bus.

The default value is `Disabled`.

The TXRX Error Detector Bit Monitor compares data bits on the LIN transmit (LINTX) and receive (LINRX) pins. If the data do not match, the Bit Monitor raises a bit-error flag. When you enable this option, the bit-error flag also produces a bit-error interrupt.

Transmit buffer interrupt

If you enable this option, the LIN node generates an interrupt while it is generating a checksum and setting the Transmitter buffer register ready flag.

The default value is `Disabled`.

References

For detailed information on the SCI module, see *TMS320F2803x Piccolo Local Interconnect Network (LIN) Module*, Literature Number SPRUGE2, available at the Texas Instruments Web site.

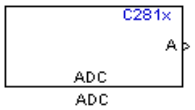
See Also

“Configuring LIN Communications”

“LIN-Based Control of PWM Duty Cycle”

C281x ADC

Analog-to-digital converter (ADC)



Libraries:
C2000 Microcontroller Blockset C281x

Description

The C281x ADC block configures the C281x ADC to perform analog-to-digital conversion of signals connected to the selected ADC input pins. The ADC block outputs digital values representing the analog input signal and stores the converted values in the result register of your digital signal processor. You use this block to capture and digitize analog signals from external sources such as signal generators, frequency generators, or audio devices.

Ports

Output

Data — ADC data
vector

The output of the C281x ADC is a vector of `uint16` values. The output values are in the range 0 to 4095 because the C281x ADC is 12-bit converter.

Data Types:

Parameters

ADC Control

Module — Select module to use
`A` (default) | `B` | `A and B`

Specify which DSP module to use:

- `A` — Displays the ADC channels in module A (ADCINA0 through ADCINA7).
- `B` — Displays the ADC channels in module B (ADCINB0 through ADCINB7).
- `A and B` — Displays the ADC channels in both modules A and B (ADCINA0 through ADCINA7 and ADCINB0 through ADCINB7)

Then, use the check boxes to select the desired ADC channels.

Conversion mode — Select conversion type
`Sequential` (default) | `Simultaneous`

Type of sampling to use for the signals:

- `Sequential` — Samples the selected channels sequentially
- `Simultaneous` — Samples the corresponding channels of modules A and B at the same time

Start of conversion — Select start of conversion
`Software` (default) | `EVA` | `External pin`

Specify the type of signal that triggers the conversion:

- `Software` — Signal from software
- `EVA` — Signal from Event Manager A (only for Module A)
- `EVB` — Signal from Event Manager B (only for Module B)
- `External` — Signal from external hardware

Sample time — Interval at which block reads data
`0.1` (default)

Time in seconds between consecutive sets of samples that are converted for the selected ADC channel(s). This is the rate at which values are read from the result registers. To execute this block asynchronously, set **Sample Time** to `-1`, check the **Post interrupt at the end of conversion** box.

To set different sample times for different groups of ADC channels, you must add separate C281x ADC blocks to your model and set the desired sample times for each block.

Data type — Select the output data type
`uint16` (default) | `double` | `single` | `int8` | `uint8` | `int16` | `int32` | `uint32`

Date type of the output data.

Post interrupt at end of conversion — Enable to post an asynchronous interrupt
`Off` (default) | `on`

Enable this check box to post an asynchronous interrupt at the end of each conversion. The interrupt is posted at the end of conversion.

Input channels

Number of conversions — Select number of ADC channels for conversion
`1` (default) | `2` | `3` | …

Number of ADC channels to use for analog-to-digital conversions.

Conversions no. # — Select ADC channel for each conversion number
`1` (default) | `2` | `3` | …

Specific ADC channel to associate with each conversion number.

In oversampling mode, a signal at a given ADC channel can be sampled multiple times during a single conversion sequence. To oversample, specify the same channel for more than one conversion. Converted samples are output as a single vector.

Use multiple output ports — Enable to output multiple ports
`off` (default) | `on`

If more than one ADC channel is used for conversion, you can use separate ports for each output and show the output ports on the block. If you use more than one channel and do not use multiple output ports, the data is output in a single vector.

More About

Triggering

The C281x ADC trigger mode depends on the internal setting of the source start-of-conversion (SOC) signal. In unsynchronized mode the ADC is usually triggered by software at the sample time intervals specified in the ADC block. For more information on configuring the specific parameters for this mode, see “Configuring Acquisition Window Width for ADC Blocks”.

In synchronized mode, the Event (EV) Manager associated with the same module as the ADC triggers the ADC. In this case, the ADC is synchronized with the pulse width modulator (PWM) waveforms generated by the same EV unit via the **ADC Start Event** signal setting. The **ADC Start Event** is set in the C281x PWM block. See that block for information on the settings.

Note The ADC cannot be synchronized with the PWM if the ADC is in cascaded mode.

Modes

The C281x ADC block supports ADC operation in dual and cascaded modes. In dual mode, either module `A` or module `B` can be used for the ADC block, and two ADC blocks are allowed in the model. In cascaded mode, both module `A` and module `B` are used for a single ADC block.

Version History

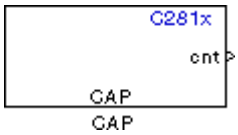
Introduced in R2016b

See Also

C281x PWM | C28x Hardware Interrupt

C281x CAP

Receive and log capture input pin transitions



Description

The C281x CAP module provides input capture functionality for systems where precise timing of external events is important. The C281x CAP block sets parameters for the capture units (CAPs) of the Event Manager (EV) module. The capture units log transitions detected on the capture unit pins by recording the times of the input signal transitions into a two-level deep FIFO stack. You can set the capture unit pins to detect rising edge, falling edge, either type of transition, or no transition. The cnt output of the block gives the captured value of the EV running timer.

The C281x chip has six capture units — three associated with each EV module. Capture units 1, 2, and 3 are associated with EVA and capture units 4, 5, and 6 are associated with EVB. Each capture unit is associated with a capture input pin.

Each group of EV module capture units can use one of two general-purpose (GP) timers on the target board. EVA capture units can use GP timer 1 or 2. EVB capture units can use GP timer 3 or 4. When a transition occurs, the module stores the value of the selected timer in the two-level deep FIFO stack.

The C281x CAP module shares GP Timers with other C281 blocks. For more information and guidance on sharing timers, see “Sharing General Purpose Timers Between C281x Peripherals”.

Note You can have up to two C281x CAP blocks in a model—one block for each EV module.

Outputs

This block has up to two outputs: a `cnt` (count) output and an optional, FIFO status `flag` output. The `cnt` output holds the value of the EV timer captured during the detected transitions. The `cnt` output gives the captured values of the running counter based on the value set in **Output data format** parameter. The status flag outputs are:

- `0` — The FIFO is empty. Either no captures have occurred or the previously stored captures have been read from the stack. (The binary version of this flag is `00`.)
- `1` — The FIFO has one entry in the top register of the stack. (The binary version of this flag is `01`.)
- `2` — The FIFO has two entries in the stack registers. (The binary version of this flag is `10`.)
- `3` — The FIFO has two entries in the stack registers and one or more captured values have been lost. This occurs because another capture occurred before the FIFO stack was read. This means that the FIFO stack is read when you execute the block as specified by your scheduling scheme synchronously, if a sample time is used or asynchronously, if triggered by an interrupt or an idle task. The new value is placed in the bottom register. The bottom register value is pushed to the top of the stack and the top value is pushed out of the stack. (The binary version of this flag is `11`.)

Parameters

Data Format Pane

Module

Select the Event Manager (EV) module to use:

- `A` — Use CAPs 1, 2, and 3.
- `B` — Use CAPs 4, 5, and 6.

Output overrun status flag

Select to output the status of the elements in the FIFO. The data type of the status flag is uint16.

Output data format

The type of data to output:

- `Send 2 elements (FIFO Buffer)` — Sends the latest two values. The output is updated when there are two elements in the FIFO, which is indicated by bit 13 or 11 or 9 being sent (CAP x FIFO). If the CAP is polled when fewer than two elements are captures, old values are repeated. The CAP registers are read as follows:

  1. The CAP x FIFO status bits are read and the value is stored in the status flag.
  2. The top value of the FIFO is read and stored in the output at index 0.
  3. The new top value of the FIFO (the previously stored bottom stack value) is read and stored in the output at index 1.

- `Send 1 element (oldest)` — Sends the older of the two most recent values. The output is updated when there is at least one element in the FIFO, which is indicated by the bits 13:12, or 11:10, or 9:8 being sent. The CAP registers are read as follows:

  1. The CAP x FIFO status bits are read and the value is stored in the status flag.
  2. The top value of the FIFO is read and stored in the output.

- `Send 1 element (latest)` — Sends the most recent value. The output is updated when there is at least one element in the FIFO, which is indicated by the bits 13:12, or 11:10, or 9:8 being sent. The CAP registers are read as follows:

  1. The CAP x FIFO status bits are read and the value is stored in the status flag.
  2. If the FIFO buffer contains two entries, the bottom value is read and stored in the output. If the FIFO buffer contains one entry, the top value is read and stored in the output.

Sample time

Time between outputs from the FIFO. If new data is not available, the previous data is sent.

Data type

Data type of the output data. Available options are `auto`, `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, and `boolean`. The `auto` option uses the data type of a connected block that outputs data to this block. If this block does not receive an input, `auto` sets the data type to `double`.

Note The output of the C281x CAP block can be vectorized.

CAP Panes

The CAP panes set parameters for individual CAPs. The particular CAP affected by a CAP pane depends on the EV module you selected:

- **CAP1** controls CAP 1 or CAP 4, for EV module A or B, respectively.
- **CAP2** controls CAP 2 or CAP 5, for EV module A or B, respectively.
- **CAP3** controls CAP 3 or CAP 6, for EV module A or B, respectively.

Enable CAP

Select to use the specified capture unit pin.

Edge Detection

Type of transition detection to use for this CAP. Available types are `Rising Edge`, `Falling Edge`, `Both Edges`, and `No transition`.

Time Base

Select which target board GP timer the CAP uses as a time base. CAPs 1, 2, and 3 can use `Timer 1` or `Timer 2`. CAPs 4, 5, and 6 can use `Timer 3` or `Timer 4`.

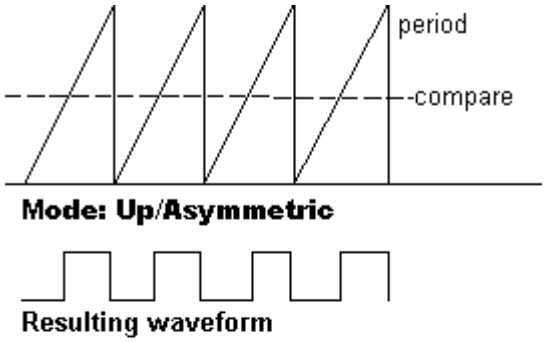
Clock source

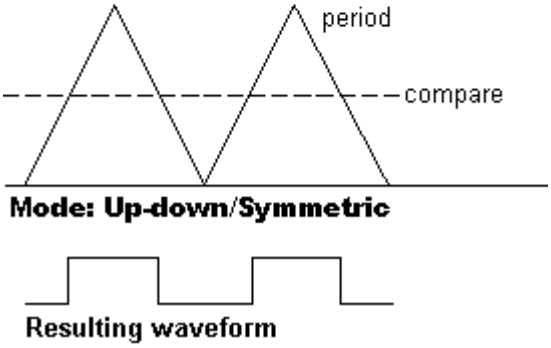
This option is available only for the CAP 3 pane. You can select `Internal` to use the internal time base. Also configure the **Counting mode**, **Timer prescaler**, and **Timer period source** for the internal time base.

Select `QEP circuit` to generate the input clock from the quadrature encoder pulse (QEP) submodule.

Counting mode

Select `Up` to generate an asymmetrical waveform output, or `Up-down` to generate a symmetrical waveform output, as shown in the following illustration.





The Counting mode is for the internal timer settings.

When you specify the **Counting mode** as `Up` (asymmetric) the waveform:

- Starts low
- Goes high when the rising period counter value matches the **Compare value**
- Goes low at the end of the period

When you specify the **Counting mode** as `Up-down` (symmetric) the waveform:

- Starts low
- Goes high when the increasing period counter value matches the **Compare value**
- Goes low when the decreasing period counter value matches the **Compare value**

Counting mode becomes unavailable when you set **Clock source** to `QEP circuit`.

Timer Prescaler

Clock divider factor by which to prescale the selected GP timer to produce the desired timer counting rate. Available options are `none`, `1/2`, `1/4`, `1/8`, `1/16`, `1/32`, `1/64`, and `1/128`. The following table shows the rates that result from selecting each option.

Scaling	Resulting Rate (µs)
none	0.01334

Scaling	Resulting Rate (µs)
1/2	0.02668
1/4	0.05336
1/8	0.10672
1/16	0.21344
1/32	0.42688
1/64	0.85376
1/128	1.70752

Note These rates assume a 75 MHz input clock.

Timer period source

Select `Specify via dialog` to enable the **Timer period** parameter. Select `Input port` to create a block input, **T1**, that accepts the timer period value.

Timer period

Set the length of the timer period in clock cycles. Enter a value from `0` to `65535`. The value defaults to `65535`.

If you know the length of a clock cycle, you can easily calculate how many clock cycles to set for the timer period. The following calculation determines the length of one clock cycle:

$$Sysclk(150MHz) \rightarrow HISPCLK(1/2) \rightarrow InputClockPrescaler(1/128)$$

In this calculation, you divide the System clock frequency of 150 MHz by the high-speed clock prescaler of 2. Then, you divide the resulting value by the timer control input clock prescaler, 128. The resulting frequency is 0.586 MHz. Thus, one clock cycle is 1/.586 MHz, which is 1.706 µs.

Post interrupt on CAP

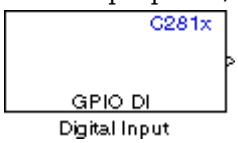
Check this check box to post an asynchronous interrupt on CAP.

See Also

C28x Hardware Interrupt

C281x GPIO Digital Input

General-purpose I/O pins for digital input



Library

C2000 Microcontroller Blockset/ F28004x

Description

This block configures the general-purpose I/O (GPIO) registers that control the GPIO shared pins for digital input. Each I/O port has one MUX register, which is used to select peripheral operation or digital I/O operation.

Note To avoid losing new settings, click **Apply** before changing the **IO Port** parameter.

Parameters

IO Port

Select the input/output port to use: `GPIOPA`, `GPIOPB`, `GPIOPD`, `GPIOPE`, `GPIOPF`, or `GPIOPG` and select the I/O Port bits to enable for digital input. (There is no GPIOPC port on the C281x.) If you select multiple bits, vector input is expected. Cleared bits are available for peripheral functionality. Multiple GPIO DI blocks cannot share the same I/O port.

Note The input function of the digital I/O and the input path to the related peripheral are enabled on the board. If you configure a pin as digital I/O, the corresponding peripheral function cannot be used.

The following tables show the shared pins.

GPIO A MUX

Bit	Peripheral Name (Bit =1)	GPIO Name (Bit = 0)
0	PWM1	GPIOA0
1	PWM2	GPIOA1
2	PWM3	GPIOA2
3	PWM4	GPIOA3
4	PWM5	GPIOA4
5	PWM6	GPIOA5
8	QEP1/CAP1	GPIOA8
9	QEP2/CAP2	GPIOA9
10	CAP3	GPIOA10

GPIO B MUX

Bit	Peripheral Name (Bit =1)	GPIO Name (Bit = 0)
0	PWM7	GPIOB0
1	PWM8	GPIOB1
2	PWM9	GPIOB2
3	PWM10	GPIOB3
4	PWM11	GPIOB4
5	PWM12	GPIOB5
8	QEP3/CAP4	GPIOB8
9	QEP4/CAP5	GPIOB9
10	CAP6	GPIOB10

Sample time

Time interval, in seconds, between consecutive input from the pins.

Data type

Data type of the data to obtain from the GPIO pins. The data is read as 16-bit integer data and then cast to the selected data type. Valid data types are `auto`, `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32` or `boolean`.

Note The width of the vectorized data output by this block is determined by the number of bits selected in the **Block Parameters** dialog box.

See Also

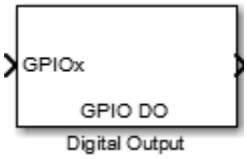
C281x GPIO Digital Output

C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F2838x-M4/F28004x/F28002x/F28003x GPIO Digital Input C280x/C2802x/C2803x/C2805x/C2806x/

C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F2838x-M4/F28004x/F28002x/F28003x GPIO Digital Output

C281x GPIO Digital Output

General-purpose I/O pins for digital output



Library

C2000 Microcontroller Blockset/ C281x

C2000 Microcontroller Blockset/F28004x

Description

This block configures the general-purpose I/O (GPIO) registers that control the GPIO shared pins for digital output. Each I/O port has one MUX register, which is used to select peripheral operation or digital I/O operation.

Note Fixed-point inputs are not supported for this block.

Note To avoid losing new settings, click **Apply** before changing the **IO Port** parameter.

Parameters

IO Port

Select the input/output port to use: `GPIOPA`, `GPIOPB`, `GPIOPD`, `GPIOPE`, `GPIOPF`, or `GPIOPG` and select the I/O Port bits to enable for digital input. (There is no GPIOPC port on the C281x.) If you select multiple bits, vector input is expected. Cleared bits are available for peripheral functionality. Multiple GPIO DO blocks cannot share the same I/O port.

Note The input function of the digital I/O and the input path to the related peripheral are enabled on the board. If you configure a pin as digital I/O, the corresponding peripheral function cannot be used.

The following tables show the shared pins.

GPIO A MUX

Bit	Peripheral Name (Bit =1)	GPIO Name (Bit = 0)
0	PWM1	GPIOA0
1	PWM2	GPIOA1
2	PWM3	GPIOA2
3	PWM4	GPIOA3
4	PWM5	GPIOA4
5	PWM6	GPIOA5
8	QEP1/CAP1	GPIOA8
9	QEP2/CAP2	GPIOA9
10	CAP3	GPIOA10

GPIO B MUX

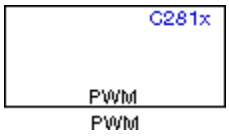
Bit	Peripheral Name (Bit =1)	GPIO Name (Bit = 0)
0	PWM7	GPIOB0
1	PWM8	GPIOB1
2	PWM9	GPIOB2
3	PWM10	GPIOB3
4	PWM11	GPIOB4
5	PWM12	GPIOB5
8	QEP3/CAP4	GPIOB8
9	QEP4/CAP5	GPIOB9
10	CAP6	GPIOB10

See Also

C281x GPIO Digital Input

C281x PWM

Pulse width modulators (PWMs)



Description

F2812 DSPs include a suite of pulse width modulators (PWMs) used to generate various signals. This block provides options to set the A or B module Event Managers to generate the waveforms you require. The twelve PWMs are configured in six pairs, with three pairs in each module.

The C281x PWM module shares GP Timers with other C281 blocks. For more information and guidance on sharing timers, see “Sharing General Purpose Timers Between C281x Peripherals”.

Note All inputs to the C281x PWM block must be scalar values.

Parameters

Timer Pane

Module

Specify which target PWM pairs to use:

- `A` — Displays the PWMs in module A (PWM1/PWM2, PWM3/PWM4, and PWM5/PWM6).
- `B` — Displays the PWMs in module B (PWM7/PWM8, PWM9/PWM10, and PWM11/PWM12).

Note PWMs in module A use Event Manager A, Timer 1, and PWMs in module B use Event Manager B, Timer 3.

Waveform period source

Source from which the waveform period value is obtained. Select `Specify via dialog` to enter the value in **Waveform period** or select `Input port` to use a value from the input port.

Note All inputs to the C281x PWM block must be scalar values.

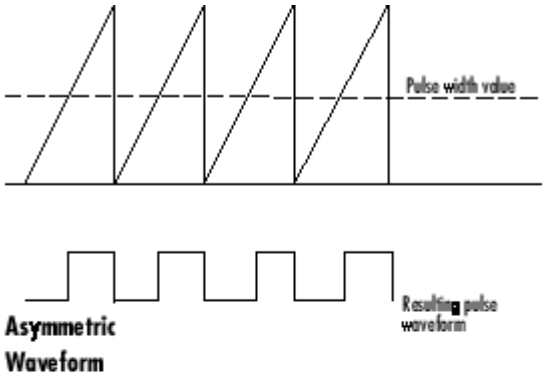
Waveform period

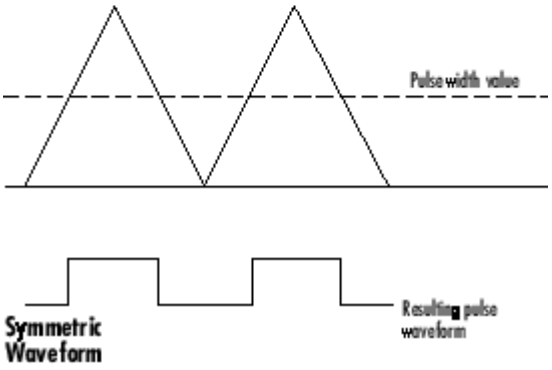
Period of the PWM waveform measured in clock cycles or in seconds, as specified in the **Waveform period units**.

Note The term *clock cycles* refers to the high-speed peripheral clock on the F2812 chip. This clock is 75 MHz by default because the high-speed peripheral clock prescaler is set to 2 (150 MHz/2).

Waveform type (counting mode)

Type of waveform to be generated by the PWM pair. The F2812 PWMs can generate two types of waveforms: `Asymmetric(Up)` and `Symmetric(Up-down)`. The following illustration shows the difference between the two types of waveforms.





Waveform period units

Units in which to measure the waveform period. Options are `Clock cycles`, which refer to the high-speed peripheral clock on the F2812 chip (75 MHz), or `Seconds`. Changing these units changes the **Waveform period** value and the **Duty cycle** value and **Duty cycle units** selection.

Timer prescaler

Divide the clock input to produce the desired timer counting rate.

Outputs Pane

Enable PWM#/PWM#

Check to activate the PWM pair. PWM1/PWM2 are activated via the Output 1 pane, PWM3/PWM4 are on Output 2, and PWM5/PWM6 are on Output 3.

Duty cycle source

Select `Specify via dialog` to use the dialog box to enter a **Duty cycle** value for the pair of PWM outputs. Select `Input port` to use the input port, **W#**, to enter a **Duty cycle** value for the pair of PWM outputs.

The input port **W1** corresponds to PWM1/PWM2. **W2** corresponds to PWM3/PWM4. **W3** corresponds to PWM5/6.

Note All inputs to the C281x PWM block must be scalar values.

Duty cycle

Set the ratio of the PWM waveform pulse duration to the PWM **Waveform period**.

Duty cycle units

Units for the duty cycle. Valid choices are `Clock cycles` and `Percentages`. Changing these units changes the **Duty cycle** value, and the **Waveform period** value and **Waveform period units** selection.

Note Using percentages can cause some additional computation time in generated code. This may or may not be noticeable in your application.

Logic Pane

Control logic source

Configure the control logic for all PWMs enabled on the Outputs tab. Valid settings are `Specify via dialog` (default setting) or to `Input port`.

`Specify via Dialog` enables **PWM control logic** settings for each PWM output:

- `Forced high` causes the pulse value to be high.

  `Active high` causes the pulse value to go from low to high.

  `Active low` causes the pulse value to go from high to low.

  `Forced low` causes the pulse value to be low.

`Input port` adds an input port to the PWM block for setting the C2000 ACTRX register. Each PWM uses 2 bits to set the following options:

- 00 Forced Low
- 01 Active Low
- 10 Active High
- 11 Forced High

Bits 11–0 of the 16–bit Compare Action Control Registers for module A control PWM1-6

Bits 11–0 of the 16–bit Compare Action Control Registers for module B control PWM1-6

For example: If a decimal value of 3222 is read at the input port while using PWM module A, the following PWM settings will be honored:

3222 = 0C96h = 110010010110b

So that:

- PW1: Active High
- PW2: Active Low

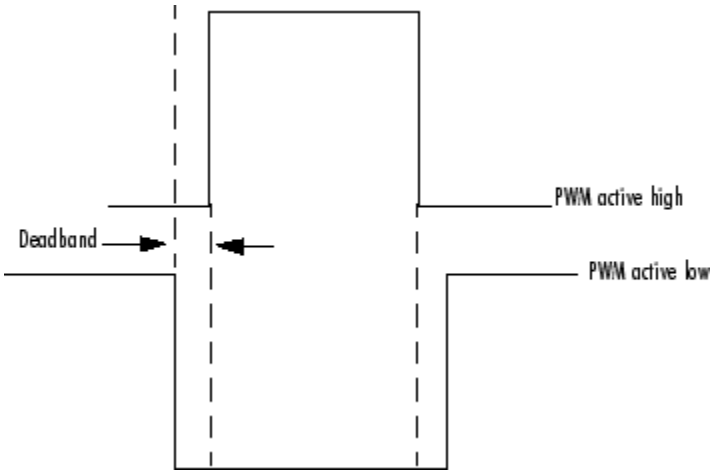
- PW3: Active Low
- PW4: Active High
- PW5: Forced Low
- PW6: Forced High

For more information, see the section on Compare Action Control Registers (ACTRA and ACTRB) in the Texas Instruments™ document “TMS320x281x DSP Event Manager (EV) Reference Guide”, literature number SPRU065.

Deadband Pane

Use deadband for PWM#/PWM#

Enables a deadband area without signal overlap at the beginning of particular PWM pair signals. The following figure shows the deadband area.



Deadband prescaler

Number of clock cycles, which, when multiplied by the Deadband period, determines the size of the deadband. Selectable values are 1, 2, 4, 8, 16, and 32.

Deadband period source

Source from which the deadband period is obtained. Select `Specify via dialog` to enter the values in the **Deadband period** field or select `Input port` to use a value, in clock cycles, from the input port.

Note All inputs to the C281x PWM block must be scalar values.

Deadband period

Value that, when multiplied by the Deadband prescaler, determines the size of the deadband. Selectable values are from 1 to 15.

ADC Control Pane

ADC start event

Controls whether this PWM and ADC associated with the same EV module are synchronized. Select `None` to disable synchronization or select an event to generate the source start-of-conversion (SOC) signal for the associated ADC.

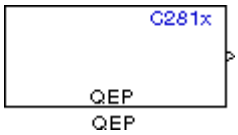
- `None` — The ADC and PWM are not synchronized. The EV does not generate an SOC signal and the ADC is triggered by software (that is, the A/D conversion occurs when the ADC block is executed in the software).
- `Underflow interrupt` — The EV generates an SOC signal for the ADC associated with the same EV module when the board's general-purpose (GP) timer counter reaches a hexadecimal value of FFFF.
- `Period interrupt` — The EV generates an SOC signal for the ADC associated with the same EV module when the value in GP timer matches the value in the period register. The value set in **Waveform period** above determines the value in the register.

  Note If you select `Period interrupt` and specify a sampling time less than the specified **(Waveform period)/(Event timer clock speed)**, zero-order hold interpolation will occur. (For example, if you enter 64000 as the waveform period, the period for the timer is 64000/75 MHz = 8.5333e-004. If you enter a **Sample time** in the C281x ADC dialog box that is less than this result, it will cause zero-order hold interpolation.)

- `Compare interrupt` — The EV generates an SOC signal for the ADC associated with the same EV module when the value in the GP timer matches the value in the compare register. The value set in **Duty cycle** above determines the value in the register.

C281x QEP

Quadrature encoder pulse circuit



Description

Each F2812 Event Manager has three capture units, which can log transitions on its capture unit pins. Event Manager A (EVA) uses capture units 1, 2, and 3. Event Manager B (EVB) uses capture units 4, 5, and 6.

The quadrature encoder pulse (QEP) circuit decodes and counts quadrature encoded input pulses on these capture unit pins. QEP pulses are two sequences of pulses with varying frequency and a fixed phase shift of 90 degrees (or one-quarter of a period). The circuit counts both edges of the QEP pulses, so the frequency of the QEP clock is four times the input sequence frequency.

The QEP, in combination with an optical encoder, is useful for obtaining speed and position information from a rotating machine. Logic in the QEP circuit determines the direction of rotation by which sequence is leading. For module A, if the QEP1 sequence leads, the general-purpose (GP) Timer counts up and if the QEP2 sequence leads, the timer counts down. The pulse count and frequency determine the angular position and speed.

The C281x QEP module shares GP Timers with other C281 blocks. For more information and guidance on sharing timers, see “Sharing General Purpose Timers Between C281x Peripherals”.

Parameters

Module

Specify which QEP pins to use:

- `A` — Uses QEP1 and QEP2 pins.
- `B` — Uses QEP3 and QEP4 pins.

Counting mode

Specify how to count the QEP pulses:

- `Counter` — Count the pulses based on GP Timer 2 (or GP Timer 4 for EVB).
- `RPM` — Count the rotations per minute.

Positive rotation

Defines whether to use `Clockwise` or `Counterclockwise` as the direction to use as positive rotation. This field appears only if you select `RPM`.

Initial count

Initial value for the counter. The value defaults to `0`.

Encoder resolution (pulse/revolution)

Number of QEP pulses per revolution. This field appears only if you select `RPM`.

Enable QEP index

Reset the QEP counter to zero when the QEP index input on CAP3_QEPI1 transitions from low to high.

Enable index qualification mode

Qualify the QEP index input on CAP3_QEPI1. Check that the levels on CAP1_QEP1 and CAP2_QEP2 are high before asserting the index signal as valid.

Timer period

Set the length of the timer period in clock cycles. Enter a value from `0` to `65535`. The value defaults to `65535`.

If you know the length of a clock cycle, you can easily calculate how many clock cycles to set for the timer period. The following calculation determines the length of one clock cycle:

$$Sysclk(150MHz) \rightarrow HISPCLK(1/2) \rightarrow InputClockPrescaler(1/128)$$

In this calculation, you divide the System clock frequency of 150 MHz by the high-speed clock prescaler of 2. Then, you divide the resulting value by the timer control input clock prescaler, 128. The resulting frequency is 0.586 MHz. Thus, one clock cycle is 1/.586 MHz, which is 1.706 µs.

Sample time

Time interval, in seconds, between consecutive reads from the QEP pins.

Data type

Data type of the QEP pin data. The circuit reads the data as 16-bit data and then casts it to the selected data type. Valid data types are `auto`, `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32` or `boolean`.

References

For more information on the QEP module, consult the following documents, available at the Texas Instruments Web site:

- *TMS320x280x, 2801x, 2804x Enhanced Quadrature Encoder Pulse (eQEP) Module Reference Guide*, Literature Number SPRU790
- *Using the Enhanced Quadrature Encoder Pulse (eQEP) Module in TMS320x280x, 28xxx as a Dedicated Capture Application Report*, Literature Number SPRAAH1

C281x Timer

Configure general-purpose timer in Event Manager module

Description

The C281x contains two event-manager (EV) modules. Each module contains two general-purpose (GP) timers. You can use these timers as independent time bases for various applications.

Use the C281x Timer block to set the periodicity of one GP timer and the conditions under which it posts interrupts. Each model can contain up to four C281x Timer blocks.

The C281x Timer module configures GP Timers that other C281 blocks share. For more information and guidance on sharing timers, see “Sharing General Purpose Timers Between C281x Peripherals”.

Parameters

ModuleTimer no

Select which of four possible timers to configure. Setting **Module** to `A` lets you select `Timer 1` or `Timer 2` in **Timer no**. Setting **Module** to `B` lets you select `Timer 3` or `Timer 4` in **Timer no**.

Clock source

When **Timer no** has a value of `Timer 2` or `Timer 4`, use this parameter to select the clock source for the event timer. You can choose either `Internal` or `QEP circuit`. When you select `Internal`, you can configure other options such as **Timer period source**, **Counting mode**, and **Timer prescaler**.

Timer period source

Select the source of the event timer period. Use `Specify via dialog` to set the period using **Timer period**. Select `Input port` to create an input, **T**, that accepts the value of the timer period in clock cycles, from `0` to `65535`. **Timer period source** becomes unavailable when **Clock source** is set to `QEP circuit`.

Timer period

Set the length of the timer period in clock cycles. Enter a value from `0` to `65535`. The value defaults to `10000`.

If you know the length of a clock cycle, you can easily calculate how many clock cycles to set for the timer period. The following calculation determines the length of one clock cycle:

$$Sysclk(150MHz) \rightarrow HISPCLK(1/2) \rightarrow InputClockPrescaler(1/128)$$

In this calculation, you divide the System clock frequency of 150 MHz by the high-speed clock prescaler of 2. Then, you divide the resulting value by the timer control input clock prescaler, 128. The resulting frequency is 0.586 MHz. Thus, one clock cycle is 1/.586 MHz, which is 1.706 µs.

Compare value source

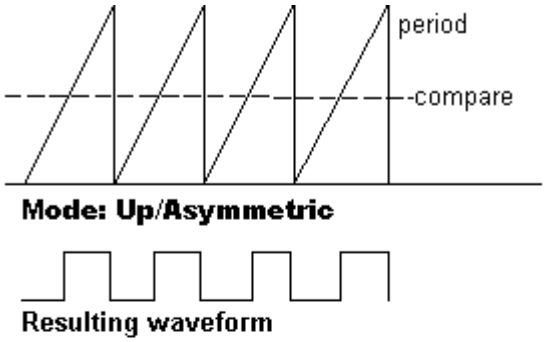
Select the source of the compare value. Use `Specify via dialog` to set the period using the **Compare value** parameter. Select `Input port` to create a block input, **W**, that accepts the value of the compare value, from `0` to `65535`.

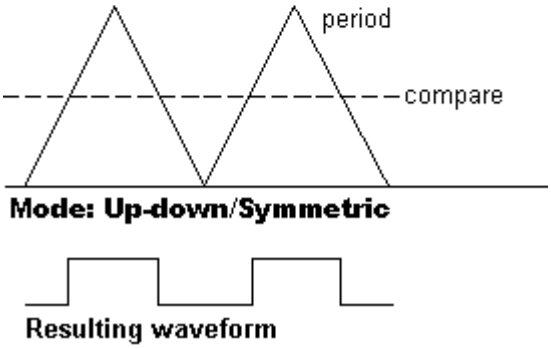
Compare value

Enter a constant value for comparison to the running timer value for generating interrupts. Enter a value from `0` to `65535`. The value defaults to `5000`. The timer only generates interrupts if you enable **Post interrupt on compare match**.

Counting mode

Select `Up` to generate an asymmetrical waveform output, or `Up-down` to generate a symmetrical waveform output, as shown in the following illustration.





When you specify the **Counting mode** as `Up` (asymmetric) the waveform:

- Starts low
- Goes high when the rising period counter value matches the **Compare value**
- Goes low at the end of the period

When you specify the **Counting mode** as `Up-down` (symmetric) the waveform:

- Starts low
- Goes high when the increasing period counter value matches the **Compare value**

- Goes low when the decreasing period counter value matches the **Compare value**

Counting mode becomes unavailable when **Clock source** is set to `QEP circuit`.

Timer prescaler

Divide the clock input to produce the desired timer counting rate.

Timer prescaler becomes unavailable when **Clock source** is set to `QEP circuit`.

Post interrupt on period match

Generate an interrupt when the value of the timer reaches its maximum value as specified in **Timer period**.

Post interrupt on underflow

Generate an interrupt when the value of the timer cycles back to `0`.

Post interrupt on overflow

Generate an interrupt when the value of the timer reaches its maximum, `65535`. Also set **Timer period** to `65535` for this parameter to work.

Post interrupt on compare match

Generate an interrupt when the value of the timer equals **Compare value**.

References

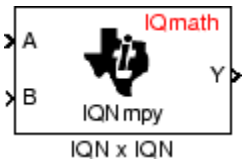
TMS320x281x DSP Event Manager (EV) Reference Guide, Literature Number: SPRU065, available from the Texas Instruments Web site.

See Also

C28x Hardware Interrupt

C2000 Division IQN

Divide IQ numbers



Library

C2000 Microcontroller Blockset/ Optimization/ C28x IQmath

Description

This block divides two numbers that use the same Q format, using the Newton-Raphson technique. The resulting quotient uses the same Q format at the inputs.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

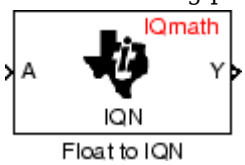
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 Float to IQN

Convert floating-point number to IQ number



Description

This block converts a floating-point number to an IQ number. The Q value of the output is specified in the dialog.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Q value

Q value from 1 to 30 that specifies the precision of the output

References

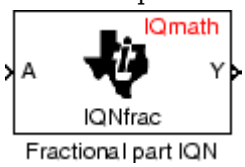
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 Fractional part IQN

Fractional part of IQ number



Description

This block returns the fractional portion of an IQ number. The returned value is an IQ number in the same IQ format.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

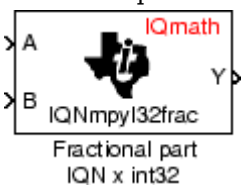
Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

C2000 Fractional part IQN x int32

Fractional part of result of multiplying IQ number and long integer



Description

This block multiplies an IQ input and a long integer input and returns the fractional portion of the resulting IQ number.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

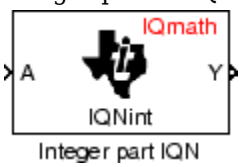
Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

C2000 Integer part IQN

Integer part of IQ number



Description

This block returns the integer portion of an IQ number. The returned value is a long integer.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

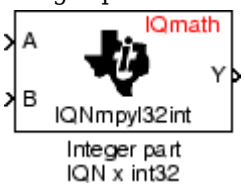
Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

C2000 Integer part IQN x int32

Integer part of result of multiplying IQ number and long integer



Description

This block multiplies an IQ input and a long integer input and returns the integer portion of the resulting IQ number as a long integer.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

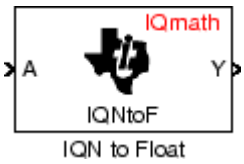
Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

C2000 IQN to Float

Convert IQ number to floating-point number



Description

This block converts an IQ input to an equivalent floating-point number. The output is a single floating-point number.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

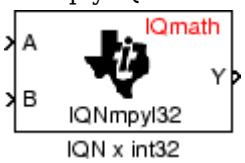
Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

C2000 IQN x int32

Multiply IQ number with long integer



Description

This block multiplies an IQ input and a long integer input and produces an IQ output of the same Q value as the IQ input.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

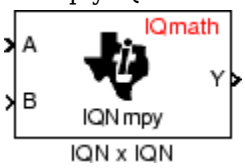
Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

C2000 IQN x IQN

Multiply IQ numbers with same Q format



Description

This block multiplies two IQ numbers. Optionally, it can also round and saturate the result.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Multiply option

Type of multiplication to perform:

- `Multiply` — Multiply the numbers.
- `Multiply with Rounding` — Multiply the numbers and round the result.
- `Multiply with Rounding and Saturation` — Multiply the numbers and round and saturate the result to the maximum value.

References

For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 IQN1 to IQN2

Convert IQ number to different Q format

Description

This block converts an IQ number in a particular Q format to a different Q format.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Q value

Q value from 1 to 30 that specifies the precision of the output

References

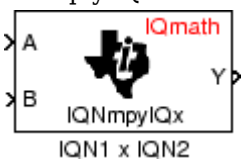
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 IQN1 x IQN2

Multiply IQ numbers with different Q formats



Description

This block multiples two IQ numbers when the numbers are represented in different Q formats. The format of the result is specified in the dialog box.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Q value

Q value from 1 to 30 that specifies the precision of the output

References

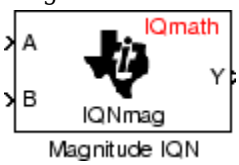
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 Magnitude IQN

Magnitude of two orthogonal IQ numbers



Description

This block calculates the magnitude of two IQ numbers using

$$\sqrt{a^2 + b^2}$$

The output is an IQ number in the same Q format as the input.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

References

For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The users guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

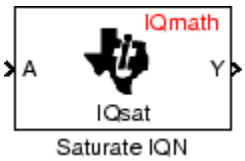
Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

C2000 Saturate IQN

Saturate IQ number



Description

This block saturates an input IQ number to the specified upper and lower limits. The returned value is an IQ number of the same Q value as the input.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Upper Limit

Maximum real-world value to which to saturate

Lower Limit

Minimum real-world value to which to saturate

References

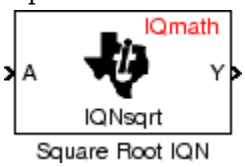
For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 Square Root IQN

Square root or inverse square root of IQ number



Description

This block calculates the square root or inverse square root of an IQ number and returns an IQ number of the same Q format. The block uses table lookup and a Newton-Raphson approximation.

Negative inputs to this block return a value of zero.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Function

Whether to calculate the square root or inverse square root

- `Square root (_sqrt)` — Compute the square root.
- `Inverse square root (_isqrt)` — Compute the inverse square root.

References

For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

C2000 Trig Fcn IQN

Sine, cosine, or arc tangent of IQ number

Trig Fcn IQN

Description

This block calculates basic trigonometric functions and returns the result as an IQ number. Valid Q values for `_IQsinPU` and `_IQcosPU` are 1 to 30. For all others, valid Q values are from 1 to 29.

Note The implementation of this block does not call the corresponding Texas Instruments library function during code generation. The TI function uses a global Q setting and the MathWorks code used by this block dynamically adjusts the Q format based on the block input. See “Using the IQmath Library” for more information.

Parameters

Function

Type of trigonometric function to calculate:

- `_IQsin` — Compute the sine `(sin(A))`, where A is in radians.
- `_IQsinPU` — Compute the sine per unit `(sin(2*pi*A))`, where A is in per-unit radians.
- `_IQcos` — Compute the cosine `(cos(A))`, where A is in radians.
- `_IQcosPU` — Compute the cosine per unit `(cos(2*pi*A))`, where A is in per-unit radians.

References

For detailed information on the IQmath library, see the user's guide for the *C28x IQmath Library - A Virtual Floating Point Engine*, Literature Number SPRC087, available at the Texas Instruments Web site. The user's guide is included in the zip file download that also contains the IQmath library (registration required).

See Also

C2000 Absolute IQN, C2000 Division IQN, C2000 Float to IQN, C2000 Fractional part IQN, C2000 Fractional part IQN x int32, C2000 Integer part IQN, C2000 Integer part IQN x int32, C2000 IQN to Float, C2000 IQN x int32, C2000 IQN x IQN, C2000 IQN1 to IQN2, C2000 IQN1 x IQN2, C2000 Magnitude IQN, C2000 Saturate IQN, C2000 Square Root IQN, C2000 Trig Fcn IQN

F2807x/F2837xD/F2837xS/F28004x/F28003x/ F2838x DAC

Configures the DAC to generate an analog output on the specified DAC channel A/B/C (12-bit)

Libraries:
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x

Description

Generate an analog output on the specified DAC channel A, B, or C for F2837x/F2807x/F28004x and F2838x/C28x processors. The block accepts a 12–bit value as an input in the range 0 to 4095. You can saturate a value higher than 4095 to 4095 with the **Saturate on input outflow** option.

The output pins of this block are multiplexed with the ADC block input. When you use a DAC block in your model, the corresponding channels of the ADC cannot be used as input. If used, the ADC samples the DAC output.

The pins that are shared between ADC and DAC are DACOUTA/ADCINA0, DACOUTB/ADCINA1, and DACOUTC/ADCINB1. The input to the DAC block can be double, float, int, and uint. The block typecasts the input to uint16.

Ports

Input

Port_1 — Input signal
`real` | `scalar`

The input port through which the block accepts the digital input value to convert it to analog signal.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `double`

Parameters

DAC channel — The DAC channel to generate analog output
12 (default)

Enter the DAC channel on which to generate the analog signal. The DAC channels that you can select are A, B, or C.

Saturate on input overflow (>4095) — The option to saturate the input value on input overflow
`off` | `on`

Select this check box to saturate the input value to 4095 when there is an input value is higher than 4095.

Compatibility Considerations

Earlier to R2021b, when **Saturate on input overflow (>4095)** parameter is deselected, an input value to the DAC greater than `65536` results the output to saturate at `65535`. Starting R2021b, the output wraps around i.e., for an input of `65580`, results in output `44 (65580-65536)`.

Version History

Introduced in R2016b

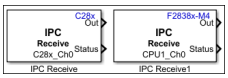
See Also

Topics

c280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x/F28002x/F28003x ePWM
C2802x/C2803x/C2805x/C2806x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x/F28002x/F28003x ADC

F2837xD/F2838x/F2838x-M4 IPC Receive

Receive data from either core



Libraries:
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2838x / C28x

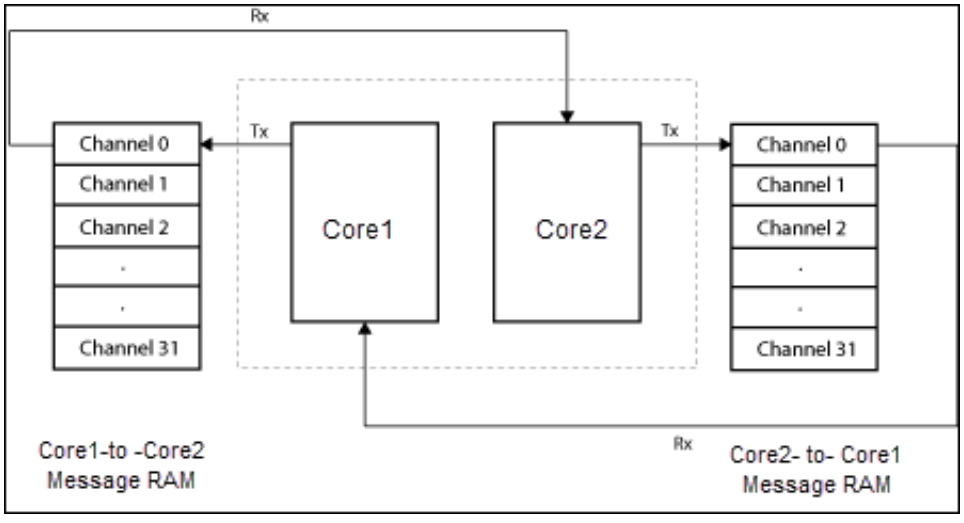
Description

The IPC Receive block receives and outputs data sent from one Core to the other.

Core1 transmits data to its allocated memory (Core1-to-Core2 Message RAM) and receives data from the allocated memory of Core2 (Core2-to-Core1 Message RAM). Similarly Core2 transmits data to its allocated memory (Core2-to-Core1 Message RAM) and receives data from allocated memory of Core1 (Core1-to-Core2 Message RAM). For F2838xD, Core1/Core2 can be CPU1,CPU2 or ARM Cortex-M4 (CM) and for F2837xD, Core1/Core2 can be CPU1 or CPU2.

If Core1 and Core2 are both C28x core, then the data and channel structure between two cores are allocated in Message RAM and the data array is allocated in Global Shared RAM. In C28x core, by default all the channel structures are created in Message RAM.

However, if one of the core is ARM Cortex M4 (applicable only for F2838x), then the data array is allocated only using Message RAM as global shared RAM is not available between cores. In order to accommodate more data, only required channel structures are created in Message RAM. Hence if the channel number used for transmit block in one core does not match with the receive block in other core the data transmission will not occur.



A hardware interrupt block can be used along with the IPC Receive block for receiving data based on hardware interrupts.

- **C28x processor** - Channels 0, 1, 2, and 3 are configured for hardware interrupts IPC0, IPC1, IPC2, and IPC3 respectively.

- **F2838x-M4 processor** - Channels 0, 1, 2, 3, 4, 5, 6 and 7 are configured for hardware interrupts IPC0, IPC1, IPC2, IPC3, IPC4, IPC5, IPC6 and IPC7 respectively.

These hardware interrupts can be set in the hardware interrupt block using these parameters: **CPU interrupt number** 1 and **PIE interrupt numbers** 13, 14, 15, and 16 respectively.

Ports

Output

Out — IPC receive
vector | scalar

Data read from the other Core.

Status — IPC receive status
0 | 1 | 2 | 3 | 4 | 6

The status port outputs one of these values:

- 0 — No errors
- 1 — Data not available
- 2 — Data type mismatch
- 4 — Data length mismatch
- 6 — Data type and Data length mismatch

Note When no data is transmitted, the IPC receive block receives 0 of uint16 data type and data type mismatch status is displayed

Parameters

Source — Source selected to receive data
`C28x (CPU1/CPU2)/CPU1` (default) | `ARM Cortex-M4 (CM)/CPU2`

The source at which you want to receive data. The source selection is based on the processor you choose. For F2838x(C28x) processor, the source is either C28x (CPU1/CPU2) or ARM Cortex-M4 (CM). For F2838x-M4 ARM core processor, the source is CPU1 or CPU2.

The IPC Receive block mask displays the current source and the channel selected. For example, if the block displays `C28x_Ch0`, then the source is `C28x (CPU1/CPU2)` and channel is `0`. Similarly if the IPC Receive block displays `CM_Ch1`, then the source is `ARM Cortex-M4 (CM)` and channel is `1`.

Note This parameter is only available for F2838x(C28x) and F2838x-M4 ARM core processors.

Channel — Channel selected to receive data
`0` (default) | 0–31

The channel at which you want to receive data. Each channel is a separate memory location in the shared memory.

Note The transmitter and receiver have 32 channels each to transmit and receive data. For data transmission and reception, the transmitter and receiver must be set to the same channel number.

Data type — Type of data to be received
`uint16` (default) | single | int8 | uint8 | int16 | int32 | uint32 | boolean

The type of data the block receives.

Vector data is stored in the global shared RAM, and the address of the data is stored in the MSGRAM.

Data length — Size of data to be received
`1` (default) | positive integer

The number of data units received at each sample time. If the data length is 1, the block interprets each incoming piece of data as a scalar value; if the data length is greater than 1, the block interprets each incoming piece of data as a vector with length equal to **Data length**. The maximum size for scalar and vector data is 32 bits.

Enable blocking — Specify if Core must wait to read data
`off` (default) | on

When enabled, the Core waits until data is available from the other Core.

Sample time — Interval at which block reads data
`0.001` (default) | `–1` | positive scalar

The time between data samples, measured in seconds. When you set this parameter to `-1`, Simulink determines the best sample time for the block based on the block context within the model.

Version History

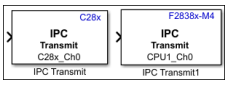
Introduced in R2018a

See Also

F2837xD/F2838x/F2838x-M4 IPC Transmit

F2837xD/F2838x/F2838x-M4 IPC Transmit

Transmit data to either core



Libraries:
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2838x / C28x

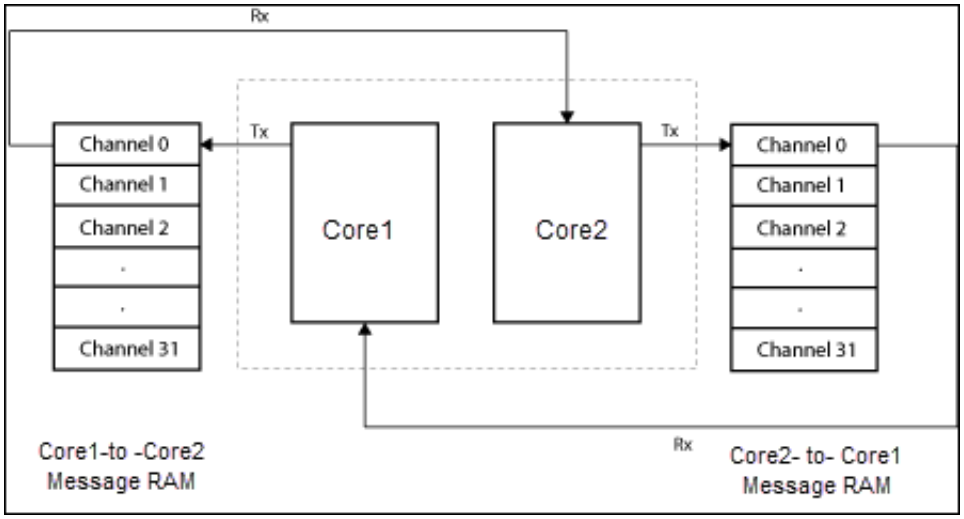
Description

The IPC Transmit block transmits data from one Core to the other.

Core1 transmits data to its allocated memory (Core1-to-Core2 Message RAM) and receives data from the allocated memory of Core2 (Core2-to-Core1 Message RAM). Similarly Core2 transmits data to its allocated memory (Core2-to-Core1 Message RAM) and receives data from allocated memory of Core1 (Core1-to-Core2 Message RAM). For F2838xD, Core1/Core2 can be CPU1,CPU2 or ARM Cortex-M4 (CM) and for F2837xD, Core1/Core2 can be CPU1 or CPU2.

If Core1 and Core2 are both C28x core, then the data and channel structure between two cores are allocated in Message RAM and the data array is allocated in Global Shared RAM. In C28x core, by default all the channel structures are created in Message RAM.

However, if one of the core is ARM Cortex M4 (applicable only for F2838x), then the data array is allocated only using Message RAM as global shared RAM is not available between cores. In order to accommodate more data, only required channel structures are created in Message RAM. Hence if the channel number used for transmit block in one core does not match with the receive block in other core the data transmission will not occur.



Ports

Input

Input — Data to be send to the other Core
uint16 | single | int8 | uint8 | int16 | int32 | uint32 | boolean

The port accepts data to be transmitted to the other Core as a vector or scalar.

Parameters

Destination — Destination selected to send data
`C28x (CPU1/CPU2)/CPU1` (default) | `ARM Cortex-M4 (CM)/CPU2`

The destination to which you want to send data. The destination selection is based on the processor you choose. For F2838x(C28x) processor, the destination is either C28x (CPU1/CPU2) or ARM Cortex-M4 (CM). For F2838x-M4 ARM core processor, the destination is CPU1 or CPU2.

The IPC Transmit block mask displays the current destination and the channel selected. For example, if the block displays `C28x_Ch0`, then the destination is `C28x (CPU1/CPU2)` and channel is `0`. Similarly if the IPC Transmit block displays `CM_Ch1`, then the destination is `ARM Cortex-M4 (CM)` and channel is `1`.

Note This parameter is only available for F2838x(C28x) and F2838x-M4 ARM core processors.

Channel — Channel selected to transmit data
`0` (default) | `0–31`

The channel from which you want to transmit data. Each channel is a separate memory location in the shared memory.

Note The transmitter and receiver have 32 channels each to transmit and receive data. For data transmission and reception, the transmitter and receiver must be set to the same channel number.

Enable blocking — Specify if Core must wait until sent data is read
`off` (default) | on

When enabled, after sending data, the Core waits until the other Core reads the data.

Version History

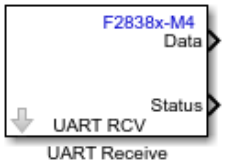
Introduced in R2018a

See Also

F2837xD/F2838x/F2838x-M4 IPC Receive

F2838x-M4 UART Receive

Receive data from the Universal Asynchronous Receiver Transmitter (UART) port



Libraries:
C2000 Microcontroller Blockset F2838x / M4

Description

Receive serial data from the Universal Asynchronous Receiver/ Transmitter (UART) port.

You can specify the ASCII characters for packaging your data with the additional package header and terminator. You can specify the data type and the data length that you want to receive using the block.

DMA interrupt will be used in the background for Data transfer from Receive FIFO to buffer. `UART_DMARx interrupt` will be triggered when any data will be received in the FIFO.

Ports

Output

Data — UART receive data
vector | scalar

Outputs the data read from the UART port.

Status — UART receive status
0 | 1 | 2 | 3 | 4 | 8

The status port outputs one of these values:

- 0 — represents no error in data reception
- 1 — represents frame error
- 2 — represents parity error
- 3 — represents data synchronization error
- 4 — represents a break in the data reception
- 8 — represents an overrun error.

Parameters

Additional package header — Prefix header
S (default)

Specify the additional package header to use as the prefix before the data packet to synchronize the data packets.

Additional package terminator — Suffix terminator
`E` (default)

Specify the additional package terminator to use as the suffix after the data packet to synchronize the data packets.

Data type — Type of data to be received
`uint8` (default) | `double` | `single` | `int8` | `int8` | `int16` | `int32` | `uint32` | `boolean`

Select the output data type.

Data length — Size of data to be received
`1` (default) | positive integer

Specify the data length to receive.

Enable blocking mode — Enable blocking mode for data transmission
`off` (default) | on

Enabling this option ensures that the FIFO buffer is checked for data availability before receiving the data.

Sample time — Interval at which block reads data
`0.1` (default)

Specify the sample time for receiving data. To execute this block asynchronously, set Sample Time to -1.

Version History

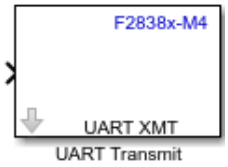
Introduced in R2020a

See Also

F2838x-M4 UART Transmit

F2838x-M4 UART Transmit

Send serial data to the Universal Asynchronous Receiver Transmitter (UART) port



Libraries:
C2000 Microcontroller Blockset F2838x / M4

Description

Send serial data through the Universal asynchronous Receiver/ Transmitter (UART) port. You can specify ASCII characters for packaging your data with the additional package header and terminator.

DMA will be used internally to copy data in FIFO.

Ports

Input

Data — UART send data
vector | scalar

The port sends the data to the UART port.

Parameters

Additional package header — Prefix header
S (default)

Specify the additional package header to use as the prefix before the data packet to synchronize the data packets.

Additional package terminator — Suffix terminator
E (default)

Specify the additional package terminator to use as the suffix after the data packet to synchronize the data packets.

Enable blocking mode — Enable blocking mode for data transmission
`off` (default) | on

Enabling this option ensures that the FIFO buffer is checked for data availability before sending the data.

Version History

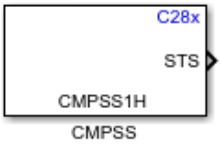
Introduced in R2020a

See Also

F2838x-M4 UART Receive

F2807x/F2837xD/F2837xS/F28004x/F2838x/F28002x/F28003x CMPSS

Compare two input voltages on comparator pins



Libraries:
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x

Description

The Comparator Subsystem consists of two modules, Comparator High (COMPH) and Comparator Low (COMPH). Each module generates a high digital output when the voltage on the first input pin (positive input) is greater than the voltage on the second input pin (negative input). And each module generates a low digital output when the voltage on the first input pin (positive input) is less than the voltage on the second input pin (negative input).

The second input pin can either be the external pin or the DAC module.

Ports

Input

DAC — DAC module
`scalar`

12-bit DAC reference value is used for the second input pin of the comparator.

The DAC range is between 0 to 4095. Any value outside the range is saturated.

Dependencies

The DAC port appears only when parameter:

- **Second input** is set to **Internal DAC**
- **DAC source select** is set to **DAC module**
- **Specify DAC/RAMP parameter(s) Via** is set to **Input port**

REF — Ramp reference value
`scalar`

Ramp reference value used by the ramp generator to create a ramp reference voltage for the second input pin of the comparator.

The REF value should be in the range 0 to 65535. Any value outside the range is saturated.

Dependencies

The REF port appears only when parameter:

- **Second input** is set to **Internal DAC**
- **DAC source select** is set to **RAMP module**
- **Specify DAC/RAMP parameter(s) Via** is set to **Input port**
- This port is available only for the COMPH module.

DEC — RAMP decrement value
`scalar`

Ramp decrement value used by the ramp generator to create a ramp reference voltage for the second input pin of the comparator.

DEC range is between 0 to 65535. Any value outside the range is saturated.

Dependencies

The DEC port appears only when parameter:

- **Second input** is set to **Internal DAC**
- **DAC source select** is set to **RAMP module**
- **Specify DAC/RAMP parameter(s) Via** is set to **Input port**
- This port is available only for the COMPH module.

DLY — RAMP delay value
`scalar`

Ramp delay value used by the ramp generator to create a ramp reference voltage for the second input pin of the comparator.

DLY range is between 0 to 8192. Any value outside the range is saturated.

Dependencies

The DLY port appears only when parameter:

- **Second input** is set to **Internal DAC**
- **DAC source select** is set to **RAMP module**
- **Specify DAC/RAMP parameter(s) Via** is set to **Input port**
- This port is available only for the COMPH module.

LCLR — Latch clear
`scalar`

Signal to clear the comparator latched output. Any value greater than 0 will clear the latch output.

Dependencies

The latch clear port appears only when **Enable latch clear** parameter is selected.

Output

STS — Comparator status output
`scalar`

The comparator module outputs 1, if the voltage on the first input pin is greater than the second input pin. Otherwise, it outputs 0.

LTH — Comparator latch
`scalar`

The comparator latch output is the tripped state of the CMPSS comparator after it is digitized and qualified by a digital filter.

The latched value can either be cleared by the software or PWMSYNC.

Dependencies

To enable this port, select the **Enable latch output** parameter.

Parameters

Comparator module — Indicates which module to use
`CMPSS1_COMPH` (default) | `CMPSSx_COMPL` | `CMPSSx_COMPH` | `where x ranges from 1 to 8`

Select which comparator module should be configured to output the comparison result.

Note Number of modules available will vary for different processors.

Second input — Source of second input pin
`External pin` (default) | `Internal DAC`

The voltage source of second input pin (negative input pin), specified as either External pin or Internal DAC.

DAC source select — Source of internal DAC
`DAC module` (default) | `RAMP module`

Select source of the internal DAC to generate voltage for the negative input pin, specified as either DAC module or RAMP module.

Dependencies

This parameter is available only for the COMPH module.

Specify DAC/RAMP parameter(s) via — Configure source DAC/RAMP value source
`Dialog` (default) | `Input port`

Select if the DAC or RAMP values are to be specified via the input port or from the dialog.

When you select input port, the block generates input ports for the comparator module.

When you set **DAC source select** parameter to **DAC module**, the block generates **DAC** port and when DAC source select is set to RAMP module it generates **REF**, **DEC**, and **DLY** port.

Parameter dependency

This parameter is available only for the COMPH module.

DAC value (DACH) — Specify DAC value
`0` (default)

Specify the DAC value for internal DAC to generate the voltage on the negative input.

DAC initial value — Specify DAC initial value
`0` (default)

Specify the DAC initial value for internal DAC to generate the voltage on the negative input.

Dependencies

The DAC port appears only when parameter:

- **Second input** is set to **Internal DAC**
- **DAC source select** is set to **DAC module**
- **Specify DAC/RAMP parameter(s) Via** is set to **Input port**

RAMP reference value — Specify RAMP reference value
`0` (default)

The RAMP reference value is in the range 0 to 65535. Any value outside the range is saturated.

The RAMP generator starts to decrement from the reference value.

Dependencies

The REF port appears only when parameter:

- **Second input** is set to **Internal DAC**
- **DAC source select** is set to **RAMP module**
- **Specify DAC/RAMP parameter(s) Via** is set to **Dialog**
- This port is available only for the COMPH module.

RAMP decrement value — Specify RAMP decrement value
`0` (default)

The RAMP decrement value is in range 0 to 65535. Any value outside the range is saturated.

The RAMP generator decrements the RAMP reference value in steps of decrement value.

Dependencies

The DEC port appears only when parameter:

- **Second input** is set to **Internal DAC**
- **DAC source select** is set to **RAMP module**
- **Specify DAC/RAMP parameter(s) Via** is set to **Dialog**
- This port is available only for the COMPH module.

RAMP delay value — Specify RAMP delay value
`0` (default)

The RAMP delay value is in range 0 to 8192. Any value outside the range is saturated.

The RAMP generator waits for the delay value before it starts to decrement the RAMP reference value in steps of RAMP decrement value.

Dependencies

The DLY port appears only when parameter:

- **Second input** is set to **Internal DAC**
- **DAC source select** is set to **RAMP module**
- **Specify DAC/RAMP parameter(s) via** is set to **Dialog**
- This port is available only for the COMPH module.

RAMP initial reference value — Specify RAMP initial reference value
`0` (default)

The RAMP initial reference value is in the range 0 to 65535. Any value outside the range is saturated.

The RAMP generator starts to decrement from the reference value.

Dependencies

The REF port appears only when parameter:

- **Second input** is set to **Internal DAC**
- **DAC source select** is set to **RAMP module**
- **Specify DAC/RAMP parameter(s) Via** is set to **Input port**
- This port is available only for the COMPH module.

RAMP initial decrement value — Specify RAMP decrement value
`0` (default)

The RAMP initial decrement value is in range 0 to 65535. Any value outside the range is saturated.

The RAMP generator decrements the RAMP reference value in steps of decrement value.

Dependencies

The DEC port appears only when parameter:

- **Second input** is set to **Internal DAC**
- **DAC source select** is set to **RAMP module**
- **Specify DAC/RAMP parameter(s) Via** is set to **Input port**
- This port is available only for the COMPH module.

RAMP initial delay value — Specify RAMP delay value
`0` (default)

The RAMP initial delay value is in range 0 to 8192. Any value outside the range is saturated.

The RAMP generator waits for the delay value before it starts to decrement the RAMP reference value in steps of RAMP decrement value.

Dependencies

The DLY port appears only when parameter:

- **Second input** is set to **Internal DAC**
- **DAC source select** is set to **RAMP module**
- **Specify DAC/RAMP parameter(s) via** is set to **Input port**
- This port is available only for the COMPH module.

Enable latch output — Comparator latch output
`off` (default) | `on`

The latched value of the digital filter output of the comparator.

Select this parameter to enable the LTH port.

Enable latch clear — Clear comparator latch value
`off` (default) | on

Latch clear signals to clear the comparator latch status signal. Any value greater than 0 will clear the latch signal.

Select this parameter to enable the LCLR port.

Sample time — Frequency at which data is read from comparator
`0.1` (default)

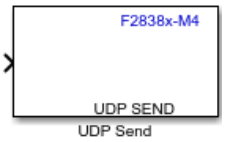
Use this parameter to specify the time interval between samples. To inherit sample time from an upstream block, set this parameter to -1.

Version History

Introduced in R2020a

F2838x-M4 UDP Send

Send UDP packets to UDP host



Libraries:
C2000 Microcontroller Blockset F2838x / M4

Description

The UDP Send block sends UDP packets to a UDP host. Use the block for stateless and connectionless data transmission.

The block sends packets from the port number specified in the **Local IP Port (-1 for automatic port assignment)** parameter. The IP address and the port number of the receiving host are specified in the **Remote IP address (255.255.255.255 for broadcast)** and **Remote IP Port** parameters.

You can choose to send UDP packets in the blocking or nonblocking mode.

Note

- Your antivirus software or firewall might block UDP traffic. Configure the software to allow traffic from a specific IP port number.
- Due to RAM memory limitations on the F2838x(ARM Cortex-M4), loading application to RAM is not supported for this block.
- CPU Timer 2 of F2838x Arm Cortex-M core (Connectivity Manager) provides time base to lwIP stack. It is configured to trigger an interrupt every 1 ms. This timer should not be re-configured if Ethernet blocks are being used in the model. If the corresponding interrupt is armed through Hardware Interrupt block, it will run the interrupt handler every 1 ms.

Ports

Input

Port_1 — Input data port
vector

The port accepts an array and sends it as UDP packets over an IP network to the receiving UDP host.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Parameters

Remote IP address (255.255.255.255 for broadcast) — IP address of receiving UDP host
`192.168.1.10` (default) | `255.255.255.255`

Specify the IP address or the host name to which the block sends the UDP packets. To broadcast packets to all the receiving hosts, specify `255.255.255.255`.

Remote IP Port — IP port on receiving UDP host
`25000` (default) | positive integer in the range `[1,65535]`

Specify the port number of the application on the receiving host to which you want to send the packets. Match the remote port number with the local port number of the receiving host.

Local IP Port (-1 for automatic port assignment) — IP port on sending UDP host
`-1` (default) |

Specify the port number of the application from which you want to send the packets.

When you specify the default value of `-1`, the block randomly selects a port from the available ports to send the data. If the receiving host expects UDP packets from a particular port number, specify the number of that port.

Wait until previous packet transmitted — Wait to sent new packets
off (default) | on

- `on` — When you select this parameter, the send operation runs in the blocking mode. In this mode, if the block is still transmitting the packets received in the previous time step, the block retains the data at the input port in the current time step and waits to send it.

  A task overrun occurs if the target hardware is still waiting for the requested data to be sent when the next send operation is scheduled to begin. To fix overruns increase the time step by using the **Sample time** parameter.
- `off` — When you clear this parameter, the send operation runs in the nonblocking mode. In this mode, if the block is still transmitting the packets received in the previous time step, the data at the input port in the current time step is dropped.

  In either mode, if the block is yet to establish the connection between the sending and receiving hosts. or if the connection is lost, the data at the input port is dropped.

Version History

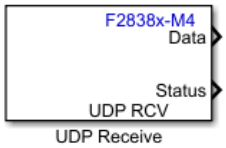
Introduced in R2020b

See Also

F2838x-M4 UDP Receive

F2838x-M4 UDP Receive

Receive UDP packets from UDP host



Libraries:
C2000 Microcontroller Blockset / / F2838x / M4

Description

The UDP Receive block receives UDP packets from a UDP host. Use the block for stateless and connectionless data exchange.

With each sample, the block outputs the contents of a UDP packet as a data vector of the size specified in the **Data size (N)** parameter.

The block receives the packets on the port number specified in the **Local IP Port** parameter from the IP address specified in the **Remote IP address (0.0.0.0 for accepting all)** parameter.

You can choose to receive the UDP packets in the blocking or nonblocking mode.

Note

- Your antivirus software or firewall might block UDP traffic. Configure the software to allow traffic from a specific IP port number.
- Due to RAM memory limitations on the F2838x(ARM Cortex-M4), loading application to RAM is not supported for this block.
- CPU Timer 2 of F2838x Arm Cortex-M core (Connectivity Manager) provides time base to lwIP stack. It is configured to trigger an interrupt every 1 ms. This timer should not be re-configured if Ethernet blocks are being used in the model. If the corresponding interrupt is armed through Hardware Interrupt block, it will run the interrupt handler every 1 ms.

Input/Output Ports

Output

Data — UDP packets received from sending UDP host
vector

At each sample time, the port outputs the contents of a packet received as a data vector of the size specified in the **Data size (N)** parameter.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

Status — Determine if requested data is received at the given time step
scalar

The port outputs `0` (success) when the received number of data elements is less than or equal to **Data size (N)** specified in the block. Otherwise, it outputs a nonzero value, indicating that no new data is available.

Data Types: `uint8`

Parameters

Local IP Port — IP port on the receiving UDP host
`25000` (default) | positive integer in the range `[1,65535]`

Specify the port number of the application from which you want to receive the packets. Match the local port number with the remote port number of the sending host.

Remote IP address (0.0.0.0 for accepting all) — IP address from which the block receives UDP packets
`192.1.168.10` (default) | `0.0.0.0`

Specify the IP address of the remote host from which the block receives packets. To receive packets from all the sending hosts, specify `0.0.0.0`.

Data type — Data type of elements in UDP packets
`uint8` (default) | `double` | `single` | `int8` | `int16` | `uint16` | `int32` | `uint32` | `boolean`

Select the data type of the elements in the UDP packets received by the block. The size of each element depends on its data type.

Data size (N) — Number of data elements in each UDP packet
`1` (default)

Specify the number of elements that you want to receive in each packet.

Ensure the following conditions are considered when data is sent to UDP Receive in the target to avoid data loss.

Note

- The UDP block can handle the maximum packet size (datagram size) of 1472 bytes. Ensure the packet size sent to the target is less than or equal to 1472 bytes.
- Ensure the size of the data sent to the target is less than or equal the packet size (datagram size).

Wait until data received — Wait until requested data is available
on (default) | off

- on — When you select this parameter, the read operation runs in the blocking mode. The read operation is blocked when the block is waiting for the requested data. If data is available, the block outputs the data. If data is not available, the block waits for the data.

  A task overrun occurs if the target hardware is still waiting for the data when the next read operation begins.

  To fix overruns, increase the time step by using the **Sample time** parameter.

- `off` — When you clear this parameter, the read operation runs in the nonblocking mode. When reading data, if data is not available, the block contains the packet received in the previous time step. In this mode, the block does not wait for the requested data to be available.

Sample time — How often this block reads packets from the sending UDP host
`0.1` (default)

Specify how often the block should read the port buffer. Enter a value greater than `0` or `-1` (for inherited sample time).

This value defaults to a sample time of `0.1` seconds. Smaller values require the processor to complete the same number of instructions in less time, and this can cause task overruns.

Version History

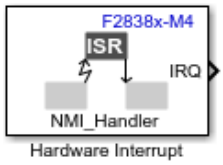
Introduced in R2020b

See Also

F2838x-M4 UDP Send on page 2-81

F2838x-M4 Hardware Interrupt

Trigger downstream function-call subsystem from interrupt service routine



Libraries:
C2000 Microcontroller Blockset F2838x / M4

Description

Use the F2838x-M4 Hardware Interrupt block to create an interrupt service routine (ISR) automatically in the generated code of your model. The ISR executes the downstream function-call subsystem associated with the block.

Using this block you can:

- Create ISRs on TI C2000 ARM Cortex-M4.
- Set ISR priority.
- Enable or disable interrupt preemption.
- Simulate the trigger of the interrupt and the downstream subsystem using a simulation input.

This block generates code only for the specified ISR. To change the configuration to enable the interrupt and specific triggering options use the settings of the chosen peripheral.

For example, to create an ISR for the UART peripheral on the Hardware Interrupt block, select `UART` in the **Interrupt group** parameter and `UART0INT_Handler` in the **Interrupt name** parameter. To create an ISR on the UART Transmit and the UART Receive blocks, set the **Interrupt name** parameter to `UART0INT_Handler`.

To trigger an ISR from a UART Transmit block, select the **Enable Transmit Interrupt** check box in **Configuration Parameters > Hardware Implementation > Target Hardware Resources > UART**. Selecting this check box has no effect if your model does not have a UART Transmit block.

An ISR from a UART Receive block is automatically triggered when you choose the necessary Hardware Interrupt block settings because the **Enable Receive Interrupt** check box is selected by default in **Configuration Parameters > Hardware Implementation > Target Hardware Resources > UART**.

Note Use MCAN Interrupt Status block inside the interrupt function call subsystem to clear the source of interrupt.

Input/Output Ports

Input

SimIRQ — Simulation interrupt input port
scalar

The interrupt block initiates a function call in simulation when you enable the SimIRQ input port. However, **SimIRQ** is ignored in the generated code.

Dependencies

To enable the **SimIRQ** port, select the **Add simulation input port** parameter.

Data Types: `Boolean`

Output

IRQ — Generate interrupt request
`Scalar`

The output of this block is a function call. The size of the function call line equals the number of interrupts the block is set to handle.

Parameters

Interrupt group — Select an interrupt group
`Cortex-M Exceptions` (default) | `MCANSS` | `ECAT` | `DCAN` | `EMAC` | `UART` | `SSI` | `I2C` | `USB` | `DMA` | `DMA` | `IPC` | `FMC` | `AES` | `Timer` | `Errors`

Interrupt group lists all the interrupt groups from your interrupt description file. Selecting an interrupt group changes the list of values in the **Interrupt name** parameter.

Interrupt name — Select ISR
`NMI_Handler` (default) | `HardFault_Handler` | **…**

The **Interrupt** name corresponds to the specific entry in the processor's interrupt vector table. The available ISRs depend on the interrupt group

Interrupt number — Read only parameter
`-14` (default) | `0` | `5` | **…**

This read-only parameter indicates the position of the selected ISR in the interrupt vector table of your target hardware.

Simulink task priority — Set priority of downstream function call
`30` (default) | positive integer or nonnegative integer

The value you specify in this parameter sets the priority of the downstream function-call subsystem. The simulink task priority of the selected (ISR) is relative to the model base rate priority.

Note The default model base sample rate priority is set to 40 with a lower priority value indicating a higher priority task. To achieve this the **Higher priority value indicates higher task priority** option is disabled in the **Solver** pane in the **Configuration Parameters**.

Disable interrupt pre-emption — Select to disable interrupt preemption
`off` (default) | `on`

By default, an interrupt can be preempted by a higher priority interrupt. Selecting this option allows low priority interrupts to complete their execution without being preempted by other interrupts.

Add simulation input port — Select to enable input port
off (default) | on

Select this option to enable the **SimIRQ** input. The Interrupt block initiates a function call in simulation when you enable the SimIRQ input port. However, **SimIRQ** is ignored in the generated code.

Version History

Introduced in R2020b

F2807x/F2837xD/F2837xS/F28004x/F28003x/F2838x SDFM

Configure filter channel for SDFM Module

Libraries:
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x

Description

The sigma delta filter module (SDFM) is a four-channel digital filter designed specifically for current measurement and resolver position decoding in motor control applications. Each input channel can receive an independent delta-sigma (ΔΣ) modulator bit stream. The bit streams are processed by four individually-programmable digital decimation filters.

The filter set includes a fast comparator (secondary filter) for immediate digital threshold comparisons for over-current and under-current monitoring and a primary data filter.

Each SDFM module consists of:

- Four independent configurable primary filter (data filter) units.
- Four independent, configurable secondary filter (comparator) units.
- Eight external pins (four sigma-delta data input pins and four sigma-delta clock input pins).
- Four different configurable modulator clock modes.

For more information on configuring filter channels, refer to SDFM Configuration Parameters on page 1-185.

Ports

Output

DFSTS — Data filter status
`0 | 1 | -1`

Status of the data filter returned as one of the following:

- 0 - Data filter not ready
- 1 - Data filter ready
- -1 - Modulator clock failure

DFLTX — Data filter
`scalar`

The primary filter data output. It is represented in either a 32-bit or a 16-bit format.

DFLTX - x represents the filter channel.

CFSTS — Comparator Status
`0 | 1 | 2 | 3 | 4 | 5 | -1`

The comparator status value differs for different processors.

The comparator status value

Comp Status Values	F2837x/07	F28004x	F2838x
0	No comparator event occurred	No comparator event occurred	No comparator event occurred
1	Lower threshold event (LLT)	Lower threshold event (LLT) No HLTZ	Comparator event 1 (CEVT1). No HLTZ
2	Higher threshold event (HLT)	Higher threshold event (HLT) No HLTZ	Comparator event 2 (CEVT2). No HLTZ
3	NA	HLTZ. No HLT and no LLT	HLTZ. No CEVT1 and no CEVT2
4	NA	HLTZ and LLT	CEVT1 and HLTZ
5	NA	HLTZ and HLT	CEVT2 and HLTZ
-1	Modulator clock failure	Modulator clock failure	Modulator clock failure

Dependencies

To enable this port, select the **Enable comparator filter output** parameter.

CFLTX — Comparator filter
`scalar`

The secondary data filter output. **CFLTX** - x represents the filter channel.

This comparator filter is available only for specific processors.

Dependencies

To enable this port, select the **Enable comparator filter output** parameter.

Parameters

SDFM module — Indicates which register to use
1 (default) | 2

Select the SDFM register that should be configured to output the result.

Note The F28004x processor has only one SDFM module.

Filter channel — Select filter module
`Filter1` (default) | `Filter2` | `Filter3` | `Filter4`

Each filter channel consists of an input control unit, a data filter unit, and a comparator filter unit. Each of these filter modules can be independently configured. The SDFM module consists of four primary filters and four secondary filters.

Data representation — Source of internal DAC
`16-bit` (default) | `32-bit`

The data filter output can be represented in either 32-bit or 16-bit format.

By default, the data filter output is represented in a 16-bit format. When the output is represented in a 16-bit format, required shifts are handled internally.

Sample time — Frequency at which data is read
`0.1` (default)

Use this parameter to specify the time interval between samples. To inherit sample time from an upstream block, set this parameter to `-1`.

Enable comparator filter output — Enables comparator output ports
`off` (default) | `on`

Select this option to enable the comparator status (CFSTS) and comparator filter (CFLTX) output ports.

Version History

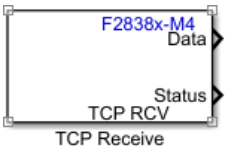
Introduced in R2020b

See Also

“Using Sigma Delta Filter Module (SDFM) to Measure the Analog Input Signal”

F2838x-M4 TCP Receive

Receive data over TCP/IP network from remote host



Libraries:
C2000 Microcontroller Blockset F2838x / M4

Description

The TCP Receive block receives data from a remote host or other target hardware over a network. The server(client mode) must be up and running before deploying the model that contains the TCP Receive block to the target.

The block outputs data either in blocking mode or in non-blocking mode. In blocking mode, the model blocks the execution while it waits for the requested data to be available. In non-blocking mode, the model runs continuously. To set the block in blocking mode, select the **Wait until data received** option.

At each time step, the **Data** port outputs data as a vector of the size specified in the **Data size (N)** parameter. The **Status** port outputs 0 or a nonzero value indicating whether new data is received. 0 at the **Status** port indicates that a valid data is received.

Note

- Your antivirus software or firewall might block TCP traffic. Configure the software to allow traffic from a specific IP port number.
- Due to RAM memory limitations on the F2838x(ARM Cortex-M4), loading application to RAM is not supported for this block.
- CPU Timer 2 of F2838x Arm Cortex-M core (Connectivity Manager) provides time base to lwIP stack. It is configured to trigger an interrupt every 1 ms. This timer should not be re-configured if Ethernet blocks are being used in the model. If the corresponding interrupt is armed through Hardware Interrupt block, it will run the interrupt handler every 1 ms.

Input/Output Ports

Output

Data — Output data
vector

At each time step, the port outputs data as a vector of the size specified in the **Data size (N)** parameter.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

Status — Requested data received at given time step
scalar

The port outputs `0` if the data is received at a given time step. Otherwise, it outputs a nonzero value, indicating that no new data is available.

Data Types: `uint8`

Parameters

Connection mode — Set the block as server or client
`Server` (default) | `Client`

Set the block as TCP/IP server or client.

When you set connection mode parameter to `Server`, provide a **Local IP Port**. The local port acts as the listening port on the TCP/IP server.

When you set connection mode parameter to `Client`, provide the **Server IP Address** and the **Server IP Port** of the TCP/IP server from which you want to receive the data.

Local IP Port — IP port on receiving host from which data is received
`25000` (default) | positive integer in the range [1,65535]

This local port number acts as a listening port on the TCP/IP server. Match the local port number with the remote port number of the sending host.

Server IP Address — Remote IP address of server from which data is received
`192.168.1.10` (default) | any valid IP address

Specify the IP address of the sending server from which the data is received.

Dependencies

This parameter appears only when **Connection mode** parameter is set to `Client`.

Server IP Port — Remote IP port on server from which data is received
`25002` (default) | positive integer in the range [1,65535]

Specify port number on the sending server from which data is received.

Dependencies

This parameter appears only when **Connection mode** is set to `Client`.

Data type — Data type of received data
`uint8` (default) | `double` | `single` | `int8` | `int16` | `uint16` | `int32` | `uint32` | `boolean`

Select the data type of the elements in the TCP data received by the block. The size of each element depends on its data type.

Data size (N) — Number of data bytes in received data
`1` (default) | positive integer

Specify number of data bytes to receive at each time step.

Wait until data received — Wait until requested data is available
`on` (default) | `off`

- `on` — When you select this parameter, the read operation runs in the blocking mode. The read operation is blocked when the block is waiting for the requested data. If data is available, the block outputs the data. If data is not available, the block waits for the data.

  A task overrun occurs if the target hardware is still waiting for the data when the next read operation begins.

  To fix overruns, increase the time step by using the **Sample time** parameter.
- `off` — When you clear this parameter, the read operation runs in the nonblocking mode. When reading data, if data is not available, the block contains the packet received in the previous time step (the block outputs 0). In this mode, the block does not wait for the requested data to be available.

Sample time — Interval at which block reads data from sending host
`0.1` (default) | nonnegative value

Specify how often the block should read the port buffer. Enter a value greater than `0` or `-1` (for inherited sample time).

This value defaults to a sample time of `0.1` seconds. Smaller values require the processor to complete the same number of instructions in less time, and this can cause task overruns.

Version History

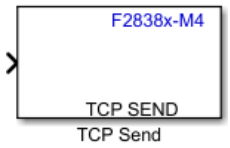
Introduced in R2020b

See Also

F2838x-M4 TCP Send

F2838x-M4 TCP Send

Send data over TCP/IP network to remote host



Libraries:
C2000 Microcontroller Blockset F2838x / M4

Description

The TCP Send block sends data to a remote host or another hardware board over a TCP/IP network. The server (client mode) must be up and running before deploying the model that contains the TCP Send block to the target.

The block sends data either in blocking mode or in non-blocking mode. In blocking mode, the model blocks the execution while it waits for the data to be sent completely. In non-blocking mode, the model runs continuously. To set the block in blocking mode, select the **Wait until previous packet transmitted** option.

Note

- Your antivirus software or firewall might block TCP traffic. Configure the software to allow traffic from a specific IP port number.
- Due to RAM memory limitations on the F2838x(ARM Cortex-M4), loading application to RAM is not supported for this block.
- CPU Timer 2 of F2838x Arm Cortex-M core (Connectivity Manager) provides time base to lwIP stack. It is configured to trigger an interrupt every 1 ms. This timer should not be re-configured if Ethernet blocks are being used in the model. If the corresponding interrupt is armed through Hardware Interrupt block, it will run the interrupt handler every 1 ms.

Input/Output Ports

Input

Port_1 — Input data
vector

The block accepts data specified as an *N*-by-1 array. The block sends this data over a TCP/IP network to the receiving host.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

Output

Status — Data transmitted at given time step
scalar

The port outputs `0` if the data is transmitted at a given time step. Otherwise, it outputs a nonzero value, indicating that data transmission is not successful.

Dependencies

This parameter appears only when **Output transmit status** parameter is enabled.

Data Types: `uint8`

Parameters

Connection mode — Set the block as server or client
`Server` (default) | `Client`

Set the block as a TCP/IP server or client.

When you set connection mode parameter to `Server`, provide a **Local IP Port**. The local port acts as the listening port on the TCP/IP server.

When you set connection mode parameter to `Client`, provide the **Server IP Address** and the **Server IP Port** of the TCP/IP server to which you want to send the data.

Server IP Address — Remote IP address of server to which to send data is sent
`192.168.1.10` (default) | `any valid IP address`

Specify the remote IP address of the receiving server to which the data is sent.

Dependencies

This parameter appears only when **Connection mode** parameter is set to `Client`.

Server IP Port — Remote IP port on server to which data is sent
`25002` (default) | positive integer in the range [1,65535]

Specify the port number on the receiving server to which data is sent.

Dependencies

This parameter appears only when **Connection mode** parameter is set to `Client`.

Local IP Port — IP port on sending host from which to send data
`25000` (default) | positive integer in the range [1,65535]

Specify the port number of the application from which to send the data. This local port acts as the listening port of the TCP/IP server.

Wait until previous packet transmitted — Wait until data received in previous time step is sent
`off` (default) | `on`

- `on` — When you select this parameter, the send operation runs in the blocking mode. In this mode, if the block is still transmitting the packets received in the previous time step, the block retains the data at the input port in the current time step and waits to send it.

  A task overrun occurs if the target hardware is still waiting for the requested data to be sent when the next send operation is scheduled to begin. To fix overruns increase the time step by using the **Sample time** parameter.

- `off` — When you clear this parameter, the send operation runs in the nonblocking mode. In this mode, if the block is still transmitting the packets received in the previous time step, the data at the input port in the current time step is dropped.

  In either mode, if the block is yet to establish the connection between the sending and receiving hosts. or if the connection is lost, the data at the input port is dropped.

Output transmit status — Option to display the transmit status during data transmission
`off` (default) | `on`

Select this option to display the transmit status during data transmission.

When you select the Output transmit status parameter, the block configures an output port. The port on the block is labeled as Status, indicating that the block outputs the status of the transmit operation at the output port.

Version History

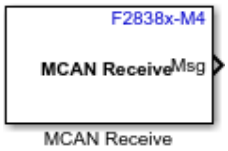
Introduced in R2020b

See Also

F2838x-M4 TCP Receive

F2838x-M4 MCAN Receive

Read data from CAN bus



Libraries:
C2000 Microcontroller Blockset / F2838x / M4

Description

The MCAN Receive block reads messages from a Controller Area Network (CAN) connected to the hardware.

In the Unpacked mode, the block outputs different fields of unpacked CAN Messages.

In the Packed mode, the block outputs a Simulink bus signal. To extract data from the Simulink bus signal, connect it to a CAN-FD Unpack block.

The MCAN Receive block receives messages from the CAN and delivers them to the Simulink model. It outputs one message or all messages at each time step, depending on the block parameters. The MCAN Receive block stores CAN messages received from the bus in a first-in, first-out (FIFO) or buffer. The FIFO or buffer delivers the messages to your model in the queued order at every time step.

Note

- For MCAN Receive block, if the data is not available, the output status shows the previously received data.
- When the **Output type** is packed and MCAN Receive block is connected to CAN FD Unpack block, the values of the `error`, `error state indicator (ESI)`, `reserved`, and `timestamp` fields of the received message are set to `0` and these values are reflected on the output of CAN FD Unpack block.

In receive output status we should mention the current behavior when data is not available. The output data will be retained to its previously received value in case of data is not available.

The number of elements to be stored in FIFO is configured in configuration parameters. In case of buffers, the new data will not be stored unless the old data is read.

The FIFO, blocking or overwrite mode settings are configured in the MCAN.

Specify the **Output type** and its properties using the block parameters dialog box. Configure additional properties of the CAN module in the Configuration Parameters. For more refer to “Model Configuration Parameters for Texas Instruments F2838x (ARM Cortex-M4)” on page 1-40.

Ports

Output

Msg — CAN Message
vector | scalar

The MCAN Receive block outputs the received CAN message (data and header) the Simulink bus signal. The CAN message received from the bus will be stored in a FIFO or buffer.

Dependencies

To enable this port, set **Output type** to `Packed`.

Data Types: `CAN Msg`

Data — Message data
vector | scalar

The block outputs data from the received CAN message. The maximum size of the data is `64` of `uint8` format.

Dependencies

To enable this port, set **Output type** to `Unpacked`.

Data Types: `uint8`

Status — Status of received message
scalar

The port outputs the message read status. The block outputs the status as `0` if it reads new message and `1` if it does not.

Dependencies

To enable this port select the **Output Status** parameter.

Data Types: `uint8`

CAN identifier(ID) — CAN message identifier
scalar

The port outputs a standard or extended from the received CAN message.

Dependencies

To enable this port, set **Output type** to `Unpacked`.

Data Types: `uint32`

Length(LEN) — CAN message length
scalar

The port outputs the length of the received CAN message in bytes.

Dependencies

To enable this port, set **Output type** to `Unpacked`.

Data Types: `uint8`

Remote transmission request(RTR) — Remote frame status
scalar

The port outputs the remote transmission status as one of these values.

- `1` - if the received CAN message is a remote frame
- `0` - if the received CAN message is a data frame

Dependencies

To enable this port, set **Output type** to `Unpacked`.

Data Types: `uint8`

Bit rate switch(BRS) — Bit rate switch status
scalar

The port outputs:

- `0` - when CAN FD frames are transmitted without bit rate switching
- `1` - when CAN FD frames are transmitted with bit rate switching

Dependencies

To enable this port, set **Output type** to `Unpacked`.

Data Types: `uint8`

Extended identifier(XTD) — Extended or standard identifier
scalar

The port outputs:

- `0` - if its 11-bit standard identifier
- `1` - if its 29-bit extended identifier

Dependencies

To enable this port, set **Output type** to `Unpacked`.

Data Types: `uint8`

Frame data format(FDF) — CAN frame data format
scalar

The port outputs the CAN frame data format as:

- `0` - Classic CAN
- `1` - CAN-FD

Dependencies

To enable this port, set **Output type** to `Unpacked`.

Data Types: `uint8`

Parameters

Read source — CAN receive message source
`Buffers` (default) | `FIFO 0` | `FIFO 1`

The CAN message received from the bus will be stored in FIFO or buffer.

The MCAN Receive block stores the message in a FIFO0, FIFO1 or buffer depending on the filter configuration you set in the hardware configuration parameters.

Buffer number (0-63) — Buffer number
`0` (default) | `1` | `2...`

Specify the number of the buffer in which you want the MCAN Receive block to store the received CAN message.

Dependencies

To enable this parameter, set **Read source** to `Buffers`.

Output Type — Message output type
`Packed` (default) | `Unpacked`

The output type of the MCAN Receive block is either `Packed` or `Unpacked`.

In the Unpacked mode, the block outputs the different fields of unpacked CAN message. In the unpacked mode, the MCAN Receive block has `7 (ID, XTD, FDF, BRS, LEN, RTR, Data)` output ports.

In the Packed mode, the block outputs a Simulink bus signal. To extract data from Simulink bus signal, connect it to the CAN-FD Unpack block.

Output Status — Enable output status
`off` (default) | `on`

When you select the **Output Status** parameter, the block configures the **Status** output port. The port outputs the read status.

Sample Time — Time interval to read message
0.1 (default) | -1 | nonnegative real value

Specify how often the block receives message, in seconds. When you specify this parameter as `-1`, Simulink determines the best sample time for the block based on the block context within the model.

Version History

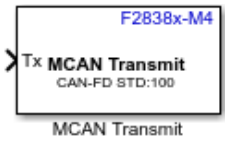
Introduced in R2021a

See Also

F2838x-M4 MCAN Transmit | “Model Configuration Parameters for Texas Instruments F2838x (ARM Cortex-M4)” on page 1-40

F2838x-M4 MCAN Transmit

Send serial data to CAN bus



Libraries:
C2000 Microcontroller Blockset / F2838x / M4

Description

The MCAN Transmit block sends messages to a Controller Area Network (CAN) connected to the hardware.

In the Raw data mode, the block accepts a 1-D array of type uint8. In the CAN msg mode, the block accepts a Simulink bus signal from CAN-FD Pack block.

Note In MCAN Transmit block, if the transmit FIFO is full, then the data is not transmitted.

Specify the **Data Format** and its properties using the block parameters dialog box. Configure the properties of CAN module in the Configuration Parameters. For more refer to “Model Configuration Parameters for Texas Instruments F2838x (ARM Cortex-M4)” on page 1-40.

Ports

Input

TX — Input data port
vector | scalar

The block accepts messages in the `Raw data` or `CAN Msg` format.

- `Raw data` – To accept the message as a `uint8` vector array, set the **Data format** as `Raw data`.
- `CAN Msg` – To accept the message in CAN message format, set the **Data format** as `CAN Msg`. You can create your messages using the CAN FD Pack block.

Data Types: `uint8` | `CAN Msg`

Output

Status — Status of CAN message transmission
scalar

Output port to display the CAN message transmission status. The status port outputs:

- `0` - When CAN message is added to transmit FIFO/Queue
- `1` - When CAN message is not added to transmit to FIFO/Queue

Dependencies

To enable this port, select the **Output Status** parameter.

Parameters

Data format — Data output type
`Raw data` (default) | `CAN Msg`

Select a type to the write message to transmit FIFO or queue. This message is later transmitted to CAN network connected to hardware.

- `Raw data` – To write message as a 1-by-*N* `uint8` array, select **Data Format** as `Raw data`.
- `CAN Msg` – To write message in CAN message format, select **Data Format** as `CAN Msg` and then perform these steps:

  1. Add a CAN FD Pack block from C2000 Microcontroller Blockset/Target Communication library to your model.
  2. Connect the output of the CAN FD Pack block to the input of the **MCAN Transmit** block
  3. Using the options in the **Data to be input as** list of the CAN Pack block, specify if you want to create your messages or you want to upload a CAN database file. If you choose to upload a CAN database file, the CAN FD Pack block inherits the message properties from the uploaded file.

Frame format — Frame format type
`CAN-FD` (default) | `Classic CAN`

The CAN frame format.

Dependencies

To enable this parameter, set **Data Format** to `Raw data`.

Identifier Type — Message identifier type
`Standard (11-bit identifier)` (default) | `Extended (29-bit identifier)`

The CAN message identifier type.

Dependencies

To enable this parameter, set **Data Format** to `Raw data`.

Identifier — Message identifier
`100` (default) | `numeric identifier of length 11 or 29 bits`

Identifier, which is 11 bits long for the standard frame size or 29 bits long for the extended frame size, specified in decimal, binary, or hex format. For binary and hex formats, use `bin2dec(' ')` and `hex2dec(' ')`, respectively, to convert the entry. The identifier is used to create CAN message transmitted to the CAN bus.

Dependencies

To enable this parameter, set **Data Format** to `Raw data`.

Length (bytes) — Length of message
`64` (default) | `positive integer less than or equal to 64(CAN-FD)` | `positive integer less than or equal to 8(Classic CAN)`

The length in bytes of data in the CAN message.

The length of message for the **Classic CAN** frame format is between `0-8` bytes, and for the **CAN-FD** frame format the range is between `0-64`. The blocks displays an error when there is mismatch between the length of data at transmission port and the length you specify here.

Dependencies

To enable this parameter, set **Data Format** to `Raw data`.

Remote Frame — Enable remote message
`off` (default) | `on`

Select this parameter to configure the CAN message as a remote message. The data at the input port is not considered for transmission.

Dependencies

To enable this parameter, set **Data format** to `Raw data` and **Frame format** to `Classic CAN`.

Enable bit rate switching — Enable bit rate switch message
`off` (default) | `on`

Select this parameter to configure the CAN message as a bit rate switch. The data in the CAN message will be transmitted at the data rate you specify in the hardware configuration parameter.

Dependencies

To enable this parameter, set **Data format** to `Raw data` and **Frame format** to `CAN-FD`.

Output Status — Enable output status
`off` (default) | `on`

When you select the **Output Status** parameter, the block configures the **Status** of the port. The port outputs CAN message transmission status.

Version History

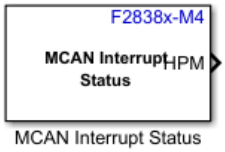
Introduced in R2021a

See Also

F2838x-M4 MCAN Receive | “Model Configuration Parameters for Texas Instruments F2838x (ARM Cortex-M4)” on page 1-40

F2838x-M4 MCAN Interrupt Status

Output specific bits of MCAN interrupt register or entire register



Libraries:
C2000 Microcontroller Blockset / F2838x / M4

Description

The MCAN Interrupt Status block outputs specific bits of MCAN_IR register or the entire MCAN_IR register. The register values are cleared after reading the specific bits or entire MCAN_IR register.

Select the `MCAN interrupt register output` parameter to output the entire MCAN interrupt register.

If `MCAN interrupt register output` parameter is not selected then the block outputs specific bits from MCAN_IR register based on the selection from `Transmit sources`, `Receive sources` and `General sources`.

Configure Receive, Transmit and other interrupt sources in configuration parameters. For more, “Model Configuration Parameters for Texas Instruments F2838x (ARM Cortex-M4)” on page 1-40.

Ports

Output

MCAN_IR — Status of MCAN interrupt register
uint32

Output port to display entire MCAN_IR register status as uint32 value.

The flags are set when one of the listed conditions is detected (edge-sensitive). The flags remain set until the host clears them. A flag is cleared by writing a `1` to the corresponding bit position. Writing a `0` has no effect. A hard reset will clear the register.

MCAN_IR Register

Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Type	RF0N	RF0W	RF0F	RF0L	RF1N	RF1W	RF1F	RF1L	HPM	TC	TCF	TFE	TEFN	TEFW	TEFF	TEFL	TSW	MRAF	TOO	DRX	RESERVED	BEU	ELO	EP	EW	BO	WDI	PEA	PED	ARA	RESERVED	RESERVED

Dependencies

To enable this port, select the **MCAN interrupt register output** parameter.

TEFN — Status of TX event FIFO new entry
boolean

Output port to display the transmit (TX) event first in, first out (FIFO) new entry status. The status port outputs:

- `0` - TX event FIFO unchanged
- `1` - TX handler wrote TX event FIFO element

Dependencies

To enable this port, select the **TX event FIFO new element** parameter.

TEFW — Status of TX event FIFO watermark reached
boolean

Output port to display the TX event FIFO watermark reached status. Once the status is cleared and the FIFO level is still above the watermark level, then the bit is not enabled. The bit is enabled only while going from low to high. The status port outputs:

- `0` - TX event FIFO fill level below watermark
- `1` - TX event FIFO fill level reached watermark

Dependencies

To enable this port, select the **TX event FIFO watermark** parameter.

TEFF — Status of TX event FIFO full
boolean

Output port to display the TX event FIFO full status. The status port outputs:

- `0` - TX event FIFO not full
- `1` - TX event FIFO full

Dependencies

To enable this port, select the **TX event FIFO full** parameter.

TEFL — Status of TX event FIFO element lost
boolean

Output port to display the TX event FIFO element lost status. The status port outputs:

- `0` - No TX Event FIFO element lost
- `1` - TX event FIFO element lost, also set after write attempt to TX event FIFO of size zero

Dependencies

To enable this port, select the **TX event FIFO element lost** parameter.

TC — Status of transmission completed
boolean

Output port to display the transmission completed status. The status port outputs:

- `0` - No transmission completed
- `1` - Transmission completed

Dependencies

To enable this port, select the **Transmission complete** parameter.

TCF — Status of transmission cancellation finished
boolean

Output port to display the transmission cancellation finished status. The status port outputs:

- `0` - No transmission cancellation finished
- `1` - Transmission cancellation finished

Dependencies

To enable this port, select the **Transmission cancellation finished** parameter.

TFE — Status of transmission FIFO empty
boolean

Output port to display the transmission FIFO empty status. The status port outputs:

- `0` - TX FIFO non-empty
- `1` - TX FIFO empty

Dependencies

To enable this port, select the **TX FIFO empty** parameter.

HPM — Status of high priority message
boolean

Output port to display the high priority message status. The status port outputs:

- `0` - No high priority message received
- `1` - High priority message received

Dependencies

To enable this port, select the **High priority message** parameter.

RF0N — Status of RX FIFO 0 new message
boolean

Output port to display the receive (RX) FIFO 0 new message status. The status port outputs:

- `0` - No new message written to RX FIFO 0
- `1` - New message written to RX FIFO 0

Dependencies

To enable this port, select the **RX FIFO 0 new message** parameter.

RF0W — Status of RX FIFO 0 watermark reached
boolean

Output port to display the receive (RX) FIFO 0 watermark reached status. Once the status is cleared and the FIFO level is still above the watermark level, then the bit is not enabled. The bit is enabled only while going from low to high. The status port outputs:

- `0` - RX FIFO 0 fill level below watermark
- `1` - RX FIFO 0 fill level reached watermark

Dependencies

To enable this port, select the **RX FIFO 0 watermark** parameter.

RF0FL — Status of RX FIFO 0 full
boolean

Output port to display the receive (RX) FIFO 0 full status. The status port outputs:

- `0` - RX FIFO 0 not full
- `1` - RX FIFO 0 full

Dependencies

To enable this port, select the **RX FIFO 0 full** parameter.

RF0ML — Status of RX FIFO 0 message lost
boolean

Output port to display the RX FIFO 0 message lost status. The status port outputs:

- `0` - No RX FIFO 0 message lost
- `1` - RX FIFO 0 message lost, also set after write attempt to RX FIFO 0 of size zero

Dependencies

To enable this port, select the **RX FIFO 0 message lost** parameter.

DRX — Status of dedicated RX buffer message
boolean

Message stored to dedicated RX buffer. The flag is set whenever a received message has been stored into a dedicated RX Buffer.

The status port outputs:

- `0` - No RX buffer updated
- `1` - At least one received message stored into an RX buffer

Dependencies

To enable this port, select the **Dedicated RX buffer message** parameter.

RF1N — Status of RX FIFO 1 new message
boolean

Output port to display the receive (RX) FIFO 1 new message status. The status port outputs:

- `0` - No new message written to RX FIFO 1
- `1` - New message written to RX FIFO 1

Dependencies

To enable this port, select the **RX FIFO 1 new message** parameter.

RF1W — Status of RX FIFO 1 watermark reached
boolean

Output port to display the receive (RX) FIFO 1 watermark reached status. Once the status is cleared and the FIFO level is still above the watermark level, then the bit is not enabled. The bit is enabled only while going from low to high. The status port outputs:

- `0` - RX FIFO 1 fill level below watermark
- `1` - RX FIFO 1 fill level reached watermark

Dependencies

To enable this port, select the **RX FIFO 1 watermark reached** parameter.

RF1FL — Status of RX FIFO 1 full
boolean

Output port to display the receive (RX) FIFO 1 full status. The status port outputs:

- `0` - RX FIFO 1 not full
- `1` - RX FIFO 1 full

Dependencies

To enable this port, select the **RX FIFO 1 full** parameter.

RF1ML — Status of RX FIFO 1 message lost
boolean

Output port to display the RX FIFO 1 message lost status. The status port outputs:

- `0` - No RX FIFO 1 message lost
- `1` - RX FIFO 1 message lost, also set after write attempt to RX FIFO 1 of size zero

Dependencies

To enable this port, select the **RX FIFO 1 message lost** parameter.

TSW — Status of timestamp wraparound
boolean

Output port to display the timestamp wraparound status. The status port outputs:

- `0` - No timestamp counter wrap-around
- `1` - Timestamp counter wrapped around

Dependencies

To enable this port, select the **Timestamp wraparound** parameter.

TOO — Status of timeout occurred
boolean

Output port to display the timeout occurred status. The status port outputs:

- `0` - No timeout
- `1` - Timeout reached

Dependencies

To enable this port, select the **Timeout occurred** parameter.

ELO — Status of error logging overflow
boolean

Output port to display the error logging overflow status. The status port outputs:

- `0` - CAN error logging counter did not overflow
- `1` - Overflow of CAN error logging counter occurred

Dependencies

To enable this port, select the **Error logging overflow** parameter.

EW — Status of error warning
boolean

Output port to display the error warning status. The status port outputs:

- `0` - Error warning status unchanged
- `1` - Error warning status changed

Dependencies

To enable this port, select the **Warning status** parameter.

WD — Status of watchdog event
boolean

Output port to display the watchdog event status. The status port outputs:

- `0` - No message RAM watchdog event occurred
- `1` - Message RAM watchdog event due to missing ready

Dependencies

To enable this port, select the **Watchdog event** parameter.

PED — Status of protocol error in data phase
boolean

Protocol error in data phase (Data Bit Time is used)

Output port to display the protocol error in data phase status. The status port outputs:

- `0` - No protocol error in data phase
- `1` - Protocol error in data phase detected

Dependencies

To enable this port, select the **Data protocol error** parameter.

MRAF — Status of message RAM access failure
boolean

The flag is set, when the RX Handler:

- Has not completed acceptance filtering or storage of an accepted message until the arbitration field of the following message has been received. In this case acceptance filtering or message storage is aborted and the RX Handler starts processing of the following message.
- Was not able to write a message to the Message RAM. In this case message storage is aborted.

In both cases the FIFO put index is not updated respectively. The new data flag for a dedicated RX Buffer is not set, a partly stored message is overwritten when the next message is stored to this location.

The flag is set when the Tx Handler could not read a message from the Message RAM in time. In this case message transmission is aborted. In case of a TX Handler access failure the MCAN is switched into Restricted Operation Mode.

Output port to display the message RAM access failure status. The status port outputs:

- `0` - No message RAM access failure occurred
- `1` - Message RAM access failure occurred

Dependencies

To enable this port, select the **Message RAM access failure** parameter.

BEU — Status of bit error uncorrected
boolean

Bit error uncorrected. Message RAM bit error detected, uncorrected. This bit is set when a double bit error is detected by the ECC aggregator attached to the Message RAM. An uncorrected Message RAM bit error sets CCCR.INIT to `1`. This is done to avoid transmission of corrupted data.

Output port to display the bit error uncorrected status. The status port outputs:

- `0` - No bit error detected when reading from Message RAM
- `1` - Bit error detected, uncorrected (e.g. parity logic)

Dependencies

To enable this port, select the **Bit error uncorrected** parameter.

EP — Status of error passive
boolean

Output port to display the error passive status. The status port outputs:

- `0` - Error passive status unchanged
- `1` - Error passive status changed

Dependencies

To enable this port, select the **Error passive status** parameter.

BO — Status of bus off
boolean

Output port to display the bus off status. The status port outputs:

- `0` - Bus off status unchanged
- `1` - Bus off status changed

Dependencies

To enable this port, select the **Bus off status** parameter.

PEA — Status of arbitration protocol error
boolean

Protocol error in arbitration phase (Nominal Bit Time is used)

Output port to display the arbitration protocol error status. The status port outputs:

- `0` - No protocol error in arbitration phase
- `1` - Protocol error in arbitration phase detected

Dependencies

To enable this port, select the **Arbitration protocol error** parameter.

ARA — Status of access to reserved address
boolean

Output port to display the reversed address access status. The status port outputs:

- `0` - No access to reserved address occurred
- `1` - Access to reserved address occurred

Dependencies

To enable this port, select the **Reversed address access** parameter.

Parameter

MCAN interrupt register output — Enable to output entire MCAN interrupt register
`off` (default) | `on`

Select this parameter to output the entire MCAN interrupt register.

When you select the **MCAN interrupt register output** parameter, the block configures an output port, **MCAN_IR**. The port outputs the status of MCAN interrupt register output.

If `MCAN interrupt register output` parameter is not selected then the block outputs specific bits from MCAN_IR register based on the selection from `Transmit sources`, `Receive sources` and `General sources`.

Transmit sources

TX event FIFO new element — Enable TX event FIFO new element status
`off` (default) | `on`

When you select the **TX event FIFO new element** parameter, the block configures an output port, **TEFN**. The port outputs the status of TX event FIFO new entry.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

TX event FIFO watermark — Enable TX event FIFO watermark status
`off` (default) | `on`

When you select the **TX event FIFO watermark** parameter, the block configures an output port, **TEFW**. The port outputs the status of TX event FIFO watermark reached.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

TX event FIFO full — Enable TX event FIFO full status
`off` (default) | `on`

When you select the **TX event FIFO full** parameter, the block configures an output port, **TEFF**. The port outputs the status of TX event FIFO full.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

TX event FIFO element lost — Enable TX event FIFO element lost status
`off` (default) | `on`

When you select the **TX event FIFO element lost** parameter, the block configures an output port, **TEFL**. The port outputs the status of TX event FIFO element lost.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

Transmission complete — Enable transmission complete status
`off` (default) | `on`

When you select the **Transmission complete** parameter, the block configures an output port, **TC**. The port outputs the status of transmission completed.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

Transmission cancellation finished — Enable transmission cancellation finished status
`off` (default) | `on`

When you select the **Transmission cancellation finished** parameter, the block configures an output port, **TCF**. The port outputs the status of transmission cancellation finished.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

TX FIFO empty — Enable Transmission FIFO empty status
`off` (default) | `on`

When you select the **TX FIFO empty** parameter, the block configures an output port, **TFE**. The port outputs the status of transmission FIFO empty.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

Receive sources

High priority message — Enable high priority message status
`on` (default) | `off`

When you select the **High priority message** parameter, the block configures an output port, **HPM**. The port outputs the status of high priority message.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

RX FIFO 0 new message — Enable RX FIFO 0 new message status
`off` (default) | `on`

When you select the **RX FIFO 0 new message** parameter, the block configures an output port, **RF0N**. The port outputs the status of RX FIFO 0 new message.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

RX FIFO 0 watermark — Enable RX FIFO 0 watermark reached status
`off` (default) | `on`

When you select the **RX FIFO 0 watermark** parameter, the block configures an output port, **RF0W**. The port outputs the status of RX FIFO 0 watermark reached.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

RX FIFO 0 full — Enable RX FIFO 0 full status
`off` (default) | `on`

When you select the **RX FIFO 0 full** parameter, the block configures an output port, **RF0FL**. The port outputs the status of RX FIFO 0 full.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

RX FIFO 0 message lost — Enable RX FIFO 0 message lost status
`off` (default) | `on`

When you select the **RX FIFO 0 full** parameter, the block configures an output port, **RF0ML**. The port outputs the status of RX FIFO 0 message lost.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

Dedicated RX buffer message — Enable dedicated RX buffer message status
`off` (default) | `on`

When you select the **Dedicated RX buffer message** parameter, the block configures an output port, **DRX**. The port outputs the status of dedicated RX buffer message.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

RX FIFO 1 new message — Enable RX FIFO 1 new message status
`off` (default) | `on`

When you select the **RX FIFO 1 new message** parameter, the block configures an output port, **RF1N**. The port outputs the status of dedicated RX FIFO 1 new message.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

RX FIFO 1 watermark — Enable RX FIFO 1 watermark reached status
`off` (default) | `on`

When you select the **RX FIFO 1 watermark** parameter, the block configures an output port, **RF1W**. The port outputs the status of RX FIFO 1 watermark reached.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

RX FIFO 1 full — Enable RX FIFO 1 full status
`off` (default) | `on`

When you select the **RX FIFO 1 full** parameter, the block configures an output port, **RF1FL**. The port outputs the status of RX FIFO 1 full.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

RX FIFO 1 message lost — Enable RX FIFO 1 message lost status
`off` (default) | `on`

When you select the **RX FIFO 1 message lost** parameter, the block configures an output port, **RF1ML**. The port outputs the status of RX FIFO 1 message lost.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

General sources

Timestamp wraparound — Enable timestamp wraparound status
`off` (default) | `on`

When you select the **Timestamp wraparound** parameter, the block configures an output port, **TSW**. The port outputs the status of timestamp wraparound.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

Timeout occurred — Enable timeout occurred status
`off` (default) | `on`

When you select the **Timeout occurred** parameter, the block configures an output port, **TOO**. The port outputs the status of timeout occurred.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

Error logging overflow — Enable error logging overflow
`off` (default) | `on`

When you select the **Error logging overflow** parameter, the block configures an output port, **ELO**. The port outputs the status of error logging overflow.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

Warning status — Enable error warning status
`off` (default) | `on`

When you select the **Warning status** parameter, the block configures an output port, **EW**. The port outputs the status of error warning.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

Watchdog event — Enable watchdog event status
`off` (default) | `on`

When you select the **Watchdog event** parameter, the block configures an output port, **WD**. The port outputs the status of watchdog event.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

Data protocol error — Enable data protocol error status
`off` (default) | `on`

When you select the **Data protocol error** parameter, the block configures an output port, **PED**. The port outputs the status of protocol error in data phase (Data Bit Time is used).

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

Message RAM access failure — Enable message RAM access failure status
`off` (default) | `on`

When you select the **Message RAM access failure** parameter, the block configures an output port, **MRAF**. The port outputs the status of message RAM access failure.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

Bit error uncorrected — Enable bit error uncorrected status
`off` (default) | `on`

When you select the **Bit error uncorrected** parameter, the block configures an output port, **BEU**. The port outputs the status of bit error uncorrected.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

Error passive status — Enable error passive status
`off` (default) | `on`

When you select the **Error passive status** parameter, the block configures an output port, **EP**. The port outputs the status of bit error uncorrected.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

Bus off status — Enable bus off status
`off` (default) | `on`

When you select the **Bus off status** parameter, the block configures an output port, **BO**. The port outputs the status of bus off status.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

Arbitration protocol error — Enable arbitration protocol error status
`off` (default) | `on`

When you select the **Arbitration protocol error** parameter, the block configures an output port, **PEA**. The port outputs the status of arbitration protocol error.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

Reversed address access — Enable reversed address access status
`off` (default) | `on`

When you select the **Reversed address access** parameter, the block configures an output port, **ARA**. The port outputs the status of access to reserved address.

Dependencies

To enable this parameter, the **MCAN interrupt register output** parameter must be disabled.

Sample time — Time interval to read message
`-1` (default) | `0.1`

Specify how often the block receives message, in seconds. When you specify this parameter as -1, Simulink determines the best sample time for the block based on the block context within the model.

Version History

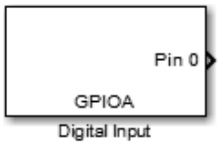
Introduced in R2021b

See Also

F2838x-M4 MCAN Receive | “Model Configuration Parameters for Texas Instruments F2838x (ARM Cortex-M4)” on page 1-40

F28M35x/F28M36x GPIO Digital Input

Read the logical value of a GPIO pin configured as input



Libraries:
C2000 Microcontroller Blockset F28M35x/ M3
C2000 Microcontroller Blockset F28M36x/ M3

Description

Read the logical value of a configured GPIO pin.

This block configures the general-purpose I/O (GPIO) MUX registers that control the operation of GPIO shared pins for digital input. Each I/O port has one MUX register that selects peripheral operation or digital I/O operation (the default). When a pin is configured for digital input, you cannot use the same pin for digital output or peripheral operation. You can configure the pull-up and the open-drain options for the individual digital input pins. To configure, go to **Configuration Parameters > Hardware Implementation > Target Hardware Resources** and select the appropriate GPIO group.

Ports

Output

Pin # — GPIO pin status
scalar | vector

The port outputs the status of the digital pin you select in the **Pin number** parameter.

Data Types: `Boolean`

Parameters

GPIO port — Port to read GPIO pin status
`GPIOA` (default) | `GPIOBGPIOC…`

Select one of GPIO ports to get the GPIO pin status.

Pin number — GPIO pin whose status you want to read
`Pin 0` (default) | `Pin 1` | `Pin 2` | `...`

Select the pin number of the GPIO pin whose status you want to read.

Sample time — Frequency at which block reads input pin values
`-1` (default) | `0.1`

Specify how often the block receives message, in seconds. When you specify this parameter as -1, Simulink determines the best sample time for the block based on the block context within the model.

Version History

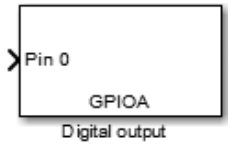
Introduced in R2016a

See Also

F28M35x/F28M36x GPIO Digital Output

F28M35x/F28M36x GPIO Digital Output

Set the logical value of a GPIO pin configured as output



Libraries:
C2000 Microcontroller Blockset F28M35x/ M3
C2000 Microcontroller Blockset F28M36x/ M3

Description

Set the logical value of the configured GPIO pin.

Configure individual general-purpose input/output (GPIO) pins to operate as digital outputs. When a pin is configured for digital output, it cannot operate as a digital input or connect to peripheral I/O signals.

Ports

Input

Pin # — Status of GPIO pin
scalar | vector

Specify a value at this port to set the status of GPIO pin(s).

Data Types: `Boolean`

Parameters

GPIO port — Port to write GPIO pin status
`GPIOA` (default) | `GPIOBGPIOC…`

Select one of GPIO ports to which you want to write the GPIO pin status.

Pin number — GPIO pin to write status
`Pin 0` (default) | `Pin 1` | `Pin 2` | `...`

Select the pin number of the GPIO pin whose status you want to write.

Initially set GPIO pin — GPIO pin status during model initialization
`on` (default) | `off`

Select this option to set the GPIO pin status during model initialization.

Version History

Introduced in R2016a

See Also

F28M35x/F28M36x GPIO Digital Input

Hardware Interrupt

Trigger downstream function-call subsystem from interrupt service routine

Libraries:
C2000 Microcontroller Blockset F28M35x/ M3
C2000 Microcontroller Blockset F28M36x/ M3

Description

Use the Hardware Interrupt block to create an interrupt service routine (ISR) automatically in the generated code of your model. The ISR executes the downstream function-call subsystem associated with the block.

Using this block you can:

- Create ISRs on TI C2000 Concerto processor.
- Set ISR priority.
- Enable or disable interrupt preemption.
- Simulate the trigger of the interrupt and the downstream subsystem using a simulation input.

This block generates code only for the specified ISR. To change the configuration to enable the interrupt and specific triggering options use the settings of the chosen peripheral.

For example, to create an ISR for the UART peripheral on the Hardware Interrupt block, select `UART` in the **Interrupt group** parameter and `UART0INT_Handler` in the **Interrupt name** parameter. To create an ISR on the UART Transmit and the UART Receive blocks, set the **Interrupt name** parameter to `UART0INT_Handler`.

To trigger an ISR from a UART Transmit block, select the **Enable Transmit Interrupt** check box in **Configuration Parameters > Hardware Implementation > Target Hardware Resources > UART**. Selecting this check box has no effect if your model does not have a UART Transmit block.

An ISR from a UART Receive block is automatically triggered when you choose the necessary Hardware Interrupt block settings because the **Enable Receive Interrupt** check box is selected by default in **Configuration Parameters > Hardware Implementation > Target Hardware Resources > UART0**.

Input/Output Ports

Input

SimIRQ — Simulation interrupt input port
scalar

The interrupt block initiates a function call in simulation when you enable the SimIRQ input port. However, **SimIRQ** is ignored in the generated code.

Dependencies

To enable the **SimIRQ** port, select the **Add simulation input port** parameter.

Data Types: `Boolean`

Output

IRQ — Generate interrupt request
`Scalar`

The output of this block is a function call. The size of the function call line equals the number of interrupts the block is set to handle.

Parameters

Interrupt group — Select an interrupt group
`Timers` (default) | `Cortex-M Exceptions` | `MCANSS` | `ECAT` | `DCAN` | `EMAC` | `UART` | `SSI` | `I2C` | `USB` | `DMA` | `DMA` | `IPC` | `FMC` | `AES` | `Errors`

Interrupt group lists all the interrupt groups from your interrupt description file. Selecting an interrupt group changes the list of values in the **Interrupt name** parameter.

Interrupt name — Select ISR
`Timer0a_Handler` (default) | `NMI_Handler` | `HardFault_Handler` | **…**

The **Interrupt** name corresponds to the specific entry in the processor's interrupt vector table. The available ISRs depend on the interrupt group

Interrupt number — Read only parameter
`19` (default) | `0` | `5` | **…**

This read-only parameter indicates the position of the selected ISR in the interrupt vector table of your target hardware.

Simulink task priority — Set priority of downstream function call
`30` (default) | `positive integer or nonnegative integer`

The value you specify in this parameter sets the priority of the downstream function-call subsystem. The simulink task priority of the selected (ISR) is relative to the model base rate priority.

Note The default model base sample rate priority is set to 40 with a lower priority value indicating a higher priority task. To achieve this the **Higher priority value indicates higher task priority** option is disabled in the **Solver** pane in the **Configuration Parameters**.

Disable interrupt pre-emption — Select to disable interrupt preemption
`off` (default) | `on`

By default, an interrupt can be preempted by a higher priority interrupt. Selecting this option allows low priority interrupts to complete their execution without being pre-empted by other interrupts.

Add simulation input port — Select to enable input port
off (default) | on

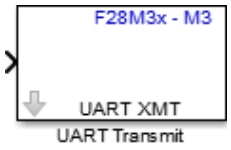
Select this option to enable the **SimIRQ** input. The Interrupt block initiates a function call in simulation when you enable the SimIRQ input port. However, **SimIRQ** is ignored in the generated code.

Version History

Introduced in R2017a

F28M35x/F28M36x UART Transmit

Transmit serial data to the Universal Asynchronous Receiver Transmitter (UART) port



Libraries:
C2000 Microcontroller Blockset F28M35x/ M3
C2000 Microcontroller Blockset F28M36x/ M3

Description

Transmit serial data through the Universal asynchronous Receiver/ Transmitter (UART) port. You can specify ASCII characters for packaging your data with the additional package header and terminator.

Ports

Input

Data — UART send data
vector | scalar

The port sends the data to the UART port.

DMA will be used internally to copy data in FIFO.

Parameters

UART Port — Select UART port for data transmission
`UART0` (default) | `UART1` | `UART2` | `UART3` | `UART4`

Select a port number from UART0 to UART4 for data transmission.

The UART Port selection allows access to the different UART modules present on the processor.

Additional package header — Prefix header
`S` (default)

Specify the additional package header to use as the prefix before the data packet to synchronize the data packets.

Additional package terminator — Suffix terminator
`E` (default)

Specify the additional package terminator to use as the suffix after the data packet to synchronize the data packets.

Enable blocking mode — Enable blocking mode for data transmission
`off` (default) | on

Enabling this option ensures that the FIFO buffer is checked for data availability before sending the data.

Version History

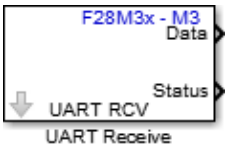
Introduced in R2016b

See Also

F28M35x/F28M36x UART Receive

F28M35x/F28M36x UART Receive

Receive data from the Universal Asynchronous Receiver Transmitter (UART) port



Libraries:
C2000 Microcontroller Blockset F28M35x/ M3
C2000 Microcontroller Blockset F28M36x/ M3

Description

Receive serial data from the Universal Asynchronous Receiver/ Transmitter (UART) port.

You can specify the ASCII characters for packaging your data with the additional package header and terminator. You can specify the data type and the data length that you want to receive using the block.

DMA interrupt will be used in the background for Data transfer from Receive FIFO to buffer. `UART_DMARx interrupt` will be triggered when any data will be received in the FIFO.

Ports

Output

Data — UART receive data
vector | scalar

Outputs the data read from the UART port.

Status — UART receive status
0 | 1 | 2 | 3 | 4 | 8

The status port outputs one of these values:

- 0 — represents no error in data reception
- 1 — represents frame error
- 2 — represents parity error
- 3 — represents data synchronization error
- 4 — represents a break in the data reception
- 8 — represents an overrun error.

Parameters

UART Port — Select UART port for data transmission
`UART0` (default) | `UART1` | `UART2` | `UART3` | `UART4`

Select a port number from UART0 to UART4 for data transmission.

The UART Port selection allows access to the different UART modules present on the processor.

Additional package header — Prefix header
`S` (default)

Specify the additional package header to use as the prefix before the data packet to synchronize the data packets.

Additional package terminator — Suffix terminator
`E` (default)

Specify the additional package terminator to use as the suffix after the data packet to synchronize the data packets.

Data type — Type of data to be received
`uint8` (default) | `double` | `single` | `int8` | `int8` | `int16` | `int32` | `uint32` | `boolean`

Select the output data type.

Data length — Size of data to be received
`1` (default) | positive integer

Specify the data length to receive.

Enable blocking mode — Enable blocking mode for data transmission
`off` (default) | on

Enabling this option ensures that the FIFO buffer is checked for data availability before receiving the data.

Sample time — Interval at which block reads data
`0.1` (default)

Specify the sample time for receiving data. To execute this block asynchronously, set Sample Time to -1.

Version History

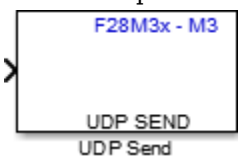
Introduced in R2016b

See Also

F28M35x/F28M36x UART Transmit

F28M35x/F28M36x UDP Send

Send UDP packets to UDP host



Library

C2000 Microcontroller Blockset/ F28M35x/ M3

C2000 Microcontroller Blockset/ F28M36x/ M3

Description

Send UDP packets to a remote UDP host. Use UDP Send block for stateless, connectionless, and byte oriented data transmission.

You can send one dimensional array of data type uint8, uint16, int16, uint32, int32, single, or double.

Parameters

Remote IP address (255.255.255.255 for broadcast)

Specify the IP address of the remote UDP host for data transmission. For broadcasting to all the remote hosts, enter ‘255.255.255.255’.

Remote IP Port

Specify the port number of the remote UDP host for data transmission.

Local IP Port (-1 for automatic port assignment)

Enter the port number of the local UDP host for data transmission.

Wait until previous packet transmitted

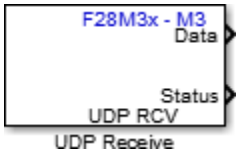
Select this check box to transmit a UDP packet, only after the previous packet is transmitted to avoid data overlap. Leave the default value ‘-1’ to allow automatic port assignment.

See Also

F28M35x/F28M36x UDP Receive

F28M35x/F28M36x UDP Receive

Receive UDP packets from the specified UDP host



Library

C2000 Microcontroller Blockset/ F28M35x/ M3

C2000 Microcontroller Blockset/ F28M36x/ M3

Description

Receive UDP packets from the specified remote host. Use the UDP Receive block to output the UDP packets received as one dimensional array of size defined in **Data size** parameter. Use UDP Receive block for stateless, connectionless, and byte oriented data transmission.

You can specify the remote UDP host in the **Remote IP address (0.0.0.0. for accepting all)** parameter to send the UDP packets to the local UDP host specified in the **Local IP Port** parameter.

The status port on the block indicates if the data is good or not.

The different values returned by the status port are as follows:

0 - represents no error in data reception

1 - represents an error in data reception.

Parameters

Local IP Port

Enter the local UDP host IP port to receive UDP packets.

Remote IP address (0.0.0.0 for accepting all)

Enter the IP address of the remote UDP host in 0.0.0.0 format to receive UDP packets. For accepting data from any valid IP address, enter ‘0.0.0.0’.

Data type

Select the data type of the UDP packets.

Data size (N)

Enter the data size of the data type you have selected in the **Data type** parameter.

Wait until data received

Select this check box to receive a UDP packet only after the previous packet is received to avoid data overlap.

Sample time

Specify how often this block should read the port buffer. Enter a value greater than zero.

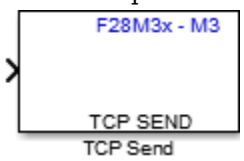
This value defaults to a sample time of 0.01 seconds. Smaller values require the processor to complete the same number of instructions in less time, which can cause task overruns.

See Also

F28M35x/F28M36x UDP Send

F28M35x/F28M36x TCP Send

Send TCP packets to a TCP host on TCP/IP network



Library

C2000 Microcontroller Blockset/ F28M35x/ M3

C2000 Microcontroller Blockset/ F28M36x/ M3

Description

Send TCP packets to another TCP host over the TCP/IP network. Use TCP Send block for connection-oriented, stateful, and stream based data transmission. Also, the TCP Send block guarantees the data transmission.

Using this block, you can send one dimensional array of data type uint8, uint16, int16, uint32, int32, single, or double.

Parameters

Local IP Port

Enter the IP port of the local host for data transmission.

Wait until previous packet transmitted

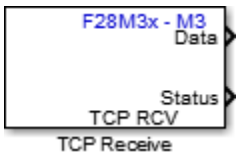
Select this check box to send a TCP/IP data packet only after the previous packet is transmitted.

See Also

F28M35x/F28M36x TCP Receive

F28M35x/F28M36x TCP Receive

Receive TCP packets from TCP host on TCP/IP network



Library

C2000 Microcontroller Blockset/ F28M35x/ M3

C2000 Microcontroller Blockset/ F28M36x/ M3

Description

Receive TCP packets from another TCP host over the TCP/IP network. Use TCP Receive block for connection-oriented, stateful, and stream based data transmission. Also, the TCP Receive block guarantees the data transmission.

This block outputs data received as an array of the size specified in the **Data size** parameter.

The status port on the block indicates if the data is good or not.

The different values returned by the status port are as follows:

0 - represents no error in data reception

1 - represents an error in data reception.

Parameters

Local IP Port

Enter the IP port number of the local host to receive data.

Data type

Select the data type of the TCP packet.

Data size (N)

Enter the data size of the data type you have selected in the **Data type** parameter.

Wait until data received

Select this check box to receive a TCP packet only after the previous packet is received to avoid data overlap.

Sample time

Specify how often this block should read the port buffer. Enter a value greater than zero.

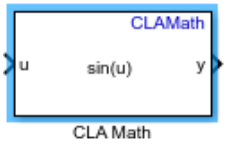
This value defaults to a sample time of 0.01 seconds. Smaller values require the processor to complete the same number of instructions in less time, which can cause task overruns.

See Also

F28M35x/F28M36x TCP Send

CLA Math

Implements CLA Math functions from CLA Math library



Libraries:
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x

Description

The CLA Math implements CLA Math functions from CLA Math library. CLA Math functions include trigonometric, logarithmic, exponential, and power functions.

Note The CLA Math block only works within the model configured for CLA.

Trigonometric functions accept input in radians. PU Trigonometric functions accept input in per unit.

The block accepts only input of datatype single and outputs data of type single.

Input/Output Ports

Input

u — Input data
vector

The block accepts data specified as an *N*-by-1 array.

Data Types: `single`

Output

y — Output data
scalar

The blocks outputs data of single as *N*-by-1 array . The block outputs the value of the selected function parameter.

Data Types: `single`

Parameters

Function — Select CLA Math function
`CLAsin` (default) | `CLAcos` | `CLAatan`

Select the CLA Math functions which includes trigonometric, logarithmic, exponential and power functions.

Version History

Introduced in R2023a

See Also

C28x CLA Task | CLA Subsystem | CLA Task Manager

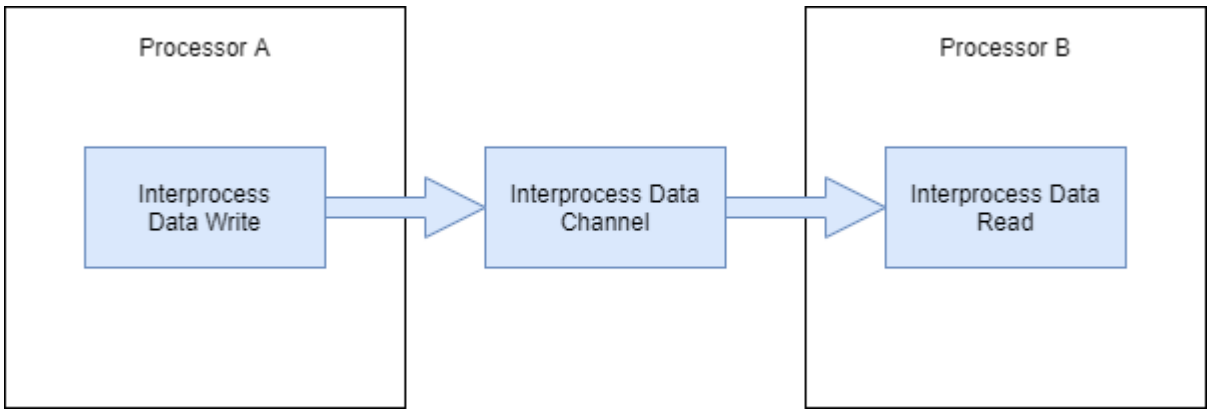
Interprocess Data Read

Receive messages from another processor using interprocess communication channel

Libraries:
SoC Blockset / Processor Interconnect
C2000 Microcontroller Blockset / Target Communication

Description

The Interprocess Data Read block asynchronously receives messages from another processor in an SoC using an interprocess data channel. The Interprocess Data Read block connects to an Interprocess Data Channel block that similarly connects to an Interprocess Data Write block contained in a separate processor reference model. In simulation, data from another processor is asynchronously received and processed in the processor containing the Interprocess Data Read block and associated asynchronous task. This diagram shows a generalized view of the interprocessor data channel connection.



Ports

Input

msg — Data message from interprocess data channel
scalar

This message port receives data messages from the connected Interprocess Data Channel block. The messages process when the Task Manager block triggers the task containing the this block. For more information on messages, see “Messages”.

Dependencies

This port appears only when **Enable status port** parameter is enabled.

Data Types: `SoCData`

Output

data — Data frame read from another processor
vector

This port emits a data frame read from another processor connected via the Interprocess Data Channel block.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `bus`

status — Interprocess data read status
0 | 1 | 2 | 4 | 6

The status port outputs one of these values:

- `0`–No errors
- `1`–Data not available
- `2`–Data type mismatch
- `4`–Data length mismatch
- `6`–Data type and Data length mismatch

Dependencies

This port appears when you disable the **Enable simulation port** parameter.

Parameters

Enable simulation port — Enable peripheral simulation port to block
`on` (default) | `off`

Select this parameter to configure the **msg** output port to enable peripheral simulation capability.

Data type — Data type of interprocess data channel
`double` (default) | `single` | `int8` | `int16` | `int32` | `int64` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `uint64` | `Boolean` | `bus`

Enter the data type used by the interprocess data channel.

You can selectively show or hide the **Data Type Assistant** options by clicking >> button or << button. For more information, refer to “Specify Data Types Using Data Type Assistant”.

Number of buffers — Number of storage buffers
1 (default) | positive integer

Number of buffers making up the storage system.

Dependencies

This parameter is visible only when you disable the **Enable simulation port** parameter.

Buffer size — Size of data vector read from interprocess data channel
1 (default) | positive integer

Enter the size of the data vector read from the interprocess data channel.

Channel number — Select the channel number
`0` (default) | `1` | `2` | `...`

Select the channel where you want to send the data.

Dependencies

This parameter is visible only when you disable the **Enable simulation port** parameter.

Enable interrupt — Enable interrupt to block
`off` (default) | `on`

Select this parameter at Interprocess Data Read and Interprocess Data Write blocks to configure the block interrupts.

Dependencies

This parameter is visible only when you disable the **Enable simulation port** parameter.

Participating cores — Select the participating cores of the processor
`The options vary based on the hardware board you select`

Select the participating cores as per the **Hardware board** you select in the Configuration Parameters window.

Dependencies

This parameter is visible only when you disable the **Enable simulation port** parameter.

Note Ensure that **Channel number**, **Number of buffers** and **Participating cores** parameters match for the corresponding Interprocess Data Read (SoC Blockset) and Interprocess Data Write (SoC Blockset) blocks of the model.

Sample time — Sample time
`-1` (default) | positive scalar

Enter the sample time of the block to apply to the timer-driven task subsystem.

Version History

Introduced in R2020b

See Also

Interprocess Data Write | Interprocess Data Channel

Topics

“Interprocess Data Communication via Dedicated Hardware Peripheral”

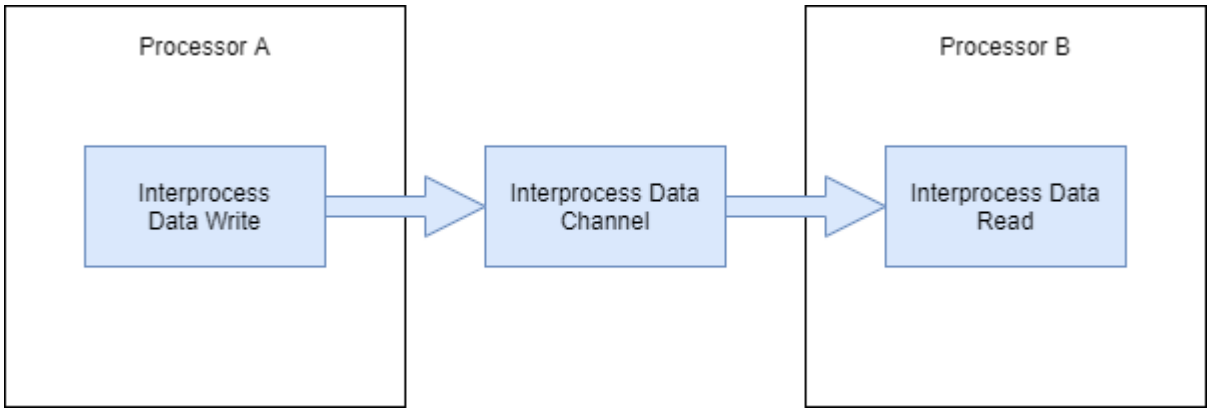
Interprocess Data Write

Send messages to another processor using interprocessor data write

Libraries:
SoC Blockset / Processor Interconnect
C2000 Microcontroller Blockset / Target Communication

Description

The Interprocess Data Write block asynchronously sends messages to another processor in an SoC using an interprocess data channel. The Interprocess Data Write block connects to an Interprocess Data Channel block that similarly connects to an Interprocess Data Read block contained in a separate processor reference model. In simulation, data from the current processor is asynchronously sent and processed in the processor containing the Interprocess Data Read block and associated asynchronous task. This diagram shows a generalized view of the interprocess data channel.



Ports

Input

data — Data input
vector

This port receives a data vector to send to another processor over the interprocess data channel.

Data Types: `single` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Output

msg — Output data message
scalar

This message port sends the output data as a message to the connected Interprocess Data Channel block. For more information on messages, see “Messages”.

Dependencies

This port appears only when **Enable status port** parameter is enabled.

Data Types: `SoCData`

Parameters

Enable simulation port — Enable peripheral simulation port to block
`on` (default) | `off`

Select this parameter to configure the **msg** output port to enable peripheral simulation capability.

Channels number — Select the channel number
`0` (default) | `1` | `2` | `...`

Select the channel where you want to send the data.

Dependencies

This parameter is visible only when you disable the **Enable simulation port** parameter.

Number of buffers — Number of storage buffers
`1` (default) | positive integer

Number of buffers making up the storage system.

Dependencies

This parameter is visible only when you disable the **Enable simulation port** parameter.

Enable interrupt — Enable interrupt to block
`off` (default) | `on`

Select this parameter at Interprocess Data Read and Interprocess Data Write blocks to configure the block interrupts.

Dependencies

This parameter is visible only when you disable the **Enable simulation port** parameter.

Participating cores — Select the participating cores of the processor
`The options vary based on the hardware board you select`

Select the participating cores as per the **Hardware board** you select in the Configuration Parameters window.

Dependencies

This parameter is visible only when you disable the **Enable simulation port** parameter.

Note Ensure that **Channel number**, **Number of buffers** and **Participating cores** parameters match for the corresponding Interprocess Data Read (SoC Blockset) and Interprocess Data Write (SoC Blockset) blocks of the model.

Version History

Introduced in R2020b

See Also

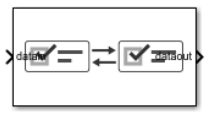
Interprocess Data Read | Interprocess Data Channel

Topics

“Interprocess Data Communication via Dedicated Hardware Peripheral”

Interprocess Data Channel

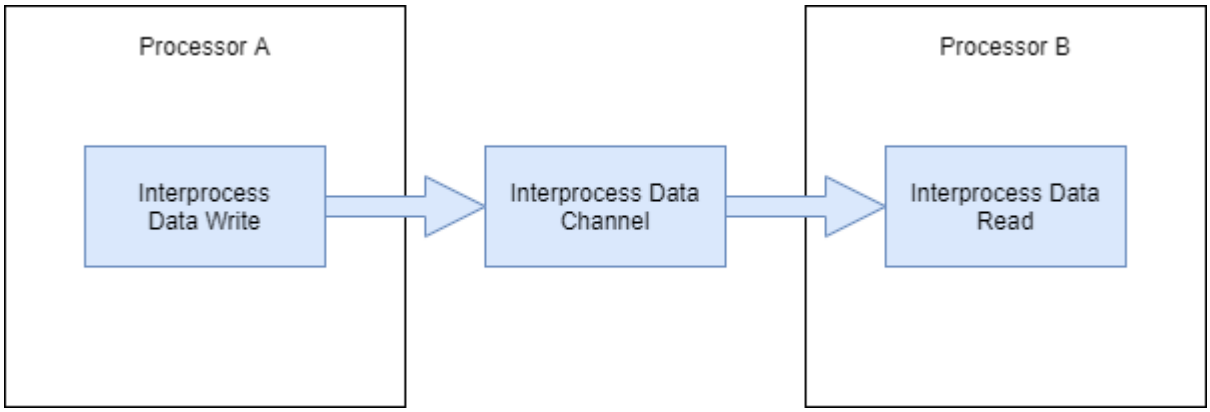
Model interprocessor data channel between two processors



Libraries:
SoC Blockset / Processor Interconnect
C2000 Microcontroller Blockset / Test Bench Blocks

Description

The Interprocess Data Channel block simulates the interprocessor data channel available in multiprocessor or OS managed SoC hardware board families. The block provides a channel for asynchronous data transfer between two processors. This diagram shows a generalized view of the interprocessor data connection.



Limitations

In an SoC model, when Interprocess Data Channel blocks form a closed-loop between two or more tasks, it can create an *artificial algebraic loop* for the Simulink solver. To break the loop, the Simulink solver implicitly adds a delay into the loop. This delay is related to an internal event and cannot be modified by the user, but the delay typically will be on the same order as the base time-step of the model. For more information on artificial algebraic loops in Simulink solvers, see “Artificial Algebraic Loops”.

Ports

Input

datain — Input data message
scalar

This message port receives input data as a message from a connected Interprocess Data Write block. For more information on messages, see “Messages”.

Data Types: `SoCData`

Output

dataout — Output data message
scalar

This message port sends output data as a message to a connected Interprocess Data Read block. For more information on messages, see “Messages”.

Data Types: `SoCData`

overwritten — Output overwrite notification signal
scalar

This port sends a true signal output whenever an overwrite of the internal buffer queue occurs. When the connected processor model executes in external mode, the connected Interprocess Data Write block generates the **overwritten** signal in the **Simulation Data Inspector** tool.

Dependencies

To enable this port, select the **Show when buffer is overwritten** parameter.

Data Types: `Boolean`

used — Output number of buffers in use
scalar

This port outputs the number of buffers currently in use in the block's internal buffer queue. When the connected processor model executes in external mode, the connected Interprocess Data Write block generates the **used** signal in the **Simulation Data Inspector** tool.

Dependencies

To enable this port, select the **Show number of used buffers** parameter.

Data Types: `Boolean`

event — Task event signal
scalar

This port sends a task event signal that triggers the Task Manager block to execute the associated event-driven task.

Note For TI’s C2000™ hardware boards, when the Interprocess Data Channel block connects to the Task Manager block, the allowed interrupts available in the **Hardware Mapping** tool must be in consecutive order starting from `IPC0`. For example:

- If one Interprocess Data Channel block is in the model, then only `IPC0` interrupt is allowed
- If two Interprocess Data Channel blocks are in the model, the only `IPC0` and `IPC1` interrupts are allowed.

Dependencies

To enable this port, select the **Show event port** parameter.

Data Types: `rteEvent`

Parameters

Number of buffers — Number of storage buffers
1 (default) | positive integer

Number of buffers making up the storage system.

Propagation delay — Propagation delay of data through the channel
`1e-6` (default) | non-negative number

Specify the propagation delay of data transfers through the this block. To ignore propagation delays, set this parameter to `0`.

Show event port — Option to enable task event ports
`off` (default) | `on`

Enable an event port that, when connected to the Task Manager block, can execute event-driven tasks.

Show number of used buffers — Option to enable buffer count ports
`off` (default) | `on`

Enable an output port that shows the current number of buffers used in the Interprocess Data Channel block internal buffer queue.

Show when buffer is overwritten — Enable port that shows buffer overwrites
`off` (default) | `on`

Enable an output port that signals when a overwrite of the Interprocess Data Channel block internal buffer queue occurred.

Version History

Introduced in R2020b

See Also

Interprocess Data Read | Interprocess Data Write

Topics

“Multiprocessor Execution” (SoC Blockset)
“Interprocess Data Communication via Dedicated Hardware Peripheral” (SoC Blockset)

Task Manager

Create and manage task executions in Simulink model

Libraries:
SoC Blockset / Processor Task Execution
C2000 Microcontroller Blockset / Scheduling

Description

The Task Manager block simulates the execution of software tasks as they would be expected to behave on an SoC processor. With the Task Manager, you can add and remove tasks from your model that can either be timer-driven or event-driven. Tasks can be represented in a model as rates, for timer-driven tasks, or function-call subsystems, for event-driven tasks, contained inside a single Model block. The Task Manager executes individual tasks based on their parameters, such as period, duration, trigger, priority, or processor core, and the combination of that task with the state of other tasks and their priorities in the running model.

Note The Task Manager block cannot be used in a referenced model. For more information on referenced models, see Model block.

The Task Manager block provides three methods to specify the duration of a task in simulation:

- A probability model of task duration defined in the block mask.
- From a data file recording of either a previous task simulation or from a task on an SoC device.
- Input ports on the block, which you can connect to more dynamic models of task duration.

Limitations

- A model containing a Task Manager blocks does not support simulation stepping. For more information on simulation stepping, see “Debug Simulations in the Simulink Editor”.

Ports

Output

Task1 — Function-call from Task1
scalar

A function-call signal that can trigger timer-driven and event-driven tasks, represented as rate or function-call subsystems in the processor Model block, respectively.

For a rate port from a timer-driven subsystem, to show on the Model block, set the **Block Parameters > Main > Schedule rates** and select ports. For a function-call port from an event-driven subsystem contained in a Function-Call Subsystem block to show on the Model block, include an Inport in the processor Model block connected to the function-call trigger port of the subsystem. In the Inport, check **Block Parameters > Signal Attributes > Output function call**.

Note The `Task1` port must be connected to either a function-call port or scheduled rate signal port on a Model block.

Dependencies

To create or remove a control signal port for a task, add or remove the task from the Task Manager block by clicking the **Add** or **Delete** buttons in the block dialog mask.

Input

***Task1*Event** — Message event notification
scalar

A message port that triggers the associated event-driven task. The `Task1Event` port receives the message from either a Memory Channel block or IO Data Source block. For more information on messages, see “Messages”.

Dependencies

To show a *Task1*`Event` port, then *Task1* must have **Type** set to `Event-driven`.

Data Types: `rteEvent`

***Task1*Dur** — Task duration
positive scalar

A positive value signal that specifies the execution duration of a task at the present time. For more information on specifying task duration, see “Task Duration” (SoC Blockset).

Dependencies

To enable this port, set the **Specify task duration via** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Parameters

Enable task simulation — Enable simulation of task duration
`on` (default) | `off`

Enable or disable the simulation of task duration. If you clear this parameter, tasks simulate using a function-call generator inheriting their period from the fundamental sample time of the model for event-driven tasks or from the dialog for timer-driven tasks.

List of tasks — List of tasks
`Task1` (default)

List of the tasks generated by the Task Manager block. Each task has a set of parameters listed in the **Main** and **Simulation** tabs of the block dialog mask.

Add — Add task
button

Add a task to the Task Manager block. During deployment, each task is encapsulated as an execution thread in the generated code. The properties of the thread are taken from the **Main** parameters for

that task. During simulation, the task uses a combination of the **Main** and **Simulation** parameters for that task.

Delete — Delete existing task
button

Remove a task from the Task Manager.

Dependencies

To enable this parameter, specify at least two tasks.

Use Schedule Editor ordering — Specify task priority using Schedule Editor tool
`off` (default) | `on`

Use the **Schedule Editor** to specify the ordering of the tasks in the SoC model. When using the **Schedule Editor**, task priority is automatically assigned to the tasks based on their order in the editor and the base rate priority of the processor model. For more information on using **Schedule Editor** to specify task priority, see “Task Management with Schedule Editor” (SoC Blockset).

Main

Name — Name of task
`Task1` (default) | character vector

Unique name of the task. The task name must only contain alphanumeric characters and underscores.

Type — Trigger type of task
`Timer-driven` (default) | `Event-driven`

Specify the task as timer-driven or event-driven. For more information on timer- and event-driven tasks, see “Timer-Driven Task” (SoC Blockset) and “Event-Driven Tasks” (SoC Blockset), respectively.

Dependencies

To enable this parameter, set `Type` to `Timer-driven`.

Period — Timer period
`0.1` (default) | positive scalar

Specify the trigger time period for timer-driven tasks.

Core — Processor core to execute task
`0` (default) | non-negative integer

Specify the number of the processor core where a task executes. For more information on selecting cores and core execution visualizations, see “Multicore Execution and Core Visualization” (SoC Blockset).

Priority — Priority of task in scheduler
`10` (default) | positive integer

Specify the schedulers priority for the event-driven task between 1 and 99. Higher priority tasks can preempt lower priority tasks, and vice versa. The task priority range is limited by the hardware attributes. For more information on task priority, see “Task Priority and Preemption” (SoC Blockset).

Dependencies

To enable this parameter, set `Type` to `Event-driven` and `Use Schedule Editor ordering` to `off`.

Drop tasks that overrun — Drop tasks that overrun
`off` (default) | `on`

Select this parameter to force tasks to drop, rather than catch up, following an overrun instance. For more information on task overruns, see "Task Overruns and Countermeasures" (SoC Blockset).

Note No more than 2 instances of a task can overrun execution when `Drop tasks that overrun` is set to `off`. Any additional task instances that overrun drop automatically.

Simulation

Play recorded task execution sequence — Enable playback from file
off (default) | on

Select this parameter for the Task Manager block to play back the recorded execution data provided from the specified **File name** parameter. For more information on replaying task execution, see "Task Execution Playback Using Recorded Data" (SoC Blockset).

Specify task duration via — Source of task execution time
`Dialog` (default) | `Input port` | `Record task execution statistics`

Specify the source of the timing information for the task execution.

- `Dialog` - Use a normally distributed probabilistic model with **Mean**, **Deviation**, **Min**, and **Max** defined in the block dialog mask.
- `Input port` – When set from *Input port*, the block input port dynamically defines the execution duration.
- `Record task execution statistics` – Use a normally distributed probabilistic model with mean and deviation provided in file specified by **File name**.

For more information on configuring task duration, see "Task Duration" (SoC Blockset).

Task duration settings

Add — Adds distribution
button

Adds a distribution to the set of normal distributions that generates an execution duration. For more information on configuring task duration, see "Task Duration" (SoC Blockset).

Note Only a maximum five distributions can be assigned to a single task.

Delete — Remove distribution
button

Remove a distribution from the set of normal distributions.

Percent — Likelihood of distribution
`100` (default) | positive scalar

Specify the likelihood of each normal distribution. The **Percent** weighted sum of normal distributions determines the task duration likelihood. For more information on configuring task duration, see “Task Duration” (SoC Blockset).

Note The sum of **Percent** for all the distributions in a single task must equal `100`.

Mean — Mean task duration in simulation
`1e-06` (default) | positive scalar

Specify the mean duration of the task during simulation of the task. The simulated task duration uses a normal distribution with a specified **Mean** and **SD** parameter values as a first-order approximation of the task behavior. For more information on configuring task duration, see “Task Duration” (SoC Blockset).

SD — Standard deviation of task duration in simulation
`0` (default) | positive scalar

Specify the standard deviation duration of the task during simulation of the task. The simulated task duration uses a normal distribution with a specified **Mean** and **SD** as a first-order approximation of the task behavior. For more information on configuring task duration, see “Task Duration” (SoC Blockset).

Min — Lower limit of task duration
`1e-06` (default) | positive scalar

Lower limit of a task duration distribution. For more information on configuring task duration, see “Task Duration” (SoC Blockset).

Max — Upper limit of task duration
`1e-06` (default) | positive scalar

Upper limit of a task duration distribution. For more information on configuring task duration, see “Task Duration” (SoC Blockset).

File name — File containing diagnostic scheduling data
filepath

The data in this file specifies the **Mean** and **SD** parameter values. When the **Play recorded task execution sequence** parameter is selected, the specified CSV file provides the explicit task execution timing. The CSV file contains the diagnostic data of the task scheduler previously recorded from the hardware board. For more information on configuring task duration, see “Task Duration” (SoC Blockset).

Dependencies

To enable this parameter, set the **Specify task duration via** parameter to `Recorded task execution statistics`.

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

To automatically generate C code for your design, and execute on an SoC device, use the **SoC Builder** tool. To generate and execute C code for your SoC models, Embedded Coder features are required. For more information on generating code for SoC designs, see “Generate SoC Design” (SoC Blockset).

The tasks in the Task Manager block execute as threads in the generated code. The task parameters in the Task Manager block specify the priority and execution core of the thread.

See Also

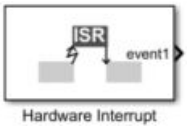
IO Data Source | Memory Channel

Topics

“Get Started with Multiprocessor Blocks on MCUs” (SoC Blockset)
“What is Task Execution?” (SoC Blockset)
“Task Duration” (SoC Blockset)

Hardware Interrupt

Trigger downstream function-call subsystems from interrupt service routine



Libraries:
C2000 Microcontroller Blockset/ Scheduling

Description

Use the Hardware Interrupt block to create an interrupt service routine (ISR) automatically in the generated code of your model. The ISR executes the downstream function-call subsystem associated with event ports of the block.

The function call subsystem associated with the event output port run at same priority as that of ISR priority.

Using this block you can:

- Create ISRs.
- Set ISR priority.
- Enable or disable interrupt preemption.
- Use the **Hardware Mapping** tool to configure the desired event or interrupt.

The **Hardware Mapping** tool allows you to configure the hardware interrupts tasks for the selected hardware board. With this tool, you can map the tasks in your software model to the available event sources and interrupts:

- Manually select the task in **Mapping Browser** > **Tasks** > **`CPU name`**. Select the desired event or interrupt source. For more, see “Configure Interrupts and Events Using Hardware Mapping”

Ports

Input

***name* Event Task** — Function-call event input simulation
scalar

The simulation-only message input port, when connected to an Function-Call Generator block, the block acts as pass-through with the output emitted on the ***name* Event** port in simulation.

Dependencies

To enable this port, select the **Enable simulation port** parameter.

Output

***name* Event** — Generate interrupt request
`Scalar`

The output of this block is a function-call. The number of function call outputs will be same as events selected to serve in an ISR.

Parameters

Number of events to serve — Specify number of events to serve
1 (default) | 2 | 3 | <32

Specify the number of events to serve for the Hardware Interrupt bock. This parameter enables the specified number events as output port `event#`

Simulink task priority — Set priority of selected ISR
`50` (default) | `positive integer or nonnegative integer`

The value you specify in this parameter sets the priority of the downstream function-call subsystem. The simulink task priority of the selected (ISR) is relative to the model base rate priority.

Note The default model base sample rate priority is set to 40 with a lower priority value indicating a higher priority task. To achieve this the **Higher priority value indicates higher task priority** option must be enabled in **Configuration Parameters > Solver** pane.

Disable interrupt pre-emption — Select to disable interrupt preemption
`off` (default) | `on`

By default, an interrupt can be preempted by a higher priority interrupt. Selecting this option allows low priority interrupts to complete their execution without being preempted by other interrupts.

Enable simulation port — Enable simulation ports to block
`off` (default) | `on`

Select this parameter to add an compatible simulation input port.

Version History

Introduced in R2023a

See Also

Hardware Mapping | “Configure Interrupts and Events Using Hardware Mapping”

ADC Interface

Convert analog signal on ADC input pin to digital signal

Description

The ADC Interface block simulates the analog-to-digital conversion (ADC) of a hardware board. The input analog signal gets sampled and converted into a representative digital value. A start event message signals the block to sample the input analog voltage signal. When the conversion completes, the block emits the digital representation of the analog signal and sends an event to a Task Manager block. At this point, a connected task can execute with the new ADC sample.

Ports

Input

start — Start analog to digital conversion
start an analog to digital conversion event

Specify an event signal to start the sampling and measurement of the **analog** input port signal.

Data Types: `rteEvent`

analog — Analog voltage signal
scalar

Input analog voltage signal to convert into a digital measurement.

Data Types: `double` | `single`

Output

digital — SoC message data
scalar

This port sends the ADC Interface input signal data as a message to the **msg** input port of the ADC Read block.

Data Types: `SoCData`

wd event — Analog watchdog task event signal
scalar

This port sends a message at whenever the analog voltage signal exceeds the specified **Lower threshold** and **Upper threshold** property values. This output connects to the input of the Task Manager block to execute the associated event-driven task to react to the over- or under-voltage input event.

Dependencies

To enable this port, enable the **Enable analog watchdog** parameter.

Data Types: `rteEvent`

event — Task event signal
scalar

This port sends a message at each analog to digital signal conversion event. This output connects to the input of the Task Manager block to execute the associated event-driven task after executing the ADC event.

Dependencies

To enable this port, enable the **Enable interrupt** parameter.

Data Types: `rteEvent`

Parameters

Single Channel

Resolution (bits) — Resolution of digital measurement
12 (default) | 16

An input analog signal can be represented in digital values in the form of 12 or 16 bits. The minimum value of an analog signal that can be represented in 1 bit is called resolution. One bit represents the minimum voltage resolution measurable by the ADC. The minimum voltage resolution can be determined using the following equation:

where *n* is the **Resolution (bits)** and `Vref` is the **Voltage reference (V)** parameter values.

Example: `16`

Voltage reference (V) — Reference voltage in ADC
3 (default) | `3.3`

The reference voltage determines the total voltage range that the ADC can convert into a digital value without saturating. Any voltage signal higher than this value produces the maximum possible value that can represented by the **Resolution (bits)** parameter.

Example: `3.3`

Acquisition time (s) — Time required for ADC to capture input voltage
`320e-9` (default) | positive scalar

Specify the time required for the ADC to capture the input voltage during sampling.

Example: `200e-9`

Conversion time (s) — Time to convert physical voltage sample to digital value
`240e-9` (default) | positive scalar

Specify the required time to convert the physical voltage sample to the digital representation and output the value.

Example: `20e-9`

Charge/dischard time constant (s) — Charge or discharge time constant of the ADC acquisition circuit
`0` (default) | nonnegative scalar

Specify the charge or discharge time constant of the ADC sample acquisition circuitry.

Multichannel

Number of channel — Number of channels used in multichannel sampling
1 (default) | integer in the range 1 to 16

Specify the number of channels used by the ADC module. Specifying 2 or more channels allows for either more efficient or precise measurements of the input signal.

Conversion type — Type multichannel conversion
`Sequential` (default) | `Simultaneous` | `Oversampling`

Select the type of multichannel conversion.

- `Sequential` — Take sequential measurements on each ADC channel. At a new ADC event, the next channel in the sequence of channels takes a new measurement of the input signal. All other previous channel values remain unchanged. Sequential measurement improves sampling by allowing for individual conversion times of each channel to exceed the sample rate of the ADC module.
- `Simultaneous` — Take simultaneous measurements on each ADC channel. At a new ADC event, all channels take a new measurement of the input signal, replacing the previously captured value. Simultaneous measurement allows for noise to be removed from the measurement using an average value or other filter.
- `Oversampling` — Take oversampled measurements across the channels of the ADC. Between two timer-driven ADC events, each channel takes a time offset ADC measurement, resulting in the channels sampling the input signal evenly between the two ADC events. The resulting channel output provides an oversampled measurement of the input signal at each sample. Oversampling measurement allows for the ADC module to exceed the theoretical Nyquist sample rate of the individual channel and ADC hardware.

Event

Enable interrupt — Option to enable interrupt event generation
enable (default) | disable

Select this parameter for the ADC Interface block to generate an interrupt following an ADC acquisition and to enable the **event** output port. You can connect this **event** port to a Task Manager block to simulate asynchronous ADC operation.

Condition — Condition on when to trigger interrupt
`Acquisition time` (default) | `Acquisition + Conversion time`

Select the timing condition for when to generate the ADC interrupt event. Using `Acquisition + Conversion time`, the interrupt is generated when the complete measurement is available. Using `Acquisition time`, the interrupt is generated prior to the measurements availability. Allowing for the associated task to start during the conversion and reduce execution delay in the total measurement cycle.

Enable analog watchdog — Option to enable analog watchdog interrupt event generation
off (default) | on

Select this parameter for the ADC Interface block to generate an analog watchdog interrupt following an ADC acquisition where the input voltage exceeds the specified **Lower threshold** and **Upper**

threshold parameter values. Selecting this parameter also enables the **wd event** output port, which you can connect to a Task Manager block to simulate task action on an over- or under-voltage event on the ADC input signal.

Lower threshold — Lower threshold watchdog trigger
`0.1` (default) | real-valued scalar

Specify the lower threshold value of the analog input signal on which to trigger an analog watchdog interrupt event.

Example: `0.2`

Upper threshold — Upper threshold watchdog trigger
`2.9` (default) | real-valued scalar

Specify the upper threshold value of the analog input signal on which to trigger an analog watchdog interrupt event.

Example: `3.0`

Interrupt latency (s) — Interrupt generation latency
`0` (default) | positive scalar

Specify the time required by the ADC hardware module from the completion of the conversion to the generation of the interrupt in software.

Example: `0.00001`

Version History

Introduced in R2020b

See Also

ADC Read | PWM Write | PWM Interface

Topics

“Get Started with Multiprocessor Blocks on MCUs” (SoC Blockset)
“Integrate MCU Scheduling and Peripherals in Motor Control Application” (SoC Blockset)

External Websites

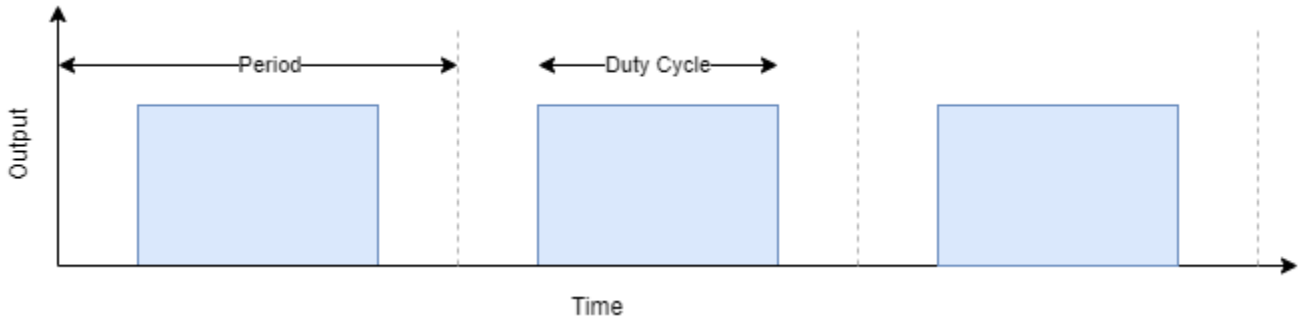
https://en.wikipedia.org/wiki/Analog-to-digital_converter

PWM Interface

Simulate pulse width modulation (PWM) output from hardware

Description

The PWM Interface block simulates the PWM output of a hardware board. This blocks gets duty cycle data messages from a connected PWM Write block that can either generate a switching pulse-width-modulated waveform or pass the duty cycle value to the output.



Ports

Input

msg — SoC message data
numeric vector

This port receives the duty cycle data from the **msg** port of a connected PWM Write block.

Data Types: `SoCData`

Output

PWM — Pulse-width-modulated signal
scalar

This port outputs the pulse-width-modulated rectangular wave defined by the **dCycle** input port.

Dependencies

To enable this port, set the **Output mode** parameter to `Switching`.

Data Types: `double`

~PWM — Complimentary pulse-width-modulated signal
scalar

This port outputs the complimentary **PWM** signal.

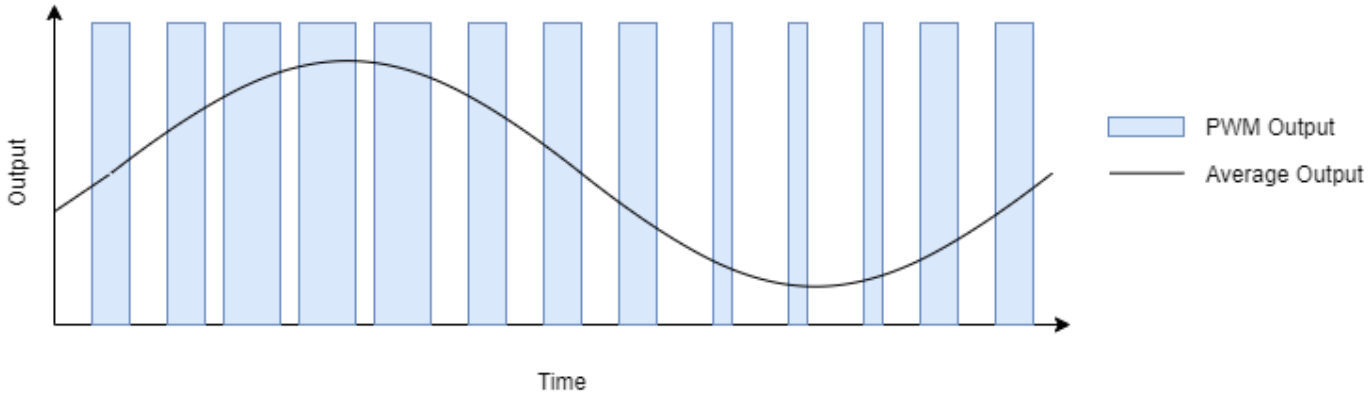
Dependencies

To enable this port, set the **Output mode** parameter to `Switching`.

Data Types: `double`

dCycle — Analog approximation of pulse-width-modulated signal
scalar

This port emits the averaged value of the PWM waveform, which is a pass-through of the duty cycle input value. This image shows the average output signal equivalent to the PWM output.



Dependencies

To enable this port, set the **Output mode** parameter to `Average`.

Data Types: `double`

event — Event emitted on each PWM cycle
scalar

This port sends a message during each PWM output event that can connect to the **start** port of the ADC Interface block to synchronize ADC and PWM events in closed-loop systems.

Dependencies

To enable this port, the **Type** parameter must be set to `ADC start` or `ADC start and PWM interrupt`.

Data Types: `rteEvent`

event*n* — Replicate event emitted on each PWM cycle
scalar

This port creates a replica port of the **event** output port to coordinate multiple ADC modules with the PWM module.

Dependencies

To enable this port, set the **Type** parameter to `ADC start` or `ADC start and PWM interrupt` and the **Number of replicas** parameter to a value greater than or equal to 2.

Data Types: `rteEvent`

interrupt — Interrupt event emitted on each PWM cycle
scalar

This port sends a message during each PWM output event that can connect to the Task Manager block to trigger other tasks in response to the PWM output update.

Dependencies

To enable this parameter, set the **Type** parameter to `PWM interrupt` or `ADC start and PWM interrupt`.

Data Types: `rteEvent`

Parameters

Main

PWM waveform period (s) — Period of PWM waveform
`50e-6` (default) | positive scalar

Specify the period of the PWM waveform in seconds.

Note For `PWM waveform period (s)` of 10ns, the duty cycle must be greater than 1%.

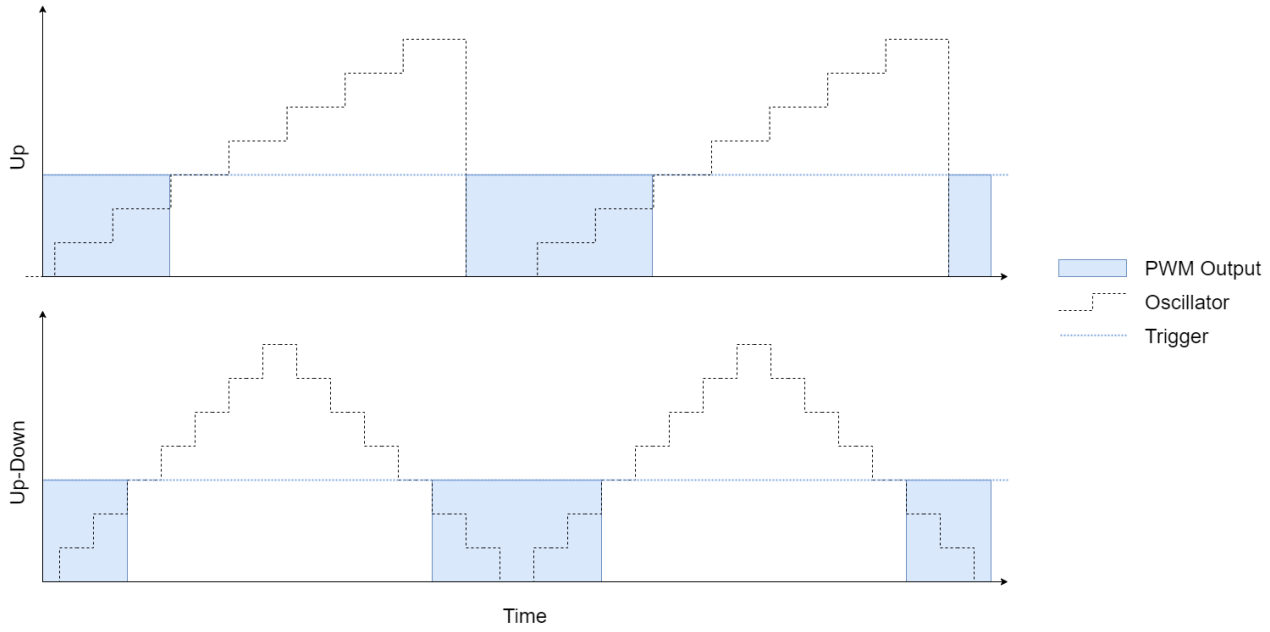
Output mode — Output mode
`Switching` (default) | `Average`

Simulate the output signal as either a true PWM waveform by specifying `Switching` or as the average of the duty cycle by specifying `Average`.

Example: `Switching`

Counter mode — Counter waveform
`Up-Down` (default) | `Up` | `Down`

The counter mode specifies the shape of the underlying sawtooth waveform that drives the PWM output signal inside the PWM module. In `Up` mode, the sawtooth counter increments to the maximum and then resets to zero on each period. In `Down` mode, the sawtooth counter decrements to zero then resets to the maximum. In `Up-Down` mode, the sawtooth counter oscillates from zero to the maximum value.



Example: `Up`

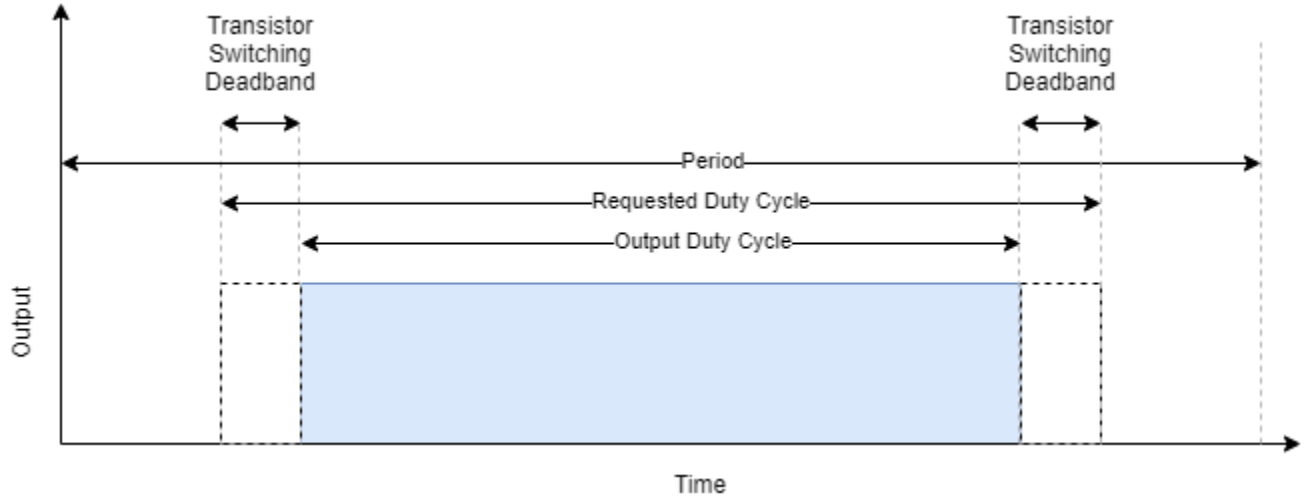
Sampling mode — Sampling mode
`End of PWM period` (default) | `Mid or End of PWM period` | `Immediate (at compare matches)`

Specify the time at which the input duty cycle is sampled.

Example: `Mid or End of PWM period`

Dead time (s) — Dead band switching delay
`1e-6` (default) | positive scalar

A time delay is introduced between turning off one of the transistors of a leg of an inverter and turning on the other transistor to ensure that a dead short circuit does not occur. This diagram shows the expected duty cycle and the delay introduced by the transistor switching the dead band.



Example: `450e-9`

PWM Output

At *`position`* of period — Signal change at position in period
`High` | `Low` | `Change` | `NoChange`

Specify the state of the PWM waveform signal at the *`position`* in the waveform relative to the total period. When set to `High` or `Low`, the waveform output changes to `1` or `0`, respectively. When set to `Change`, the waveform inverts the current value. When set to `NoChange`, the waveform does not change. The *`position`* can either be the `start` or `mid` point of the PWM waveform. This table gives the default settings for these parameters.

Parameter	Default
At start of period	`High`
At mid of period	`NoChange`

Dependencies

`At mid of period` is only available when the **Counter mode** parameter is set to Up-Down.

At compare *`n`* — Signal change at comparator n trigger
`High` | `Low` | `Change` | `NoChange`

Specify the state of the PWM waveform signal when the internal PWM counter triggers comparator *`n`*. When set to `High` or `Low`, the waveform output changes to `1` or `0`, respectively. When set to `Change`, the waveform inverts the current value. When set to `NoChange`, the waveform does not change. Two comparators, `1` and `2`, are available to modify the PWM signal. This table gives the default settings for these parameters.

Parameter	Default
At compare 1	`Low`
At compare 2	`NoChange`

Dependencies

At compare 1 and **At compare 2** parameters are only available when the **Counter mode** parameter is set to `Up` or `Down`.

At compare *n* *direction* count — Signal change at comparator n trigger
`High` | `Low` | `Change` | `NoChange`

Specify the state of the PWM waveform signal when the internal PWM counter crosses the comparator *n* value in the specified *direction*. When set to `High` or `Low`, the waveform output changes to `1` or `0`, respectively. When set to `Change`, the waveform inverts the current value. When set to `NoChange`, the waveform does not change. Two comparators, `1` and `2`, are available to modify the PWM signal. This table gives the default settings for these parameters.

Parameter	Default
At compare 1 up count	`Low`
At compare 1 down count	`High`
At compare 2 up count	`NoChange`
At compare 2 down count	`NoChange`

Dependencies

These arguments are only available when the **Counter mode** parameter is set to `Up-Down`.

Phase

Phase offset in degree (0-360) — PWM waveform offset
scalar from `0` to `360`

Specify the phase of the PWM waveform relative period of waveform. The phase is represented as a scalar between `0` to `360` degrees.

Event

Type — Type of events to generate
`ADC start` (default) | `PWM interrupt` | `ADC start and PWM interrupt`

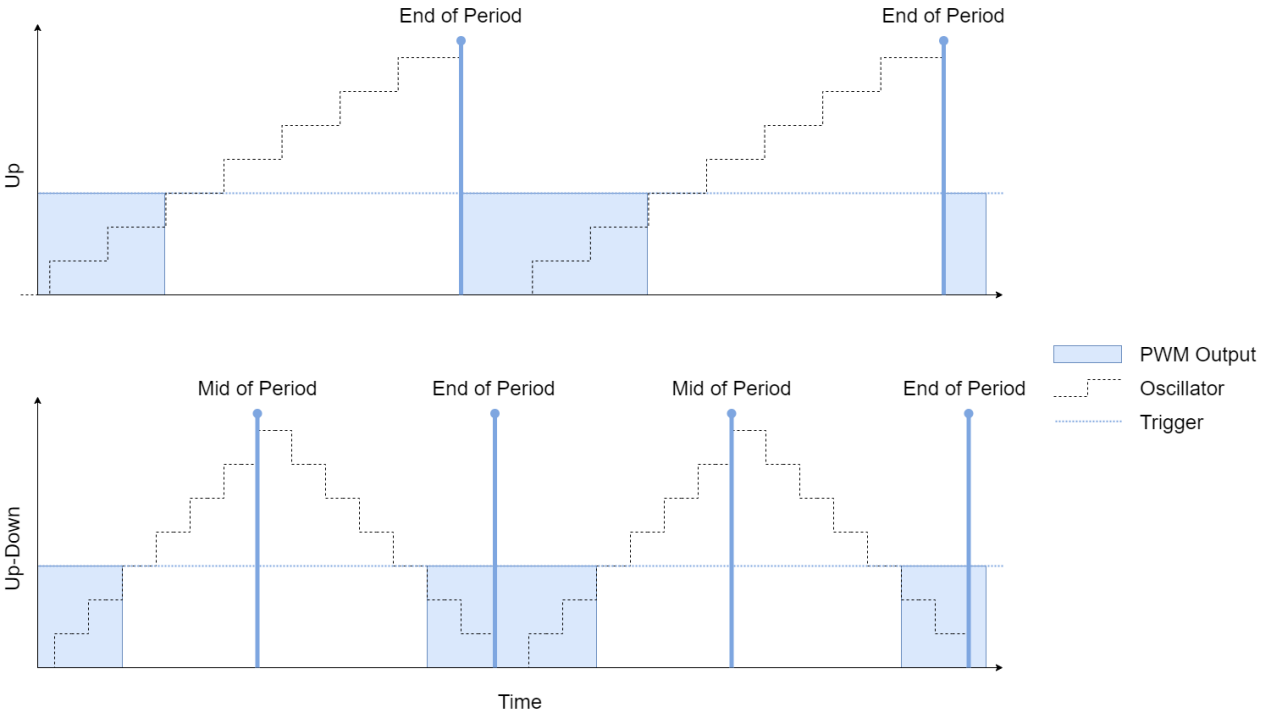
Specify the types of events on which to generate events. When the **Type**value is set to:

- `ADC start` — Generate an event to trigger the start ADC conversion.
- `PWM interrupt` — Generate an interrupt event to trigger the start of a task.
- `ADC start and PWM interrupt` — Generate events for both ADCs and tasks.

Example: `ADC start and PWM interrupt`

ADC start condition — Trigger mode relative to PWM waveform
`End of PWM period` (default) | `Mid of PWM period` | `Mid or End of PWM period` | `Compare 1 up count` | `Compare 1 down count` | `Compare 2 up count` | `Compare 2 down count`

Specify when this block triggers an event relative to the PWM waveform.



Example: `Mid or End of PWM period`

Dependencies

To enable this parameter, the **Type** parameter must be set to `ADC start` or `ADC start and PWM interrupt`.

Generate on — Generate event on multiple of PWM update
`1st event` (default) | *n*`th event` | `16th event`

Specify to generate and output an ADC trigger event on the specified multiple of the PWM vent. For example, if **Generate on** is set to the `6th event`, the PWM Interface block receives 6 messages updates the output 6 times before generating an ADC event message.

Example: `4th event`

Dependencies

To enable this parameter, the **Type** parameter must be set to `ADC start` or `ADC start and PWM interrupt`.

Number of replicas — Generate replica event ports
1 (default) | integer from 1 to 16

Generate replica ADC event ports and events on the block. Use this coordinate the triggering of multiple ADCs modules from a single PWM Interface block.

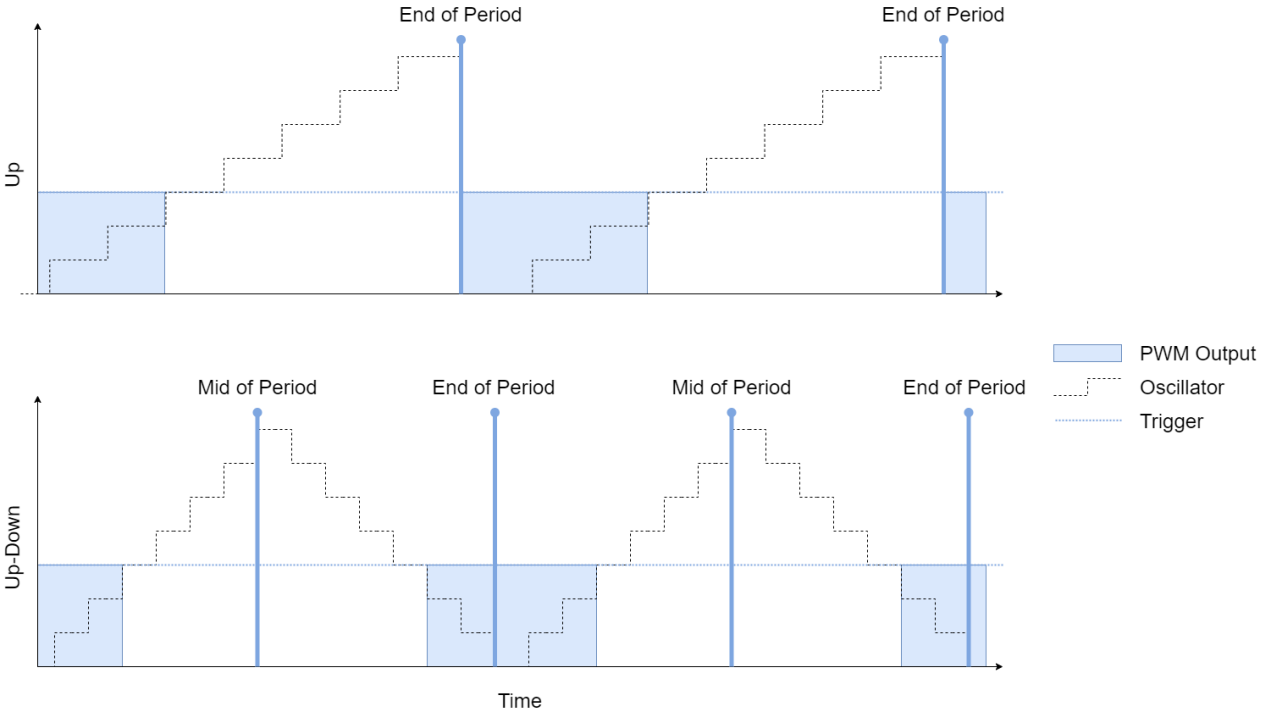
Example: 4

Dependencies

To enable this parameter, the **Type** parameter must be set to `ADC start` or `ADC start and PWM interrupt`.

PWM interrupt condition — Trigger mode relative to PWM waveform
`End of PWM period` (default) | `Mid of PWM period` | `Mid or End of PWM period` | `Compare 1 up count` | `Compare 1 down count` | `Compare 2 up count` | `Compare 2 down count`

Specify when this block triggers an interrupt event relative to the PWM waveform.



Example: `Mid or End of PWM period`

Dependencies

To enable this parameter, the **Type** parameter must be set to `PWM interrupt` or `ADC start and PWM interrupt`.

Interrupt latency (s) — Interrupt generation latency
`0` (default) | positive number

Specify the time required by the PWM hardware module from the completion of the output update to the generation of the interrupt in software.

Example: `0.00001`

Dependencies

To enable this parameter, the **Type** parameter must be set to `PWM interrupt` or `ADC start and PWM interrupt`.

Generate on — Generate event on multiple of PWM update
`1st event` (default) | *n*`th event` | `16th event`

Specify to generate and output a PWM interrupt trigger event on the specified multiple of the PWM vent. For example, if **Generate on** is set to the `6th event`, the PWM Interface block receives 6 messages updates the output 6 times before generating an PWM interrupt event message.

Example: `4th event`

Dependencies

To enable this parameter, the **Type** parameter must be set to `PWM interrupt` or `ADC start and PWM interrupt`.

Version History

Introduced in R2020b

See Also

PWM Write | ADC Interface

Topics

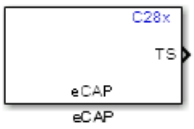
“Get Started with Multiprocessor Blocks on MCUs” (SoC Blockset)
“Integrate MCU Scheduling and Peripherals in Motor Control Application” (SoC Blockset)

External Websites

https://en.wikipedia.org/wiki/Pulse-width_modulation

C28x eCAP

Receive and log transitions on capture input pin or configure auxiliary pulse width modulator



Libraries:
C2000 Microcontroller Blockset / C2802x
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / C2834x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x
C2000 Microcontroller Blockset / F28M35x / C28x
C2000 Microcontroller Blockset / F28M36x / C28x

Description

The eCAP block captures the timing of important external events, such as Hall sensor signals in speed measurements of rotating machinery. When not used in capture mode, the block can be used in APWM mode, which is a single-channel, asymmetric pulse width modulator (APWM). You can add one eCAP block to your model for each capture pin. You cannot assign the same eCAP pin to two eCAP blocks in a model. eCAP and APWM modes use the same pins. In eCAP mode, the pins are used as input to capture the transitions. In APWM mode, the pins are used to output a PWM waveform.

Input/Output Ports

Input

SI — Synchronization input from software
scalar

The input from the software used to synchronize the eCAP counter. The synchronization occurs when the synchronization input value is 1.

Dependencies

The port appears only when:

- On the **General** tab, you select **Operating mode > eCAP** or **APWM**.
- On the **General** tab, you select **Enable counter Sync-In mode** and **Enable software-forced counter synchronizing input**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

RA — One-Shot capture sequence
scalar

Starts a One-Shot capture sequence.

A 2-bit stop register is used to compare the Mod4 counter output, and when the register and counter values are equal the Mod4 counter is stopped.

Dependencies

The port appears only when:

- On the **General** tab, you select **Operating mode > eCAP**.
- On the **eCAP** tab, you set **Select mode control > One-Shot** and select **Enable One-Shot re-arming control input**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

T — APWM period
scalar

Period of the APWM.

Dependencies

The port appears only when:

- On the **General** tab, you select **Operating mode > APWM**.
- On the **APWM** tab, you select **Waveform period source > Input port**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

W — APWM width
scalar

Width of the APWM.

Dependencies

The port appears only when:

- On the **General** tab, you select **Operating mode > APWM**.
- On the **APWM** tab, you select **Duty cycle source > Input port**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

Output

The output ports appear only in eCAP mode.

TS — Output timestamps of capture events
vector

TS is a vector of 4 signal dimension corresponding to CAP1, CAP2, CAP3 and CAP4 time stamp values depending on the capture event selected in **Stop value after** on the **eCAP** tab. Use the **Enable reset counter after capture event # time-stamp** option to reset the counter after an event. This option is useful for finding the time difference between the events.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

CF — Status of capture event
vector

CF is a vector of dimension 4, corresponds to CEVT1-CEVT2. The status of the capture event. `0` indicates that no event has occurred. `1` indicates that the event specified by the **Stop value after** parameter has occurred at the eCAP pin.

Dependencies

The port appears only when you select **Enable capture event status flag output** on the **eCAP** tab.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

OF — Status of overflow
scalar

The status of overflow. `0` indicates that counter has not overflowed. `1` indicates that the counter has overflowed from the highest value to 0.

Dependencies

The port appears only when you select **Enable overflow status flag output** on the **eCAP** tab.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

Parameters

General

Operating mode — Select eCAP or APWM mode
`eCAP` (default) | `APWM`

When you select `eCAP`, the block captures and logs pin transitions for each capture unit to a FIFO buffer. When you select `APWM`, the block generates asymmetric pulse width modulation (APWM) waveforms for driving downstream systems.

eCAPx pin — Select capture unit pin
`eCAP1` (default) | `eCAP2` | `eCAP3` | `eCAP4` | `eCAP5` | `eCAP6` | `eCAP7`

Select the required eCAP module to have a dedicated eCAP pin for capturing the external events.

The pin selection for the eCAP module can be done by browsing to **Hardware Implementation** > **Target hardware resources**. The selection option is provided only if the module has more than one pin that can be configured for an eCAP module.

Counter phase offset value (0 ~ 4294967295) — Time base for event captures
`0` (default) | integer in [0 4294967295]

This value provides the time base for event captures, clocked by the system clock. A phase register is used to synchronize with other counters via software- or hardware-forced synchronization. For information about software- or hardware-forced synchronization, see the **Enable counter Sync-In mode** parameter. This value is useful in APWM mode when you need a phase offset between capture modules. Set the phase offset to an integer from 0 to 42949667295 (2^{32}) counts.

Enable counter Sync-In mode — Enable TSCTR counter to load from CTRPHS register
`off` (default) | `on`

Synchronization can be done using the SYNCI event or the software. When synchronization occurs, the shadow register CTRPHS is loaded into the active counter TSCTR in the current eCAP module and the eCAP modules downstream.

Enable software-forced counter synchronizing input — Synchronize one or more eCAP time bases
`off` (default) | `on`

A software method for synchronizing one or more eCAP time bases. The synchronization occurs when the synchronization input value is 1.

Dependencies

This parameter appears only when **Enable counter Sync-In mode** is selected.

Sync output selection — Synchronize eCAP counter with other eCAP counters
`Disabled` (default) | `Pass through` | `CTR=PRD`

Synchronizes an eCAP counter with other eCAP counters. The options are:

- `CTR=PRD` — Triggers the sync-out signal when the counter value equals the period.
- `Pass through` — The sync-in event is passed through as the sync-out signal.

  Note

  - Pass through option in case of F2838x/002x and 003x processor is only applicable for `SWSYNC input`. `ECAPSYNCIN` signals cannot be passed through and must be configured by the downstream eCAP modules under **Hardware Implementation** > **Target hardware resources** > **eCAP**.
  - Sync output selection of `CTR=PRD` is only applicable for eCAP when **Operating mode** is set to `APWM`.

- `Disabled` — Disables the sync-out signal.

Sample time — Frequency at which block reads input pin value
`0.001` (default)

Sample time for the block in seconds.

eCAP

To enable configuration parameters on the eCAP tab, set **Operating mode** to `eCAP` on the **General** tab.

Event prescaler (integer from 0 to 31) — Prescales input signal in multiples of 2
`0` (default) | scalar integer in [0 31]

The input signal is prescaled by twice the value of this parameter. For example, if you enter `1`, the input is prescaled by 2, and for `31`, the input is prescaled by 62. Entering `0` bypasses the input prescaler, leaving the input capture signal unchanged.

Select mode control — Mode of capture
`Continuous` (default) | `One-Shot`

The `Continuous` option performs continuous timestamp captures (events 1 through 4) using a circular buffer.

The `One-Shot` option enables the **Enable One-Shot rearming control via input port** option.

Enable One-Shot rearming control via input port — Re-arms a One-Shot capture sequence
`off` (default) | `on`

When this parameter is selected, a One-Shot capture sequence is re-armed as follows:

1. Mod4 counter is reset to zero.
2. Mod4 counter is unfrozen.
3. Capture register loading is enabled.

Dependencies

This parameter appears only when you select `One-Shot` for **Select mode control**.

Stop value after — Number of capture events after which capture stops
`Capture Event 1` (default) | `Capture Event 2` | `Capture Event 3` | `Capture Event 4`

The number of capture events after which you want to stop the capture sequence.

Enable reset counter after capture event # time-stamp — Resets counter after capture event
`off` (default) | `on`

The eCAP process resets the counter after receiving a capture event timestamp. In this case, **#** represents the number of the capture event set in the **Stop value after** parameter.

Select capture event # polarity — Start capture event on rising edge or falling edge
`Rising Edge` (default) | `Falling Edge`

The option that starts a capture event. In this case, **#** represents the number of the capture event set in the **Stop value after** parameter.

Time-Stamp counter data type — Data type of counter
`uint32` (default) | `double` | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `boolean`

The data type of the timestamp counter.

Enable capture event status flag output — Output capture event status
`off` (default) | `on`

Outputs the capture event status flag at the output port **CF**. The block outputs `0` until the event is captured. After the event, the flag value is `1`.

Capture flag data type — Data type of output port CF
`uint32` (default) | `double` | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `boolean`

The data type of the output port **CF**.

Dependencies

This parameter appears only when you select **Enable capture event status flag output**.

Enable overflow status flag output — Output status of elements of FIFO buffer
`off` (default) | `on`

Outputs the status of the elements of the FIFO buffer at the output port **OF**.

Overflow flag data type — Data type of output port OF
`uint32` (default) | `double` | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `boolean`

The data type of the output port **OF**.

Dependencies

This parameter appears only when you select **Enable overflow status flag output**.

APWM

To enable configuration parameters on the APWM tab, set **Operating mode** to `APWM` in the **General** tab.

Waveform period units — Units for measuring waveform period
`Seconds` (default) | `Clock cycles`

`Clock cycles` uses the high-speed peripheral clock cycles of the processor.

Waveform period source — Source from which waveform period value is obtained
`Specify via dialog` (default) | `Input port`

Select `Specify via dialog` to enter the value in **Waveform period**, or select `Input port` to use a value from the **T** input port.

Waveform period — Period of PWM waveform
`0.001` (default)

Period of the PWM waveform measured in clock cycles or seconds, as specified in **Waveform period units**.

Note The term clock cycles refers to the high-speed peripheral clock on the F2812 chip. This high-speed peripheral clock is 75 MHz by default because the high-speed peripheral clock prescaler is set to 2 (150 MHz/2).

Dependencies

This parameter appears only when **Waveform period source** is set to `Specify via dialog`.

Duty cycle units — Units for measuring duty cycle
`Percentages` (default) | `Clock cycles`

The units used for measuring the duty cycle.

Duty cycle source — Source from which duty cycle for PWM waveform is obtained
`Specify via dialog` (default) | `Input port`

Select `Specify via dialog` to enter the value in **Duty cycle**, or select `Input port` to use a value from the **W** input port.

Duty cycle — Ratio of PWM waveform pulse duration to PWM waveform period
50 (default)

The ratio of PWM waveform pulse duration to PWM waveform period. This ratio is expressed in **Duty cycle units**.

Output polarity select — Set active level for output
`Active High` (default) | `Active Low`

When you select `Active High`, the compare value (duty cycle) defines the high time. Selecting `Active Low` directs the compare value to define the low time.

Interrupt

Post interrupt on capture event # — Set interrupt source to capture event
`off` (default) | `on`

You can use the C28x Hardware Interrupt block to respond to this interrupt. In this case, **#** represents the number of the capture event set in the **Stop value after** parameter.

Dependencies

This parameter appears only when you set **Operating mode** to `eCAP` in the **General** tab.

Post interrupt on counter overflow — Trigger interrupt on counter overflow
`off` (default) | `on`

Triggers an interrupt when the counter overflows.

Dependencies

This parameter appears only when you set **Operating mode** to `eCAP` in the **General** tab.

Post interrupt on counter equal period match — Post interrupt when counter equals period register
`off` (default) | `on`

Posts interrupt when the value of counter is same as the value of the period register (CTR = PRD).

Dependencies

This parameter appears only when you set **Operating mode** to `APWM` in the **General** tab.

Post interrupt on counter equal compare match — Post interrupt when counter equals compare register
`off` (default) | `on`

Posts interrupt when the value of the counter is same as the value of the compare register (CTR = CMP).

Dependencies

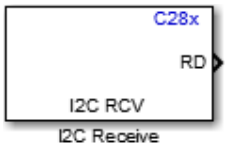
This parameter appears only when you set **Operating mode** to `APWM` in the **General** tab.

See Also

“Overview of Time-Base Synchronization in ePWM Type 4” | C28x Hardware Interrupt | c280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x/F28002x/F28003x ePWM

C28x I2C Receive

Configure inter-integrated circuit (I2C) module to receive data from I2C bus



Libraries:
C2000 Microcontroller Blockset / C2802x
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / C2834x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x
C2000 Microcontroller Blockset / F28M35x / C28x
C2000 Microcontroller Blockset / F28M36x / C28x

Description

The I2C Receive block configures the inter-integrated circuit (I2C) module to receive data from the two-wire I2C serial bus. The I2C Receive block supports I2C bus communication between the processor and external peripherals or other controllers. The block can run in either slave or master mode.

When the I2C module is configured as master, the module receives data from a slave. When the I2C module is configured as a slave, the module receives data from the master. Configure the I2C module by navigating to **Configuration Parameters > Hardware Implementation > Target hardware resources**.

To read data from a slave, send the address of the register to be read using the I2C Transmit block to the slave. Ensure that the data is sent from the Tx FIFO to the slave before the data is read from the slave using the I2C Receive block. For more information, see “Using the I2C Bus to Access Sensors”.

Input/Output Ports

Input

SAR — Slave address register value
scalar

The slave address register value.

Dependencies

This port appears only when **Slave address source** is set to `Input port`.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

Output

RD — Received data from I2C bus
scalar | vector

The data read from the I2C bus.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

status — I2C communication status
scalar

Status values from the I2C status register (I2CSTR).

Dependencies

This port appears only when **Output receiving status** is selected.

Data Types: `uint16`

Parameters

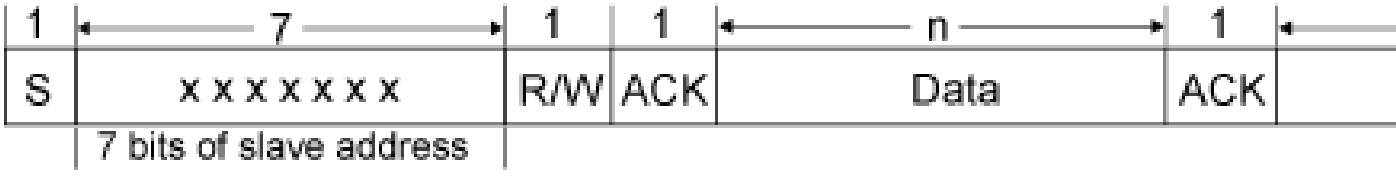
Module — Module for communication
`I2C_A` (default) | `I2C_B`

The I2C module to be used for communication. The number of I2C modules supported varies across different C2000 processors.

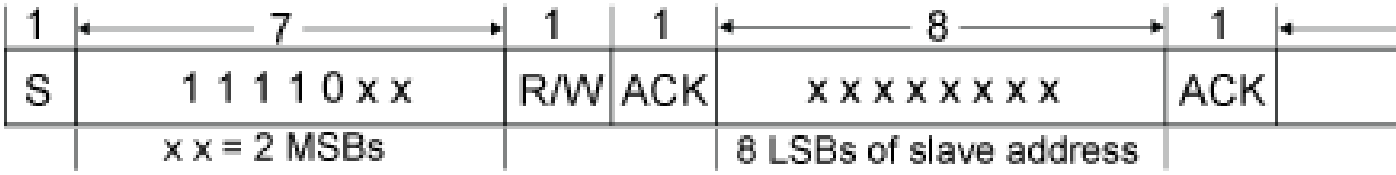
Addressing format — Address format for communication
`7–Bit addressing` (default) | `10–Bit addressing` | `Free data format`

The address format for communication. The diagram shows the format for each option. The I2C Receive block sets the R/W bit to 0.

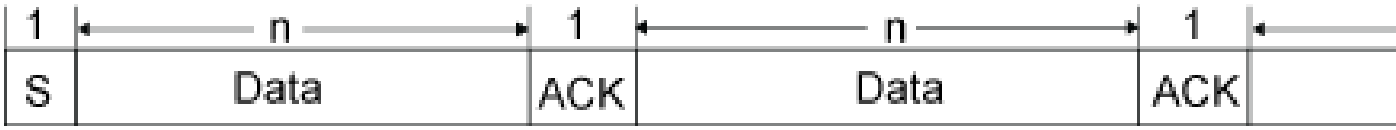
I2C Module 7-bit Addressing Format



I2C Module 10-bit Addressing Format



I2C Module Free Data Format



S — Start bit

R/W — Read/Write

ACK — Acknowledge

P — Stop bit

MSB — Most significant bit

LSB — Least significant bit

Slave address source — Slave address source of I2C slave
`Specify via dialog` (default) | `Input port`

The method for setting the slave address register of the I2C slave.

Slave address register — Slave address register value
`80` (default) | scalar

Enter a 7- or 10-bit slave address according to the addressing format selected.

Dependencies

This parameter appears only when **Slave address source** is set to `Specify via dialog`.

Bit count — Bit count for communication
`8` (default) | integer in [1 8]

The number of bits in the data byte received by the I2C module.

Read data length — Length of received data
`1` (default) | scalar

The number of **Data type** the block receives (not bytes). If this parameter is set to more than `1`, the output will be a vector.

Initial output — Value of I2C node output to model
`0` (default) | scalar | vector

The value the I2C node outputs to the model before it has received data. By default, the block outputs `0` if the I2C value is not received.

Set NACK bit — NACK bit during I2C acknowledge cycle
`off` (default) | `on`

Generates a no-acknowledge bit (NACK) during the I2C acknowledge cycle and ignores new bits from the transmitting I2C node.

Enable stop condition — Stop message to I2C Transmit block
`off` (default) | `on`

Enables the I2C Receive block (master) to send a stop message to the I2C Transmit block (slave).

Output receiving status — Indicates when I2C Receive block receives message
`off` (default) | `on`

Enables the status output port, which indicates when the I2C Receive block receives a message.

Sample time — Frequency at which data is read from I2C device
`0.001` (default) | `–1` | scalar

Sample time for the block in seconds. To execute this block asynchronously, set this parameter to `-1`.

Data type — Type of data in data vector
`int8` (default) | `uint8` | `int16` | `uint16` | `int32` | `uint32`

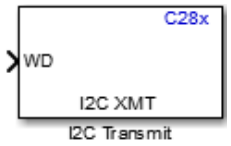
Sets the data type of the data received. If the size of the received data is less than 8 bits, then the data is right-justified.

See Also

C28x I2C Transmit

C28x I2C Transmit

Configure inter-integrated circuit (I2C) module to transmit data to I2C bus



Libraries:
C2000 Microcontroller Blockset / C2802x
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / C2834x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x
C2000 Microcontroller Blockset / F28M35x / C28x
C2000 Microcontroller Blockset / F28M36x / C28x

Description

The I2C Transmit block configures the inter-integrated circuit (I2C) module to transmit data to the two-wire I2C serial bus. The I2C Transmit block supports I2C bus communication between the processor and external peripherals or other controllers. The block can run in either slave or master mode.

When the I2C module is configured as master, the module receives data from a slave. When the I2C module is configured as a slave, the module receives data from the master. Configure the I2C module by navigating to **Configuration Parameters > Hardware Implementation > Target hardware resources**.

To read data from a slave, send the address of the register to be read using the I2C Transmit block to the slave. Ensure that the data is sent from the Tx FIFO to the slave before the data is read from the slave using the I2C Receive block. For more information, see “Using the I2C Bus to Access Sensors”.

Input/Output Ports

Input

SAR — Slave address register value
scalar

The slave address register value.

Dependencies

This port appears only when **Slave address source** is set to `Input port`.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

WD — Data written to I2C bus
scalar | vector

The data written to the I2C bus.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

Output

status — I2C communication status
scalar

Status values from the I2C status register (I2CSTR).

Dependencies

This port appears only when **Output transmitting status** is selected.

Data Types: `uint16`

Parameters

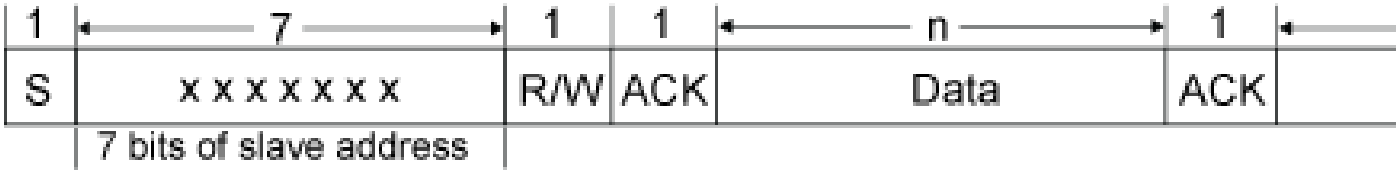
Module — Module for communication
`I2C_A` (default) | `I2C_B`

The I2C module to be used for communication. The number of I2C modules supported varies across different C2000 processors.

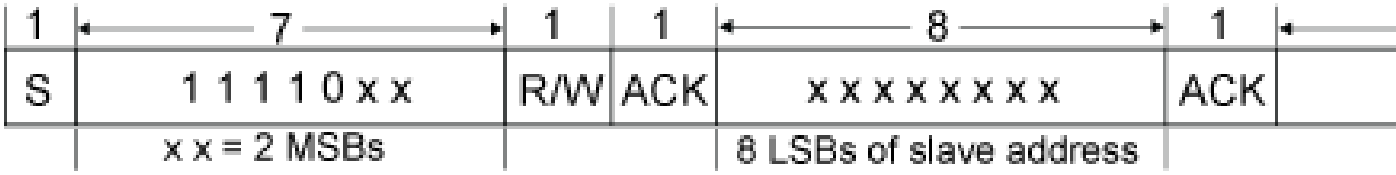
Addressing format — Address format for communication
`7–Bit addressing` (default) | `10–Bit addressing` | `Free data format`

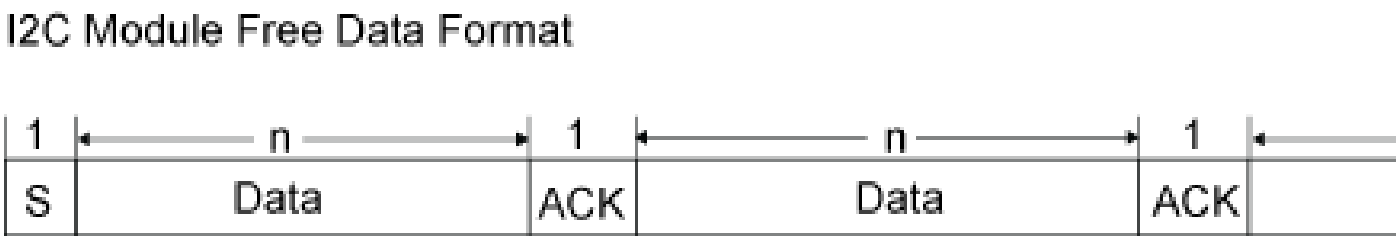
The address format for communication. The diagram shows the format for each option. The I2C Transmit block sets the R/W bit to 1.

I2C Module 7-bit Addressing Format



I2C Module 10-bit Addressing Format





S — Start bit

R/W — Read/Write

ACK — Acknowledge

P — Stop bit

MSB — Most significant bit

LSB — Least significant bit

Slave address source — Slave address source of I2C slave
`Specify via dialog` (default) | `Input port`

The method for setting the slave address register of the I2C slave.

Slave address register — Slave address register value
`80` (default) | scalar

Enter a 7- or 10-bit slave address according to the addressing format selected.

Dependencies

This parameter appears only when **Slave address source** is set to `Specify via dialog`.

Bit count — Bit count for communication
8 (default) | integer in [1 8]

The number of bits in the data byte received by the I2C module.

Enable stop condition — Send stop bit to indicate that data transmission is complete
`off` (default) | `on`

When the I2C module is configured as master, the I2C module sends out a stop bit to the I2C bus to indicate that the data transmission is complete. The I2C bus is free for any other I2C module to initiate a read/write operation.

Enable repeat mode — Retransmit data until stop or start condition is detected
`off` (default) | `on`

When you enable repeat mode, the I2C module transmits data continuously until it detects a stop or start condition. If you use this mode, also consider selecting **Enable stop condition** to ensure that data transmit stops after the stop condition.

If you disable repeat mode, the I2C module operates in standard mode, sending a specific number of data values once.

Output transmitting status — Indicates when I2C transmit block transmits message
`off` (default) | `on`

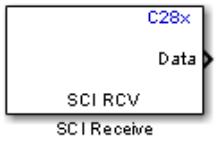
Enables the status output port, which indicates when the I2C transmit block transmits a message.

See Also

C28x I2C Receive

C28x SCI Receive

Receive data on target via serial communication interface (SCI) from host



Libraries:
C2000 Microcontroller Blockset / C2802x
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C281x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / C2834x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x
C2000 Microcontroller Blockset / F28M35x / C28x
C2000 Microcontroller Blockset / F28M36x / C28x

Description

The SCI Receive block supports asynchronous serial digital communication between the processor and other asynchronous peripherals. This block receives scalar or vector data using the specified SCI hardware module.

A model can only contain one SCI Receive block for each module. The C28x processor has four SCI modules — A, B, C, and D. The number of SCI modules available varies depending on the processor selected. You can configure the SCI modules by navigating to **Hardware Implementation > Target hardware resources**. Verify that these settings meet the requirements of your application.

Note Serial External mode with similar SCI module blocks results in conflict check error.

The block outputs data either in blocking mode or in non-blocking mode. In blocking mode, the model blocks the execution while it waits for the requested data to be available. In non-blocking mode, the model runs continuously. To set the block in blocking mode, select the **Wait until data received** option.

Input/Output Ports

Input

Length — Data length received from input port
scalar

The block accepts the length of the data to be received.

Dependencies

To enable this port, set the **Data length option** parameter to `Length via input port`.

Data Types: `uint8` | `uint16` | `uint32`

Output

Data — Data received from serial bus
scalar | vector

The data received from the serial communication bus.

- The data port is configured as **fixed size signal** when data length is set to `Length via dialog`. It outputs only received data.
- The data port is configured as **variable size signal** when data length is set to `Length via Input port` or `Variable Length`. In this case the port outputs received data along with the length of the received data.

  When **Data length** is set as `Variable Length` or `Length via input port` the output data width is `0` if the data is not received. This is independent of connection output type set for **Action taken when connection times out** parameter.

  In order to generate code for variable size signals, go to **Configuration Parameters > Code Generation > Interface** and select **Variable-size signals**. For more information, see

  - “Variable-Size Signal Basics”
  - “Variable-Size Signal Length Adaptation”

Note Currently, Monitor & Tune (External mode) is not supported to output the variable size signal.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single`

status — Status of serial communication
scalar

Indicates the status of the received serial data:

- `0` - No errors.
- `1` - A time-out occurred while the block was waiting to receive data. This is the default status if expected header is not received.
- `2` - The received data contains an error (checksum error) or if the required tail is not received.
- `3` - SCI parity error flag: occurs when a character is received with a mismatch.
- `4` - SCI framing error flag: occurs when an expected stop bit is not found.
- `5` - SCI overrun error flag: occurs when a character is transferred to the receive registers before reading the previous character.
- `6` - SCI break-detect flag: occurs when SCI receiver data line (SCIRXD) remains continuously low for at least ten bits.
- `7` - Data not available flag: occurs in non-blocking mode when data is not available in the FIFO. The status is Data not available when the data length is set to **Variable length** after receiving the header, and if data is not available in the FIFO to read as a data or terminator.

- 8 - Partial Data available: Partial data status represents the data received partially excluding header. The status is set to **Partial Data** when the data length is set to `Variable length`, and if the terminator is expected to be received and when not received up to the `Max data length`.

Dependencies

This port appears only when you select **Output receiving status**.

Data Types: `uint16`

Parameters

SCI module — SCI module for communication
`A` (default) | `B` | `C` | `D`

The SCI module used for communication. The number of SCI modules supported varies across different C2000 processors.

Additional package header — Indicates start of data
`'S'` (default) | string | char | number from 0 to 255

The data located at the front of the received data package, which is not part of the data being received, and indicates the start of data. The additional package header must be represented using ASCII characters. You can use a string or a number (0–255). You must add single quotes around strings entered for this parameter, but the quotes are not received or included in the total byte count. To specify a null value (no package header), enter two single quotes only.

The data type of header is not related to the data type mentioned on the block.

Note

- Match additional package headers or terminators with those specified in the host SCI Transmit block.
- If header is empty then whatever present in the receive buffer will be considered as starting point to receive. In case of data length option as Variable length, the tail will be calculated from this starting point itself if header is empty.
- If the expected header is not provided on the block then the data present in the FIFO is considered as a starting point of reception and from that point onwards required length of the data and terminal will be expected to retrieve.

Additional package terminator — Indicates end of data
`'E'` (default) | string | char | number from 0 to 255

The data located at the end of the received data package, which is not part of the data being received, and indicates the end of data. The additional package terminator must be represented using ASCII characters. Use a string or a number (0–255). You must add single quotes around strings entered for this parameter, but the quotes are not received or included in the total byte count. To specify a null value (no package terminator), enter two single quotes only.

The data type of terminator is not related to the data type mentioned on the block.

Number of retries for header receive check — Custom retrial count for header receive check
`16` (default) | `positive integer, finite` | `scalar`

This parameter ensures to check the expected header as part of receive data.

When the data is received in the FIFO, the block verifies for the header in the received data one by one. If the data does not match with the header, the block will discard the data and continue looking for the header in the next data until the retry count expires.

If the header matches within the retry count, then it is considered as start of the packet and the further received data is considered as valid data.

Data type — Data type of output data
`uint8` (default) | `single` | `int8` | `int16` | `uint16` | `int32` | `uint32`

The data type of the output data.

If data is transmitted as `uint8` but SCI receive block output is expected to be `int16/int32` then it is assumed that even number of bytes are present and that the output is formatted in different data type. In case if there are odd number of data bytes are present then last byte will be missed in the formatting even if the status is showing **No Error**.

Data length option — Data length the block receives
`Length via dialog` (default) | `Length via input port` | `Variable length`

Select the data length option for the block.

- `Length via dialog` - Length of the data to be received is provided via `Data length` parameter.
- `Length via input port` - the block receives variable size data depending on the length received at the input port. If the length received at the input port is greater than the length provided in the maximum data length parameter, then maximum data length is considered.
- `Variable length` - The data is received until the tail matches in non-blocking mode. Ensure that the tail is not present as a part of data. When only header and terminator is available and data is not available then the status is set to **No Error** with variable data length set to zero.
  - If terminator value does not match till maximum length or if the data is not available in between then the block will output the received data along with its length and status will be set to `Partial data available`.
  - If terminator value is not provided the block tries to receive data of **max data length**. If length of the data read is less than max length or if the data is not available in between then the block will output received data with its length and status will be set to `Partial data available`.
  - If data is not read then status is set to **Data not available**.
  - The status is read as data not available when Header and terminator received with no data.
  - Tail to be avoided as part of SCI Data when using in variable length as presence of Tail is treated as end of Data Packet.
  - Output data port length is 0 if no data is received in variable length mode.

To enable input port **length**, select data length option as `Length via input port`.

Data length — Number of data types the block receives
`1` (default) | `positive integer, finite` | `scalar`

The number of **Data type** the block receives (not bytes). If this parameter is set to more than 1, the output will be a vector. Ensure that the data length specified is same as that of the SCI Transmit block from which data is received.

Dependencies

To enable this parameter, set the **Data length option** parameter to `Length via dialog` .

Max data length — Maximum number of data types the block receives
`1` (default) | positive integer, finite | `positive integer, finite`

The maximum number of **Data type** the block can receive (not bytes).

If the data size to be received (based on data length or max data length, header, terminator and data type) is greater than FIFO size bytes then it will result in loss of data.

Dependencies

To enable this parameter, set the **Data length option** parameter to either `Length via input port` or `Variable length`.

Initial output — Default value output from block
`0` (default) | scalar | vector

The default value output from the SCI Receive block. This value is output, for example,when the **Action taken when connection timeout** parameter is set to `Output the last received value` and a connection time-out occurs before data is received.

Action taken when connection times out — Select type of output when connection times out
`Output the last received value` (default) | `Output custom value`

Specifies what to output when a connection time out occurs. If `Output the last received value` is selected, the block outputs the last received value. If a value has not been received, the block outputs the **Initial output** value.

If you select `Output custom value`, use the **Output value when connection times out** parameter to set the custom value.

Output value when connection times out — Custom output value from block when connection times out
`0` (default) | scalar | vector

Set the custom time out value.

Output value when connection times out parameter is available only when **Action taken when connection times out** parameter is set to `Output custom value`.

Sample time — Frequency at which data is read from SCI device
`0.1` (default) | `-1` | scalar

Sample time for the block in seconds. To execute this block asynchronously, set this parameter to `-1`.

Dependencies

To enable this parameter, set the **Data length option** parameter to either `Length via dialog` or `Variable length`.

The sample time will inherit properties from input port when data length option is set to `Length via input port`.

Wait until data received — Wait until requested data is available
`off` (default) | `on`

- `on` — If this option is enabled, the system waits until data is available to read (when data length is reached). The read operation runs in the blocking mode. The read operation is blocked when the block is waiting for the requested data. If data is available, the block outputs the data. If data is not available, the block waits for the data.

  A task overrun occurs if the target hardware is still waiting for the data when the next read operation begins.

  To fix overruns, increase the time step by using the **Sample time** parameter.
- `off` — If this option is disabled, the system checks FIFO at each time step (in polling mode) for data to read. If data is present, the block reads and outputs the contents. If data is not present, the block outputs the last value and continues. When you clear this parameter, the read operation runs in the nonblocking mode.

Dependencies

To enable this parameter, set the **Data length option** parameter to either `Length via input port` or `Length via dialog`.

Note When data length option is selected as **Variable length**, the block execution will always occurs in non-blocking mode.

Timeout — Amount of time in seconds the block waits until data is received
`inf` (default) | `positive value greater than 0`

Specify the amount of time that the block should wait during each time step if the data is not available in the receive FIFO to read. If timeout occurs, the read operation is aborted.

If the value is set to `inf` the block waits infinitely for data to be available in FIFO.

Note 3 types of mode can be achieved with parameters **Wait until data received** and **Timeout**.

- **Blocking mode** - In blocking mode, parameters **Wait until data received** is enabled and **Timeout** set to `inf`.

  In this mode, if data is not available in FIFO to read, it will wait for infinite time until the data is available to read.
- **Blocking mode with Timeout** - In blocking mode with timeout, parameters **Wait until data received** is enabled and **Timeout** set to `any finite value > 0`.

  In this mode if data is not available in FIFO to read, it will wait checking the FIFO status until the timeout value mentioned. If data is not available in FIFO to read within that time then the SCI Receive block will output status as `timeout`.
- **Non-Blocking mode** - In Non-Blocking mode, parameter **Wait until data received** is disabled.

In this mode the SCI Receive block will read the data if the data is available in FIFO else the SCI Receive block will output status as `Data not available`.

In order to receive data of length more than FIFO length use either blocking mode or blocking mode with timeout. This ensures of extra time to get remaining data in FIFO after reading entire FIFO.

In blocking mode with and without timeout enabled, you might encounter task overrun as it waits for the data to read.

Dependencies

To enable this parameter, set the **Data length option** parameter to either `Length via input port` or `Length via dialog` and enable the parameter `Wait until data received`.

Output receiving status — Status of serial communication
`off` (default) | `on`

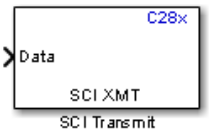
Creates a **Status** block output that provides the status of serial communication.

See Also

C28x SCI Transmit | C28x Hardware Interrupt | “Serial Configuration for External Mode and PIL” on page 1-67

C28x SCI Transmit

Transmit data from target via serial communication interface (SCI) to host



Libraries:
C2000 Microcontroller Blockset / C2802x
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C281x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / C2834x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x
C2000 Microcontroller Blockset / F28M35x / C28x
C2000 Microcontroller Blockset / F28M36x / C28x

Description

The SCI Transmit block transmits scalar or vector data using the specified SCI hardware module. The sampling rate and data type are inherited from the input port.

A model can only contain one SCI Transmit block for each module. The C28x processor has four SCI modules — A, B, C, and D. The number of SCI modules available varies depending on the processor selected. You can configure the SCI modules by navigating to **Hardware Implementation > Target hardware resources**. Verify that these settings meet the requirements of your application. Verify that these settings meet the requirements of your application.

Note

- Fixed-point inputs are not supported by this block, but you can use a Data Type Conversion block to convert the fixed-point format to native data type. In the Data Type Conversion block, set the **Input and output to have equal** parameter to `Stored Integer (SI)`.
- Serial External mode with similar SCI module blocks results in conflict check error.

The block outputs data either in blocking mode or in non-blocking mode. In blocking mode, the model blocks the execution while it waits if transmit FIFO is full. In non-blocking mode, the model runs continuously. To set the block in blocking mode, select the **Wait until previous data transmitted** option.

Input/Output Ports

Input

Data — Data written to serial bus
scalar | vector

Input data written to the serial communication bus.

If the input port of SCI Transmit is of variable data size and If the length of variable data is `0`, then the block will transmit only the header and terminator.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single`

Output

Status — Data transmitted at given time step
scalar

The port outputs `0` if the data is transmitted at a given time step. Otherwise, it outputs value `1`, indicating that transmit FIFO is full and data transmission is not successful.

Dependencies

To enable this port, select the **Output status** parameter.

Data Types: `uint8` | `uint16`

Parameters

SCI module — SCI module for communication
`A` (default) | `B` | `C` | `D`

The SCI module used for communication.

Additional package header — Indicates start of data
`'S'` (default) | string | char | number from 0 to 255

The data located at the beginning of the sent data package, which is not part of the data being transmitted, and indicates the start of data. The additional package header must be represented using ASCII characters. Use a string or a number (0–255). You must add single quotes around strings entered for this parameter, but the quotes are not sent or included in the total byte count. To specify a null value (no package header), enter two single quotes only.

The data type of header is not related to the data type mentioned on the block.

Note Match additional package headers or terminators with those specified in the host SCI Receive block.

Additional package terminator — Indicates end of data
`'E'` (default) | string | char | number from 0 to 255

The data located at the end of the sent data package, which is not part of the data being transmitted, and indicates the end of data. The additional package terminator must be represented using ASCII

characters. Use a string or a number (0–255). You must put single quotes around strings entered in this field, but the quotes are not sent or included in the total byte count. To specify a null value (no package terminator), enter two single quotes only.

The data type of terminator is not related to the data type mentioned on the block.

Frame size — Specify length of data to be received
`600` (default) | any positive integer

Specify the data bytes to be transferred. This parameter can be used when transmit rate is more than the receive rate.

Frame size is calculated by: Receive rate/Transmit rate. For example, if a signal is transmitted at `50` micro sec, and frame size is `600`, you can receive the data at `0.03` sec.

When the frame size is more than `1`, the additional package header and terminator are added at the start and end of the frame. For example, `<Header>DATA(1:frame size)<Terminator>`

Where, `DATA(1:frame size)` indicates the signal to be transmitted of length to frame size.

Wait until previous data transmitted — Wait until data received in previous time step is sent
`on` (default) | `off`

- `on` — When you select this parameter, the transmit operation runs in the blocking mode. In this mode, if transmit FIFO is full then it will wait for previous data to be transmitted and verifies if space is available in the FIFO to transmit the current data.

  A task overrun occurs if the target hardware is still waiting for the requested data to be sent when the next transmit operation is scheduled to begin. To fix overruns increase the time step by using the **Sample time** parameter.
- `off` — When you clear this parameter, the transmit operation runs in the non-blocking mode. In this mode, if the block is still transmitting the data received in the previous time step, the data at the input port in the current time step is dropped.

  In either mode, if the block is yet to establish the connection between the transmitting and receiving hosts. or if the connection is lost, the data at the input port is dropped.

Output status — Option to display the transmit status during data transmission
`off` (default) | `on`

Select this option to display the transmit status during data transmission.

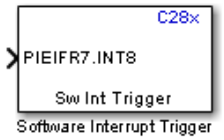
When you select the Output status parameter, the block configures an output port. The port on the block is labeled as Status, indicating that the block outputs the status of the transmit operation at the output port.

See Also

C28x SCI Receive | C28x Hardware Interrupt | “Serial Configuration for External Mode and PIL” on page 1-67

C28x Software Interrupt Trigger

Generate software-triggered nonmaskable interrupt



Libraries:
C2000 Microcontroller Blockset / C2802x
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C281x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / C2834x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x
C2000 Microcontroller Blockset / F28M35x / C28x
C2000 Microcontroller Blockset / F28M36x / C28x

Description

When you add the Software Interrupt Trigger block to a model, the block polls the values on the input port. When the input value is greater than the value in the **Trigger software interrupt when input value is greater than** parameter, the block posts the interrupt corresponding to the selected CPU and Peripheral Interrupt Expansion (PIE) numbers to the Hardware Interrupt block in the model.

To use this block, add a Hardware Interrupt block to your model. The Hardware Interrupt block processes the software-triggered interrupt from this block into an interrupt service routine on the processor. Set the interrupt number in the Hardware Interrupt block to the value you set in the Software Interrupt Trigger block.

The CPU and PIE interrupt numbers together specify a single interrupt for a single peripheral module. For information about the mapping of CPU and PIE interrupt numbers to these peripheral interrupts, see C28x Hardware Interrupt.

Note Fixed-point inputs are not supported by the Software Interrupt Trigger block.

Input/Output Ports

Input

PIEIFRx.INTy — Triggers software interrupt
scalar

The Software Interrupt Trigger block triggers the software interrupt based on the **CUP interrupt number** and **PIE interrupt number** parameters when the input value is greater than the value in the **Trigger software interrupt when input value is greater than** parameter.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

Parameters

CPU interrupt number — CPU interrupt number corresponding to hardware interrupt
`7` (default) | scalar

Enter an integer value to set the CPU interrupt number corresponding to the hardware interrupt. For information about CPU numbers of C2000 processors, see C28x Hardware Interrupt.

PIE interrupt number — PIE interrupt number corresponding to hardware interrupt
`8` (default) | scalar

Enter an integer value to set the PIE number corresponding to the hardware interrupt. For information about PIE numbers of C2000 processors, see C28x Hardware Interrupt.

Trigger software interrupt when input value is greater than — Sets value above which block posts an interrupt
`0` (default) | scalar

Enter the value for the level that indicates that the interrupt is asserted by a requesting routine.

See Also

C28x Hardware Interrupt

C28x SPI Controller Transfer

Write data to and read data from SPI peripheral device

Libraries:
C2000 Microcontroller Blockset / C2802x
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C281x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / C2834x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x
C2000 Microcontroller Blockset / F28M35x / C28x
C2000 Microcontroller Blockset / F28M36x / C28x

Description

The C28x SPI Controller Transfer block writes data to and reads data from a peripheral device over the Serial Peripheral Interface (SPI). The block runs in controller mode. The block outputs an array of the same size and data type as the input values. You can use this block with the Byte Pack and Byte Unpack blocks for heterogeneous data type transfers.

Configure the SPI modules for the specific hardware board by navigating to **Hardware Implementation > Target hardware resources**. Verify that these settings meet the requirements of your application.

Using this block, you can access an SPI device to measure quantities such as temperature and pressure.

Ports

Input

Tx — Data written to registers of SPI peripheral device (MOSI)
vector

The data written by the block to the registers of a peripheral device over the SPI interface.

Data Types: `uint16`

Output

Rx — Data read from registers of SPI peripheral device (MISO)
vector

The data read by the block from the registers of a peripheral device over the SPI interface.

Data Types: `uint16`

Parameters

Main

SPI module — SPI module to write and read data
`SPI_A` (default) | `SPI_B` | `SPI_C` | `SPI_D`

The SPI peripheral module to which the SPI peripheral device is connected. Each processor has a different number of modules.

Clock polarity — SPI clock polarity
`Rising_edge` (default) | `Falling_edge`

The clock polarity (CPOL) for SPI communication mode.

Clock phase — SPI clock phase
`No_delay` (default) | `Delay_half_cycle`

The clock phase (CPHA) for SPI communication mode.

Enable register address — Enables SPI register address
`on` (default) | `off`

Enables the **Register address** parameter.

Register address — SPI register address
`0` (default) | positive integer scalar | positive integer vector

The peripheral register address from which the block reads data.

For example, if we consider the “Using SPI to Read and Write Data to SPI EEPROM” example, to write the EEPROM memory by address `32 (0x0020)` and the write command 2, the entry can be `[2 0 32]` which corresponds to write command followed by 16-bit address and the data at the input port.

Here the address is split as `0x00 (0)` and `0x20 (32)` as two 8-bit numbers and is entered in the register address. The reason for splitting it as two 8-bit numbers is the setting in `Data bits` parameter in `Advanced` tab. Since this is set to 8 bits, the data in the vector format should not be considered more than 8-bit. If you select the parameter as `16`, then `0` and `32` corresponds to 32-bit address `(0x00000020)` instead of 16-bit.

Dependencies

This parameter appears only when you select **Enable register address**.

Advanced

Data bits — Number of bits in SPI transfer
8 (default) | integer in the range [1 16]

Length in bits of each transmitted or received character, specified as an integer in [1 16]. For example, if you select 8, the maximum value that can be transmitted using SPI is $2^8 - 1$. If you send data values greater than this value, the buffer overflows.

Chip select calling method — Method to select SPI peripheral device
`Provided by the SPI peripheral` (default) | `Explicit GPIO calls`

The SPI controller uses these methods to select SPI peripheral devices.

- `Provided by the SPI peripheral` — The SPI controller uses the **STE pin assignment** parameter in **Hardware Implementation > Target Hardware Resources > SPI** to select the peripheral device. peripheral select and deselect are handled by the SPI peripheral.
- `Explicit GPIO calls` — The SPI controller uses the general purpose input/output pins explicitly to select/deselect SPI peripheral devices. The SPI controller Transfer block selects the peripheral before data is transmitted and deselects the peripheral after data is received using GPIO pins.

Chip select pin polarity — SPI peripheral select pin polarity
`Active low` (default) | `Active high`

The logic levels supported by the peripheral select pin to select the SPI peripheral device.

- `Active low` — The device is enabled on logic low. The SPI peripheral device is enabled when its peripheral select pin is set to low.
- `Active high` — The device is enabled on logic high. The SPI peripheral device is enabled when its peripheral select pin is set to high.

Dependencies

This parameter appears only when **Chip select calling method** is set to `Explicit GPIO calls`.

Chip select pin — SPI peripheral select pin
1 (default) | positive integer scalar

The general purpose input/output pin that serves as peripheral select for SPI.

Dependencies

This parameter appears only when **Chip select calling method** is set to `Explicit GPIO calls`.

Version History

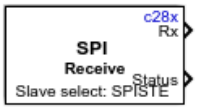
Introduced in R2017b

See Also

C28x SPI Receive | C28x SPI Transmit | C28x Hardware Interrupt

C28x SPI Receive

Receive data through Serial Peripheral Interface (SPI) on target



Libraries:
C2000 Microcontroller Blockset / C2802x
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C281x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / C2834x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x
C2000 Microcontroller Blockset / F28M35x / C28x
C2000 Microcontroller Blockset / F28M36x / C28x

Description

The SPI Receive block supports synchronous, serial peripheral input/output port communications between the processor and external peripherals or other controllers. The block can run in either peripheral or controller mode. In controller mode, the SPISIMO pin transmits data, and the SPISOMI pin receives the data. When controller mode is selected, the SPI initiates the data transfer by sending a serial clock signal (SPICLK), which is used for the entire serial communications link. Data transfers are synchronized to this SPICLK, which enables both controller and peripheral to send and receive data simultaneously. The maximum frequency for the clock is one quarter of the processor clock frequency.

The SPI device receives data and places the data in the receive buffer. The SPI Receive block reads the data from the receive buffer. In controller mode, the C28x SPI Transmit block initiates SPI transmission by writing data to the transmit buffer. Then, the data received in the receive buffer is read by the SPI Receive block. In peripheral mode, the SPI Receive block is used to read the data in the receive buffer, which is received from the controller. Then, the data is written into the transmit buffer using the SPI Transmit block. From the transmit buffer, the data is sent to the controller.

Configure the SPI modules for a specific hardware board by navigating to **Hardware Implementation > Target hardware resources**. Verify that these settings meet the requirements of your application.

Ports

Output

Rx — SPI receive data
vector

The data read from the device over the SPI interface.

Data Types: `uint16`

Status — SPI receive status
`0` | `1` | `2`

Status of receipt of data. Error status values indicate:

- 0 — No errors.
- 1 — Data loss occurred because of overflow.
- 2 — Data not ready. A time out occurred while the block was waiting to receive data.

Dependencies

This port appears only when **Enable blocking mode** is not selected.

Data Types: `uint16`

Parameters

Main

SPI module — SPI module to read data
`SPI_A` (default) | `SPI_B` | `SPI_C` | `SPI_D`

The SPI module to which the SPI peripheral device is connected. Each processor has a different number of modules.

Clock polarity — SPI clock polarity
`Rising_edge` (default) | `Falling_edge`

The clock polarity used for SPI communication mode. This parameter must be the same for both transmit and receive blocks.

Clock phase — SPI clock phase
`No_delay` (default) | `Delay_half_cycle`

The clock phase used for SPI communication mode. This parameter must be the same for both transmit and receive blocks.

Output data length — SPI output data length
1 (default) | positive integer

The received data is a vector of type `uint16` and the data length is as specified in this parameter (not bytes).

Enable blocking mode — Enable SPI blocking mode
`off` (default) | `on`

When this option is selected, the algorithm waits until data is received before continuing processing.

Sample time — SPI sample time selection
`0.1` (default) | `–1` | scalar

Sample time for the block in seconds. To execute this block asynchronously, set this parameter to `-1`.

Advanced

Data bits — Number of bits in SPI transfer
8 (default) | integer in [1 16]

Length in bits of each transmitted or received character. For example, if you select 8, the maximum value that can be transmitted using SPI is 2^8–1. If you send data greater than this value, the buffer overflows. This parameter must be the same for both transmit and receive blocks.

Chip select calling method — Method to select SPI peripheral device
`Provided by the SPI peripheral` (default) | `Explicit GPIO calls`

The SPI controller uses these methods to select SPI peripheral devices:

- `Provided by the SPI peripheral` — The SPI controller uses the STE pin assignment provided in **Hardware Implementation** > **Target hardware resources** > **SPI** to select the peripheral device. peripheral select and deselect are handled by the SPI peripheral.
- `Explicit GPIO calls` — The SPI controller uses the general purpose input/output pins instead of the STE pin of the SPI peripheral to select/deselect SPI peripheral devices. The SPI Receive block deselects the peripheral using GPIO pins after receiving data. To select the peripheral, the C28x SPI Transmit block must be used along with the SPI Receive block. Use this option only in controller mode. Select the **Enable blocking mode** option to ensure that the SPI transmission is complete before the peripheral is deselected.

Chip select pin polarity — SPI peripheral select pin polarity
`Active low` (default) | `Active high`

The logic levels supported by the peripheral select pin to select the SPI peripheral device.

- `Active low` — The device is enabled on logic low. The SPI peripheral device is enabled when its peripheral select pin is set to low.
- `Active high` — The device is enabled on logic high. The SPI peripheral device is enabled when its peripheral select pin is set to high.

Dependencies

This option appears only when **Chip select calling method** is set to `Explicit GPIO calls`.

Chip select pin — SPI peripheral select pin
`0` (default) | positive integer scalar

The general purpose input/output pin that serves as the peripheral select for SPI.

Dependencies

This option appears only when **Chip select calling method** is set to `Explicit GPIO calls`.

Version History

Introduced in R2017b

See Also

C28x SPI Transmit | C28x SPI Controller Transfer | C28x Hardware Interrupt

C28x SPI Transmit

Transmit data through Serial Peripheral Interface (SPI) on target

Libraries:
C2000 Microcontroller Blockset / C2802x
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C281x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / C2834x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x
C2000 Microcontroller Blockset / F28M35x / C28x
C2000 Microcontroller Blockset / F28M36x / C28x

Description

The SPI Transmit block supports synchronous, serial peripheral input/output port communications between the processor and external peripherals or other controllers. The block can run in either peripheral or controller mode. In controller mode, the SPISIMO pin transmits data, and the SPISOMI pin receives the data. When controller mode is selected, the SPI initiates the data transfer by sending a serial clock signal (SPICLK), which is used for the entire serial communications link. Data transfers are synchronized to this SPICLK, which enables both controller and peripheral to send and receive data simultaneously. The maximum frequency for the clock is one quarter of the processor clock frequency.

The SPI Transmit block writes data to the transmit buffer, and the data is transmitted to the SPI device. In controller mode, the SPI Transmit block initiates SPI transmission by writing the data to the SPI transmit buffer. The C28x SPI Receive block must be used along with the SPI Transmit block to read the data received in the receive buffer. In peripheral mode, the SPI Receive block is used to read the data in the receive buffer, which is received from the controller. Then, the data is written into the transmit buffer using the SPI Transmit block. From the transmit buffer, the data is sent to the controller.

The sampling rate is inherited from the input port. The supported data type is `uint16`.

When the SPI transmit interrupt is configured, the transmit FIFO interrupt flags are cleared in the step function instead of the interrupt service routine. After the data is placed in the transmit buffer, the transmit FIFO interrupt is set and the previous transmit interrupt FIFO flags are cleared. Configure the SPI modules for a specific hardware board by navigating to **Hardware Implementation** > **Target hardware resources**. Verify that these settings meet the requirements of your application.

Ports

Input

Tx — SPI Transmit data
vector

The data written to the device over the SPI interface.

Data Types: `uint16`

Output

Status — SPI transmit status
0 | 1 | 2

Status of SPI data transmission. Error status values indicate:

- 0 — No errors.
- 1 — A time-out occurred while the block was transmitting data.
- 2 — The transmitted data contains an error.

Dependencies

This port appears only when **Enable blocking mode** is not selected.

Data Types: `uint16`

Parameters

Main

SPI module — SPI module to write data
`SPI_A` (default) | `SPI_B` | `SPI_C` | `SPI_D`

The SPI module to which the SPI peripheral device is connected. Each processor has a different number of modules.

Clock polarity — SPI clock polarity
`Rising_edge` (default) | `Falling_edge`

The clock polarity used for SPI communication mode. This parameter must be the same for both transmit and receive blocks.

Clock phase — SPI clock phase
`No_delay` (default) | `Delay_half_cycle`

The clock phase used for SPI communication mode. This parameter must be the same for both transmit and receive blocks.

Enable blocking mode — SPI blocking mode enable
`off` (default) | `on`

When this option is selected, the algorithm waits until data is sent before continuing processing.

Advanced

Data bits — Number of bits in SPI transfer
8 (default) | integer in [1 16]

Length in bits of each transmitted or received character. For example, if you select 8, the maximum value that can be transmitted using SPI is 2^8–1. If you send data greater than this value, the buffer overflows. This parameter must be the same for both transmit and receive blocks.

Chip select calling method — Method to select SPI peripheral device
`Provided by the SPI peripheral` (default) | `Explicit GPIO calls`

The SPI controller uses these methods to select SPI peripheral devices:

- `Provided by the SPI peripheral` — The SPI controller uses the STE pin assignment provided in **Hardware Implementation > Target hardware resources > SPI** to select the peripheral device. peripheral select and deselect are handled by the SPI peripheral.
- `Explicit GPIO calls` — The SPI controller uses the general purpose input/output pins instead of the STE pin of the SPI peripheral to select/deselect SPI peripheral devices. The SPI Transmit block selects the peripheral using GPIO pins before transmitting data. To deselect the peripheral, you must use the C28x SPI Receive block along with the SPI Transmit block. Use this option only in controller mode. Select the **Enable blocking mode** option to ensure that the SPI transmission is complete before the peripheral is deselected.

Chip select pin polarity — SPI chip select pin polarity
`Active low` (default) | `Active high`

The logic levels supported by chip select pin to select the SPI peripheral device.

- `Active low` — The device is enabled on logic low. The SPI peripheral device is enabled when its peripheral select pin is set to low.
- `Active high` — The device is enabled on logic high. The SPI peripheral device is enabled when its peripheral select pin is set to high.

Dependencies

This option appears only when **Chip select calling method** is set to `Explicit GPIO calls`.

Chip select pin — SPI peripheral select pin
`0` (default) | positive integer scalar

The general purpose input/output pin that serves as the peripheral select for SPI.

Dependencies

This option appears only when **Chip select calling method** is set to `Explicit GPIO calls`.

Version History

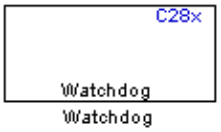
Introduced in R2017b

See Also

C28x SPI Receive | C28x SPI Controller Transfer | C28x Hardware Interrupt

C28x Watchdog

Configure counter reset source of processor watchdog module



Libraries:
C2000 Microcontroller Blockset / C2802x
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C281x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / C2834x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x

Description

This block configures the counter reset source of the watchdog module on the processor. The watchdog module, after configuration, resets the system if not serviced periodically.

Ports

Input

Input — Resets watchdog counter
scalar

When the input signal is `1`, the counter is reset.

Dependencies

This parameter appears only when **Watchdog counter reset source** is set to `Input port`.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

Parameters

Watchdog counter reset source — Watchdog counter reset source
`Specify via dialog` (default) | `Input port`

The watchdog counter reset source.

- `Input` — Create an input port on the watchdog block. When the input signal is `1`, the counter is reset.

- `Specify via dialog` — The watchdog timer is reset based on the **Sample time** value.

Sample time — Frequency at which processor resets Watchdog timer
–1 (default) | scalar

Sample time for the block in seconds. To execute this block asynchronously, set this parameter to `-1`.

Dependencies

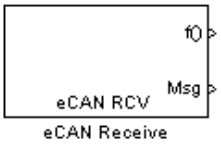
This parameter appears only when **Watchdog counter reset source** is set to `Specify via dialog`.

See Also

C28x Hardware Interrupt

C28x eCAN Receive

Enhanced Controller Area Network receive mailbox



Libraries:
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C281x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / C2834x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x

Description

The eCAN Receive block generates source code for receiving enhanced Controller Area Network (eCAN) messages through an eCAN mailbox. eCAN modules on the processor provide serial communication capability and have 32 mailboxes configurable for receive or transmit. The block supports eCAN data frames in standard or extended format.

To use the eCAN Receive block with the eCAN Pack block in the `canmsglib` library, set **Data type** to `CAN_MESSAGE_TYPE`.

Configure the eCAN modules for a specific hardware board by navigating to **Hardware Implementation** > **Target hardware resources**. Verify that these settings meet the requirements of your application.

Ports

Output

f() — Function call port
scalar

Connect a function call subsystem to this port. When a new message is received, the subsystem is executed.

Msg — Message data port
vector

The received data is output in the form of a vector of elements of the selected data type. The length of the vector is 8 bytes. When the block is used in polling mode, and a new message is not created between consecutive executions of the block, the existing message is repeated.

To use the eCAN Receive block with the CAN Unpack block in the `canmsglib` library, set **Data type** to `CAN_MESSAGE_TYPE`.

Data Types: `uint8` | `uint16` | `uint32` | `CAN_MESSAGE_TYPE`

len — Length of output message
scalar

The length of output message received in bytes.

Dependencies

This port appears only if the **Output message length** parameter is selected.

Data Types: `uint16`

ID — Output message identifier
scalar

The port outputs message identifier of received message.

Dependencies

To enable this port, select **Output message ID** parameter.

Data Types: `uint32`

Parameters

Module — eCAN module the block configures
`eCAN_A` (default) | `eCAN_B`

Determines the eCAN module configured by the eCAN Receive block.

Mailbox number(0–31) — Sets value of mailbox number register (MBNR)
`0` (default) | integer in the range [0 31]

For standard CAN controller (SCC) mode, enter a unique number from `0` to `15`. For high-end CAN controller (HECC) mode, enter a unique number from `0` to `31` . In SCC mode, transmissions from the mailbox with the highest number have the highest priority. In HECC mode, the mailbox number only determines priority if the transmit priority level (TPL) of two mailboxes is equal.

Note In a model, the same mailbox number cannot be used by either eCAN Transmit block or other eCAN Receive blocks.

Message identifier — Sets value of message identifier register (MID)
bin2dec('111000111') (default) | numeric identifier of length 11 or 29 bits

The message identifier is 11 bits long for the standard frame size or 29 bits long for the extended frame size in decimal, binary, or hex format. For binary and hex formats, use `bin2dec(' ')` and `hex2dec(' ')`, respectively, to convert the entry.

Message type — Message identifier type
`Standard (11-bit identifier)` (default) | `Extended (29-bit identifier)`

The message identifier type.

Enable acceptance filter — Enable to use ID filtering
`off` (default) | `on`

Enable this parameter to use ID filtering. ID filtering depends on **Message type (Standard or extended)** , **Message identifier** and **Message identifier mask**.

eCAN module scans the received message ID's and compares it with the filter mask. Received messages which pass the acceptance filtering are stored into the message RAM.

Note Acceptance filtering in self test mode is only supported for specific processors. F2805x/F2806x and older processors does not support acceptance filtering in self test mode.

Message identifier mask — identifier mask
`0` (default) | `1` | numeric identifier of length 11 or 29 bits

The **Message identifier mask** is 11 bits long for the standard frame size or 29 bits long for the extended frame size in decimal, binary, or hex format. For binary and hex formats, use `bin2dec(' ')` and `hex2dec(' ')`, respectively, to convert the entry.

- 0 - if the bit in **Message identifier mask** is set to `0`, then corresponding bit in the **Message identifier** will not be considered for acceptance filtering.
- 1 - if the bit in **Message identifier mask** is set to `1`, then corresponding bit in the **Message identifier** will be considered for acceptance filtering.

Dependencies

To enable **Message identifier mask parameter**, select **Enable acceptance filter**.

Sample time — Frequency at which mailbox is polled for new message
`1` (default) | `-1` | scalar

A new message causes a function call to be sent from the mailbox. If you want to update the message output only when a new message arrives, the block needs to be executed asynchronously. To execute the block asynchronously, set this parameter to `-1`, and select the **Post interrupt when message is received** option.

Note For information about setting the timing parameters of the CAN module, see “Configuring Timing Parameters for CAN Blocks”.

Data type — Output message data type
`uint16` (default) | `uint8` | `uint32` | `CAN_MESSAGE_TYPE`

The options available are:

- `uint8`: Vector length = 8 elements
- `uint16`: Vector length = 4 elements
- `uint32`: Vector length = 2 elements
- `CAN_MESSAGE_TYPE`: Outputs data as a structure. Use the CAN Unpack block to extract the data from the structure.

The length of the vector for the received message is at most 8 bytes. If the message is less than 8 bytes, the data buffer bytes are right-aligned in the output. The data are unpacked as follows using the data buffer, which is 8 bytes.

For `uint8` data, the eCAN Receive block reads each unit of 8 bytes in the registers and outputs 8-bit data to eight elements (using the lower part of the 16-bit memory).

```
Output[0] = data_buffer[0];
Output[1] = data_buffer[1];
Output[2] = data_buffer[2];
Output[3] = data_buffer[3];
Output[4] = data_buffer[4];
Output[5] = data_buffer[5];
Output[6] = data_buffer[6];
Output[7] = data_buffer[7];
```

For `uint16` data, the eCAN Receive block reads each unit of 8 bytes in the registers and outputs 16-bit data to four elements.

```
Output[0] = data_buffer[1..0];
Output[1] = data_buffer[3..2];
Output[2] = data_buffer[5..4];
Output[3] = data_buffer[7..6];
```

For `uint32` data, the eCAN Receive block reads each unit of 8 bytes in the registers and outputs 32-bit data to two elements.

```
Output[0] = data_buffer[3..0];
Output[1] = data_buffer[7..4];
```

For example, if the received message has two bytes:

```
data_buffer[0] = 0x21
data_buffer[1] = 0x43
```

The `uint16` output is:

```
Output[0] = 0x4321
Output[1] = 0x0000
Output[2] = 0x0000
Output[3] = 0x0000
```

When you select `CAN_MESSAGE_TYPE`, the block outputs the following struct data (defined in `can_message.h`):

```
struct {

  /* Is Extended frame */
  uint8_T Extended;

  /* Length */
  uint8_T Length;

  /* RTR */
  uint8_T Remote;

  /* Error */
  uint8_T Error;

  /* CAN ID */
  uint32_T ID;

  /*
```

```
  TIMESTAMP_NOT_REQUIRED is a macro that will be defined by Target teams
  PIL, if they do not require the timestamp field during code
  generation. By default, timestamp is defined. If the targets do not require
  the timestamp field, they should define the macro TIMESTAMP_NOT_REQUIRED before
  including this header file for code generation.
  */
  #ifndef TIMESTAMP_NOT_REQUIRED
    /* Timestamp */
    double Timestamp;
  #endif

  /* Data field */
  uint8_T Data[8];

};
```

Limitations

The following are the limitations when **Data type** is set as `CAN_MESSAGE_TYPE`.

- CAN unpack status can be used for data validity only when CAN unpack is directly connected to eCAN Receive block without **function trigger** and **Action when message not received** to `Reset all fields to zero`.
- CAN unpack status will not be valid in any case if **Action when message not received** of eCAN Receive block is set to `Output last received values`.
- Always use function trigger as indication of status with or without CAN unpack block for all scenarios.

Initial output — Sets output before receiving data
`0` (default) | integer

The value of the output before receiving data. You can specify the initial output values based on the **Data type** selected.

Action when message not received — Action when no new CAN message received
`Output last message received` (default) | `Reset all fields to zero`

Select an action when no new CAN message is received. You can either choose `Output last message received` or `Reset all fields to zero`.

Output message ID — Output message identifier status
`off` (default) | `on`

When you select the **Output message ID** parameter, the block configures an output port, **ID**. The port outputs the status of output message identifier.

Dependencies

To enable **Output message ID**, select **Enable acceptance filter**.

Output message length — Output message length in bytes
`off` (default) | `on`

The message length in bytes, sent to the **len** port. If not selected, the **len** port is not visible.

Post interrupt when message is received — Posts asynchronous interrupt when message is received
`off` (default) | `on`

When selected, the block posts an asynchronous interrupt when a message is received.

Interrupt line — Interrupt line of asynchronous interrupt
`0` (default) | `1`

The interrupt line the asynchronous interrupt uses. The value of this parameter sets bit 2 (GIL) in the global interrupt mask register (CANGIM):

- `1` maps the global interrupts to the ECAN1INT line.
- `0` maps the global interrupts to the ECAN0INT line.

Dependencies

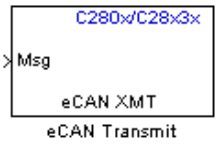
This parameter appears only when you select **Post interrupt when message is received**.

See Also

C28x eCAN Transmit | C28x Hardware Interrupt

C28x eCAN Transmit

Enhanced Controller Area Network transmit mailbox



Libraries:
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C281x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / C2834x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x

Description

The eCAN Transmit block generates source code for transmitting enhanced Controller Area Network (eCAN) messages through an eCAN mailbox. eCAN modules on the processor provide serial communication capability and have 32 mailboxes configurable for receive or transmit. This block supports eCAN data frames in standard or extended format.

Note Fixed-point inputs are not supported by this block.

Configure the eCAN modules for a specific hardware board by navigating to **Hardware Implementation** > **Target hardware resources**. Verify that these settings meet the requirements of your application.

Ports

Input

Msg — Message data
vector

Input message data.

Data Types: `uint8` | `uint16` | `uint32` | `CAN_MESSAGE_TYPE`

Parameters

Module — eCAN module the block configures
`eCAN_A` (default) | `eCAN_B`

Determines the eCAN module configured by this instance of the eCAN Transmit block.

Mailbox number(0–31) — Sets value of mailbox number register (MBNR)
`1` (default) | integer in the range [0 31]

A unique number from 0 to 15 for standard or from 0 to 31 for enhanced CAN mode. The number refers to a mailbox area in RAM. In standard mode, the mailbox number determines priority.

Note In a model, multiple eCAN Transmit blocks can have a same `Mailbox number`. But the same mailbox number cannot be used by eCAN Receive block.

Message identifier — Value of message identifier register (MID)
bin2dec('111000111') (default) | numeric identifier of length 11 or 29 bits

The message identifier is 11 bits long for the standard frame size or 29 bits long for the extended frame size in decimal, binary, or hex. For binary and hex formats, use `bin2dec(' ')` and `hex2dec(' ')`, respectively, to convert the entry. The message identifier is coded into a message that is sent to the CAN bus.

Note CAN messages use the value of the message identifier parameter in the CAN Pack block for transmission when it is used with C28x CAN Transmit block to create the CAN message.

Message type — Message identifier type
`Standard (11-bit identifier)` (default) | `Extended (29-bit identifier)`

The message identifier type.

Enable blocking mode — Sets blocking mode
`off` (default) | `on`

If selected, the CAN block waits indefinitely for a transmit (XMT) acknowledgment. If not selected, the CAN block does not wait for a transmit (XMT) acknowledgment, which is useful if the hardware fails to acknowledge transmissions.

Post interrupt when message is transmitted — Posts asynchronous interrupt when message is transmitted
`off` (default) | `on`

When selected, this block posts an asynchronous interrupt when data is transmitted.

Interrupt Line — Interrupt line of asynchronous interrupt
`0` (default) | `1`

The interrupt line the asynchronous interrupt uses. The value of this parameter sets bit 2 (GIL) in the global interrupt mask register (CANGIM):

- `1` maps the global interrupts to the ECAN1INT line.
- `0` maps the global interrupts to the ECAN0INT line.

Note For information about setting the timing parameters of the CAN module, see “Configuring Timing Parameters for CAN Blocks”.

Dependencies

This parameter appears only when **Post interrupt when message is transmitted** is selected.

More About

Data Vectors

The length of the vector for each transmitted mailbox message is 8 bytes. Input data are right-aligned in the message data buffer. The `uint8` (vector length = 8 elements), `uint16` (vector length = 4 elements), and `uint32` (vector length = 2 elements) data types are accepted. While using the eCAN Transmit block with the CAN Pack block in the `canmsglib` library, `CAN_MESSAGE_TYPE` is also accepted.

The following examples show how different types of input data are aligned in the data buffer:

For input of data type `uint32`,

```
inputdata [0] = 0x12345678
```

the data buffer is:

```
data buffer[0] = 0x78
data buffer[1] = 0x56
data buffer[2] = 0x34
data buffer[3] = 0x12
data buffer[4] = 0x00
data buffer[5] = 0x00
data buffer[6] = 0x00
data buffer[7] = 0x00
```

For input of data type `uint16`,

```
inputdata [0] = 0x1234
```

the data buffer is:

```
data buffer[0] = 0x34
data buffer[1] = 0x12
data buffer[2] = 0x00
data buffer[3] = 0x00
data buffer[4] = 0x00
data buffer[5] = 0x00
data buffer[6] = 0x00
data buffer[7] = 0x00
```

For input of data type `uint16[2]`, which is a two-element vector,

```
inputdata [0] = 0x1234
inputdata [1] = 0x5678
```

the data buffer is:

```
data buffer[0] = 0x34
data buffer[1] = 0x12
data buffer[2] = 0x78
data buffer[3] = 0x56
```

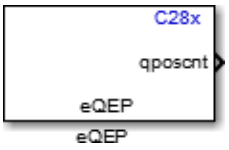
```
data buffer[4] = 0x00
data buffer[5] = 0x00
data buffer[6] = 0x00
data buffer[7] = 0x00
```

See Also

C28x eCAN Receive | C28x Hardware Interrupt

C28x eQEP

Quadrature encoder pulse block used to derive position, direction, and speed



Libraries:
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / C2834x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x
C2000 Microcontroller Blockset / F28M35x / C28x
C2000 Microcontroller Blockset / F28M36x / C28x

Description

The enhanced quadrature encoder pulse (eQEP) block is used along with a linear or rotary incremental encoder to get position, direction, and speed information from a rotating machine.

The eQEP peripheral module inputs include QEPA, QEPB, QEPI (index), and QEPS (strobe).

To configure your device to work with the block, navigate to **Model Configuration Parameters > Hardware Implementation**, select your device at **Hardware board**, and expand **Target hardware resources**.

Input/Output Ports

Input

swi — Dynamically update initialization value for position counter
scalar

If the input is `true`, the position counter is initialized to the value in the **Initialization value (0~4294967295)** on page 2-0 parameter.

Dependencies

This port appears only when, in the **Position counter** tab, you select **Enable software initialization** and set **Software initialization source** to `Input port`.

Data Types: `Boolean`

cmp — Value for comparing position
scalar

Input value that generates the position compare sync signal.

Dependencies

This port appears only when, in the **Compare output** tab, you select **Enable position-compare sync signal output** and set **Compare value source** to `Input port`.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

iel — Software index marker
scalar

Software index event marker for latching the position counter.

Dependencies

This port appears only when, in the **Position counter** tab:

- You set **Position counter reset mode** to `Reset on the maximum position` or `Reset on the first index event`.
- You select **Output latch position counter on index event**.
- You set **Index event latch of position counter** to `Software index marker via input port`.

Note In **Compare output** tab, if **Sync output pin selection** is set to `Index pin is used for sync output`, then the **Index event latch of position counter** parameter cannot be set to `Software index marker via input port`.

Data Types: `Boolean`

Output

qposcnt — Position counter signal
scalar

Position counter signal (PCSOUT) received from the position counter and control unit (PCCU).

Dependencies

This port appears only when, in the **Position counter** tab, you select **Output position counter**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

pcef — Position counter error flag on error
scalar

Outputs the position counter error flag on an error.

- `0` — No error occurred during the last index transition.
- `1` — Position counter error.

Dependencies

This port appears only when, in the **Position counter** tab:

- You set **Position counter reset mode** to `Reset on an index event`.
- You select **Output position counter error flag**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

qdf — Direction flag of quadrature module
scalar

Direction flag of the quadrature module.

- `0` — counterclockwise rotation (or reverse movement).
- `1` — clockwise rotation (or forward movement).

Dependencies

This port appears only when, in the **General** tab, you select **Quadrature direction flag output port**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

qctmr — Capture timer signal
scalar

Outputs the eQEP capture timer signal.

Dependencies

This port appears only when, in the **Speed calculation** tab, you select **Enable eQEP capture** and **Output capture timer**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

qcprd — Capture period signal
scalar

Outputs the capture period signal, which holds the period count value between the last successive eQEP position events.

Dependencies

This port appears only when, in the **Speed calculation** tab, you select **Enable eQEP capture** and **Output capture period timer**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

coef — eQEP overflow error flag
scalar

Outputs overflow error flag (COEF flag) in the event of capture timer overflow between unit position events.

- `0` — Overflow has not occurred.
- `1` — Overflow occurred in the eQEP capture timer register (QEPCTMR).

Dependencies

This port appears only when, in the **Speed calculation** tab, you select **Enable eQEP capture** and **Enable and output overflow error flag**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

cdef — Direction change error flag
scalar

Outputs the direction change error flag.

- `0` — Capture direction error has not occurred.
- `1` — Direction change occurred between two capture position events.

Dependencies

This port appears only when, in the **Speed calculation** tab, you select **Enable eQEP capture** and **Enable and output direction change error flag**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

qctmrlat — Capture timer latched value
scalar

Outputs the capture timer latched value from the QCTMRLAT register.

Dependencies

This port appears only when, in the **Speed calculation** tab, you select **Enable eQEP capture** and **Output capture timer latched value**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

qcprdlat — Capture timer period latched value
scalar

Outputs the capture timer period latched value from the QCPRDLAT register.

Dependencies

This port appears only when, in the **Speed calculation** tab, you select **Enable eQEP capture** and **Output capture timer period latched value**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

qposlat — Position counter latched value
scalar

Outputs position counter latched value from the QPOSLAT register.

Dependencies

This port appears only when, in the **Speed calculation** tab, you select **Enable eQEP capture** and **Output position counter latched value**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

qposilat — Latches position counter on index event
scalar

eQEP index input can be configured to latch the position counter register (QPOSCNT) as output on the occurrence of a definite event on this pin.

Dependencies

This port appears only when, in the **Position counter** tab:

- You set **Position counter reset mode** to `Reset on the maximum position` or `Reset on the first index event`.
- You select **Output latch position counter on index event**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

qposslat — Latches position counter on strobe event
scalar

eQEP strobe input can be configured to latch the position counter register (QPOSCNT) as output on the occurrence of a definite event on this pin.

Dependencies

This port appears only when, in the **Position counter** tab:

- You set **Position counter reset mode** to `Reset on the maximum position` or `Reset on the first index event`.
- You select **Output latch position counter on strobe event**.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `single` | `double` | `Boolean`

Parameters

General

Module — eQEP module to obtain position, direction, and speed
`eQEP1` (default) | `eQEP2` | `eQEP3`

The eQEP peripheral module used to obtain position, direction, and speed information. The number of eQEP modules supported varies for different C2000 processors.

Position counter mode — Mode that matches how input to eQEP peripheral is encoded
`Quadrature-count` (default) | `Direction-count` | `Up-count` | `Down-count`

The eQEP peripheral inputs are QEPA, QEPB, QEPI, and QEPS. Configure the GPIO pins for these inputs by navigating to **Configuration Parameters > Hardware Implementation > Target hardware resources > eQEP**.

Input signals QEPA and QEPB to the eQEP peripheral are processed by the quadrature decoder unit (QDU) in the eQEP peripheral to produce clock (QCLK) and direction (QDIR) signals. Choose the position counter mode that matches the way the input to the eQEP module is encoded:

- `Quadrature-count` — Two square signals (A and B) 90 degrees out of phase are sent to the eQEP peripheral. The QDU uses the phase relationship of these two inputs to generate quadrature clock and direction signals.
- `Direction-count` — Direction and clock signals are directly sent to the eQEP peripheral. The QEPA pin provides the clock input, and the QEPB pin provides the direction input.
- `Up-count` — The position counter is used to measure the frequency of the signal in the QEPA pin. The direction is hard-wired for up count in this mode.

- `Down-count` — The position counter is used to measure the frequency of the signal in the QEPA pin. The direction is hard-wired for down count in this mode.

Positive rotation — Direction of rotation
`Clockwise` (default) | `Counterclockwise`

Set the direction of rotation.

If `Clockwise` is selected, the quadrature count operation is performed without swapping the quadrature clock inputs fed to the QDU.

If `Counterclockwise` is selected, reverse counting is performed by swapping the quadrature clock inputs fed to the QDU. The quadrature decoder reverses the counting direction.

Dependencies

This parameter appears only when you set **Position counter mode** to `Quadrature-count` on the **General** tab.

External clock rate — Measurement resolution
`2x resolution: Count the rising/falling edge` (default) | `1x resolution: Count the rising edge only`

Select the resolution of the clock generator that the position counter uses as input so that the counting occurs on both rising and falling edges of the QEPA input or on the rising edge only. Choosing the former increases the measurement resolution by a factor of 2.

Dependencies

This parameter appears only when you set **Position counter mode** to `Direction-count`, `Up-count`, or `Down-count`.

Quadrature direction flag output port — Creates port for direction flag
`off` (default) | `on`

Creates a port (qdf) on the block that gives the direction flag of the quadrature module.

Dependencies

This parameter appears only when, in the **General** tab, **Position counter mode** is set to `Quadrature-count`.

Invert input QEPxX polarity — Inverts polarity of eQEP input
`off` (default) | `on`

Inverts the polarity of the eQEP peripheral inputs. The **Invert input QEPxA polarity** checkbox corresponds to QEPA, **Invert input QEPxB polairty** corresponds to QEPB, and so on.

Index pulse gating option — Enables gating of index pulse with strobe input
`off` (default) | `on`

Enables the gating of the peripheral input index signal with the peripheral input strobe signal. In this case, the internal index signal is high only when both the peripheral input signals eQEPxI and eQEPxS are high.

Sample time — Frequency at which block reads position counter
`0.0001` (default) | `-1` | scalar

Sample time for the block in seconds. To execute this block asynchronously, set this parameter to `-1`.

Position counter

Output position counter — Outputs position counter signal
`on` (default) | `off`

Outputs the position counter signal PCSOUT from the position counter and control unit (PCCU). The position counter register counts up or down on every eQEP pulse based on the direction of the input. The count value is proportional to the position from a given reference point.

Maximum position counter value (0~4294967295) — Specifies maximum position counter value
`4294967295` (default) | integer in [0, 4,294,967,295]

Enter a maximum value (QPOSMAX) for the position counter. If the position counter reaches QPOSMAX, the position counter is set to `0` on the next increment of the counter. If the position counter is `0`, the position counter is set to QPOSMAX on the next decrement of the counter.

Enable set to init value on index event — Enables option to set initialization value for position counter
`off` (default) | `on`

Enables option to set the position counter to its initialization value on an index event.

Set to init value on index event — Initialization value for position counter
`Rising edge` (default) | `Falling edge`

Sets the position counter to its initialization value on the rising edge or falling edge of the index event.

Dependencies

This parameter appears only when, in the **Position counter** tab, you select **Enable set to init value on index event**.

Enable set to init value on strobe event — Enables option to set initialization value for position counter
`off` (default) | `on`

Enables option to set the position counter to its initialization value on a strobe event.

Set to init value on strobe event — Sets initialization value for position counter
`Rising edge` (default) | `Depending on direction`

The `Rising edge` option sets the position counter to its initialization value on the rising edge of the strobe input. The `Depending on direction` option sets the position counter to its initialization value on the:

- rising edge of the strobe input, in the forward direction.
- falling edge of the strobe input, in the reverse direction.

Dependencies

This parameter appears only when, in the **Position counter** tab, you select **Enable set to init value on strobe event**.

Enable software initialization — Enables option to set initialization value for position counter
`off` (default) | `on`

Allows the position counter to be set to its initialization value using the software.

Software initialization source — Specifies initialization source of position counter
`Input port` (default) | `Set to init value at start up`

Choose the `Set to init value at start up` option to initialize the position counter to the value entered in **Initialization value** at the start of the execution of the program. Choose the `Input port` option to update the initialization value dynamically based on an input initialization signal (input port **swi**). If the input **swi** is `true`, the position counter is initialized to the **Initialization value**.

Dependencies

This parameter appears only when, in the **Position counter** tab, you select **Enable software initialization**.

Initialization value (0~4294967295) — Initialization value for position counter
`2147483648` | integer in [0, 4,294,967,295]

Enter the initialization value for the position counter.

Dependencies

This parameter appears only when you select **Enable set to init value on index event**, **Enable set to init value on strobe event**, or **Enable software initialization**.

Position counter reset mode — Resets position counter
`Reset on an index event` (default) | `Reset on the maximum position` | `Reset on the first index event` | `Reset on a time unit event`

Position counter reset mode, depending on the nature of the system the eQEP module is working with.

- `Reset on an index event` — If the index event occurs during the forward direction, then the position counter is reset to 0 on the next eQEP clock. If the index event occurs during the reverse direction, then the position counter is reset to the value in the QPOSMAX register on the next eQEP clock.
- `Reset on the maximum position` — During the forward direction, when the position counter is equal to QPOSMAX, the position counter is reset to 0 on the next eQEP clock, and the position counter overflow flag is set. During the reverse direction, when the position counter is equal to 0, the position counter is reset to QPOSMAX on the next QEP clock, and the position counter underflow flag is set.
- `Reset on the first index event` — If the index event occurs during the forward direction, the position counter is reset to 0 on the next eQEP clock. If the index event occurs during the reverse direction, the position counter is reset to the value in the QPOSMAX register on the next eQEP clock. The position counter is reset using the `Reset on the first index event` option only on the first index event occurrence. After the first index event occurrence, the position counter is reset based on the maximum position.
- `Reset on a unit time event` — The QPOSCNT value is latched to the QPOSLAT register on a unit time event. The QPOSCNT register is then reset to 0 for the forward direction and QPOSMAX for the reverse direction. You can use this option for frequency measurement.

Output position counter error flag — Outputs position counter error flag on error
`off` (default) | `on`

Outputs the position counter error flag on error. When you select this option, the output port **pcef** is created.

Dependencies

This parameter appears only when, in the **Position counter** tab, you set **Position counter reset mode** to `Reset on an index event`.

Output latch position counter on index event — Latches position counter on index event
`off` (default) | `on`

When this option is enabled, the position counter QPOSCNT latches into QPOSLAT on the occurrence of an event on the strobe pin.

Dependencies

This parameter appears only when, in the **Position counter** tab, you set **Position counter reset mode** to `Reset on the maximum position` or `Reset on the first index event`.

Index event latch of position counter — Configures position counter to latch on an event
`Rising edge` (default) | `Falling edge` | `Software index marker via input port`

Configures the eQEP position counter to latch on the index event selected.

Dependencies

This parameter appears only when, in the **Position counter** tab:

- You set **Position counter reset mode** to `Reset on the maximum position` or `Reset on the first index event`.
- You select **Output latch position counter on index event**.

Output latch position counter on strobe event — Latches position counter on strobe event
`off` (default) | `on`

The eQEP strobe input can be configured to latch the position counter (QPOSCNT) into QPOSSLAT on occurrence of a definite event on this pin. This option latches the position counter on each strobe event.

Dependencies

This parameter appears only when, in the **Position counter** tab, you set **Position counter reset mode** to `Reset on the maximum position` or `Reset on the first index event`.

Strobe event of latched position counter — Configures position counter to latch on strobe
`Rising edge` (default) | `Depending on direction`

`Rising edge` latches on the rising edge of the strobe event input. `Depending on direction` latches on the rising edge in the forward direction and the falling edge in the reverse direction.

Dependencies

This parameter appears only when, in the **Position counter** tab:

- You set **Position counter reset mode** to `Reset on the maximum position` or `Reset on the first index event`.
- You select **Output latch position counter on strobe event**.

Speed calculation

To view the other parameters of this tab, select the **Enable QEP capture** option.

Enable QEP capture — Enables edge capture unit
`off` (default) | `on`

The eQEP peripheral includes an integrated edge capture unit to measure the elapsed time between the unit position events. This option enables the edge capture unit.

Output capture timer — Outputs capture timer
`off` (default) | `on`

Outputs the capture timer value from the QCTMR register.

Output capture period timer — Outputs capture period
`off` (default) | `on`

Outputs the period count value between the last successive eQEP position events from the QCPRD register.

eQEP capture timer prescaler — Prescales capture timer clock frequency
`128` (default) | `1` | `2` | `4` | `8` | `16` | `32` | `64`

The eQEP capture timer runs from prescaled SYSCLKOUT. The capture timer clock frequency is the frequency of SYSCLKOUT divided by the value you choose for this parameter.

Unit position event prescaler — Prescales quadrature clock
`128` (default) | `1` | `2` | `4` | `8` | `16` | `32` | `64` | `256` | `512` | `1024` | `2048`

The timing of the unit position event is determined by prescaling the quadrature clock (QCLK). QCLK is divided by the prescalar value you choose for this parameter.

Enable and output overflow error flag — Outputs overflow error flag when capture timer overflows
`off` (default) | `on`

Enables and outputs the eQEP overflow error flag (COEF) in the event of capture timer overflow between unit position events.

Enable and output direction change error flag — Outputs direction change error flag
`off` (default) | `on`

Enables and outputs the direction change error flag (CDEF) when direction change occurs between the unit position events.

Capture timer and position — QEP capture latch mode
`On position counter read` (default) | `On unit time-out event`

Event that triggers the latching of the capture timer and capture period register:

- `On position counter read` — Latch on position counter read by the processor. The capture timer and capture period values are latched into the QCTMRLAT and QCPRDLAT registers when the processor reads the QPOSCNT register.
- `On unit time-out event` — Latch on unit time-out. The position counter, capture timer, and capture period values are latched into the QPOSLAT, QCTMRLAT, and QCPRDLAT registers on unit time-out.

Unit timer period (0~4294967295) — Sets unit timer period
`100000000` (default) | value in the range [0, 4,294,967,295]

Set the unit timer period.

Dependencies

This parameter appears only when you set **Capture timer and position** to `On unit time-out event`.

Output capture timer latched value — Outputs capture timer latched value
`off` (default) | `on`

Outputs the capture timer latched value from the QCTMRLAT register at the output port **qctmrlat**.

Output capture timer period latched value — Outputs capture timer period latched value
`off` (default) | `on`

Outputs the capture timer period latched value from the QCPRDLAT register at the output port **qcprdlat**.

Output position counter latched value — Outputs position counter latched value
`off` (default) | `on`

Outputs the position counter latched value from the QPOSLAT register at the output port **qposlat**.

Compare output

To view the other parameters of this tab, select the **Enable position-compare sync signal output** option.

Enable position-compare sync signal output — Enables position compare sync signal output
`off` (default) | `on`

The eQEP peripheral includes a position compare unit that generates the position compare sync signal when the position counter register (QPOSCNT) and the position compare register (QPOSCMP) values match. This option enables the position compare sync signal output. The sync signal can be output using an index pin or strobe pin of the eQEP peripheral.

Sync output pin selection — GPIO pin used for sync signal
`Index pin is used for sync output` (default) | `Strobe pin is used for sync output`

The GPIO pin used for the sync signal output. Use the index pin or strobe pin of the eQEP peripheral to output the position compare sync signal.

Compare value source — Source of value for position comparison
`Specify via dialog` (default) | `Input port`

Source of the value to be used for the position compare register (QPOSCMP). When this parameter is set to `Input port` the input port **cmp** is created.

Position compare shadow load mode — Shadow mode for generating position compare sync signal
`Load on QPOSCNT=0` (default) | `Shadow disabled(load immediate)` | `Load on QPOSCNT=QPOSCMP`

This parameter lets you enable or disable shadow mode for updating the position compare (QPOSCMP) register. When shadow mode is enabled, you can also choose an event to trigger the loading of the shadow register value into the active register. When shadow mode is disabled, the processor directly loads the value into the active register.

`Load on QPOSCNT=0` loads on a position counter zero event, and `Load on QPOSCNT=QPOSCMP` loads when the QPOSCNT and QPOSCMP values match. When you select these options, shadow mode is enabled.

Position compare value (0~4294967295) — Value for comparing postiton
`4294967295` (default) | value in the range [0, 4,294,967,295]

This value is loaded into the position compare register (QPOSCMP).

Dependencies

This parameter appears only when you set **Compare value source** to `Specify via dialog`.

Sync output pulse width (1~4096) — Pulse width of position compare sync output signal
`1` (default) | value in the range [0, 4,096]

The pulse stretcher logic in the position compare unit generates a programmable position compare sync pulse output on the position compare match.

A value from `1` to `4096` that determines the pulse width of the position compare sync output signal. The width of the output pulse, measured in SYSCKOUT cycles, is four times the entered value.

Polarity of sync output — Polarity of sync output
`Active high` (default) | `Active low`

Select the polarity of the sync output signal generated.

Watchdog unit

Watchdog timer enable — Enables watchdog time-out flag
`off` (default) | `on`

The eQEP peripheral contains a watchdog timer that monitors the quadrature clock to indicate that the motion-control system is operating. The timer is reset on an edge transition of the quadrature clock. The watchdog unit generates an interrupt, which you can enable in the **Interrupt** tab.

Watchdog timer — Time-out value for watchdog timer
`65535` (default) | value in the range [0, 65,535]

The time period after which the watchdog unit generates an interrupt.

Dependencies

This parameter appears only when you select **Watchdog timer enable**.

Signal Data Types

When you select signals as output in the other tabs, the corresponding signals appear in this tab. For example, when you select the **Output position counter** option on the **Position counter** tab, the **Position counter value data type** option appears on this tab. Using this tab, you can select the data types of the signals.

The valid data types are `auto`, `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, and `boolean`.

The following table summarizes the options for which you can set the data type in the **Signal data types** tab:

Pane	Option
General	Quadrature direction flag output port
Position counter	Output position counter (selected by default)
	Output position counter error flag
	Output latch position counter on index event
	Output latch position counter on strobe event
Speed calculation	Output capture timer
	Output capture period timer
	Enable and output overflow error flag
	Enable and output direction change error flag
	Output capture timer latched value
	Output capture timer period latched value
	Output position counter latched value

Interrupt

Interrupts corresponding to specific events are enabled or disabled based on the settings in this tab. The generated interrupts are used with the C28x Hardware Interrupt.

Position counter error interrupt enable — Enables position counter error interrupts
`off` (default) | `on`

Enables position counter error interrupts. The position counter interrupt is generated only in index event reset mode. When eQEP is configured in this mode, the position counter value is latched to the QPOSILAT register, and the direction information is recorded on every index event marker. If the latched value is not equal to 0 or QPOSMAX, the position counter error interrupt is generated.

Quadrature phase error interrupt enable — Enables quadrature phase error interrupts
`off` (default) | `on`

Enables quadrature phase error interrupts. In quadrature count mode, the quadrature inputs QEPA and QEPB are expected to be 90 degrees out of phase. The quadrature phase error interrupt is generated when edge transition is detected simultaneously on the QEPA and QEPB signals.

Quadrature direction change interrupt enable — Enables quadrature direction change interrupt
`off` (default) | `on`

When the direction of counting changes, the quadrature direction change interrupt is generated.

Watchdog timeout interrupt enable — Enables watchdog timeout interrupts
`off` (default) | `on`

The eQEP peripheral contains a watchdog timer that monitors the quadrature clock. If no quadrature clock event is detected until the watchdog timer matches the watchdog period, time-out occurs and the watchdog timeout interrupt is generated.

Position counter underflow interrupt enable — Enables position counter underflow interrupts
`off` (default) | `on`

Enables position counter underflow interrupts. In the reverse direction, if the position counter reaches 0, then the position counter is reset to QPOSMAX on the next QEP clock and the position counter underflow interrupt is generated.

Position counter overflow interrupt enable — Enables position counter overflow interrupts
`off` (default) | `on`

Enables position counter overflow interrupts. In the forward direction, if the position counter reaches QPOSMAX, the position counter is reset to 0 on the next QEP clock, and the position counter overflow interrupt is generated.

Position-compare ready interrupt enable — Enables position compare ready interrupts
`off` (default) | `on`

Enables position compare ready interrupts. When the position compare register is configured for shadow mode, the position compare ready interrupt is generated after the shadow register value is loaded into the active register.

Position-compare match interrupt enable — Enables position compare match interrupts
`off` (default) | `on`

Enables position compare match interrupts. The position compare match interrupt is generated when the position counter value QPOSCNT matches with the active position compare register QPOSCMP.

Strobe event latch interrupt enable — Enables strobe event latch interrupts
`off` (default) | `on`

Enables strobe event latch interrupts. The strobe event latch interrupt is generated when the position counter is latched to the QPOSSLAT register during a strobe event latch.

Index event latch interrupt enable — Enables index event latch interrupts
`off` (default) | `on`

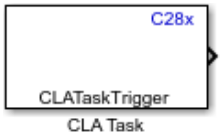
Enables index event latch interrupts. The strobe event latch interrupt is generated when the position counter is latched to the QPOSILAT register during an index event latch.

Unit timeout interrupt enable — Enables unit timeout interrupts
`off` (default) | `on`

Enables unit timeout interrupts. The unit time-out interrupt is generated when the unit timer matches the unit period.

C28x CLA Task

Create CLA task that executes downstream function-call subsystem on CLA core



Libraries:
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x

Description

The CLA Task block creates a (Control Law Accelerator) CLA task that executes a downstream function-call subsystem on the CLA core. CLA is a coprocessor that allows parallel processing. Utilizing the CLA for time-critical tasks frees up the main CPU to perform other system and communication functions concurrently.

You can specify the interrupt source to trigger the CLA task. You can create up to eight CLA tasks to execute on the CLA core.

For information about how to configure a CLA block to execute a downstream function-call subsystem, see “Overview of CLA Configuration for C2000 Processors Using Subsystem”.

Ports

Output

Port_1 — Function-call signal to a function-call subsystem or function-call model
scalar

The output triggers the CLA task that executes a downstream function-call subsystem on the CLA core.

Parameters

CLA task number — CLA task number executed on CLA core
1 (default) | integer in [1, 8]

The CLA task number that you want to execute on the CLA core.

CLA task trigger source — Source of CLA task trigger
`Software` (default) | peripheral interrupt

The software or peripheral interrupt source that triggers the CLA task to execute on the CLA core.

Wait until completion of task — Block C28x execution until CLA task is completed
`off` (default) | on

Select the parameter **Wait until completion of task**, for the software source to trigger the CLA task.

Enabling this will halt the C28x execution until all the pending CLA tasks are completed.

Dependencies

To enable this parameter, set the **CLA task trigger source** parameter as `software`.

Sample time — Frequency at which function-call subsystem is triggered
`0.2` (default) | `-1` | scalar

Set the frequency at which the function-call subsystem is triggered by the **CLA task trigger source**. To execute this block asynchronously, set this parameter to `-1`.

Dependencies

To enable this parameter, select the `Software` option in the **CLA task trigger source**.

Version History

Introduced in R2016b

See Also

CLA Subsystem | “Overview of CLA Configuration for C2000 Processors Using Subsystem”

c280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F28004x/F28002x/F28003x ePWM

Generate enhanced Pulse Width Modulated (ePWM) waveforms

Libraries:
C2000 Microcontroller Blockset / C2802x
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / C2834x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F28M35x / C28x
C2000 Microcontroller Blockset / F28M36x / C28x

Description

Configures the Type 1 to Type 4 enhanced Pulse Width Modulator (ePWM) to generate PWM waveforms. The number of available ePWM modules (ePWM1-ePWM16) vary between C2000 processors. For more information on ePWM type, refer to C2000 Real-Time Control Peripheral Reference Guide.

Use this block to generate ePWM waveforms. Multiple ePWM modules are available on C28x devices. Each module generates two PWM signals ePWMA and ePWMB.

When you enable the high-resolution pulse width modulator (HRPWM), the ePWM block uses the scale factor optimizing (SFO) software library. The SFO library can “dynamically determine the number of micro edge positioner (MEP) steps per system clock (SYSCLKOUT) period.” For more information, see `TMS320x28xx, 28xxx High-Resolution Pulse Width Modulator (HRPWM) Reference Guide`, available on the Texas Instruments web site.

This block is common to various C28x devices. Available parameters vary based on the library from which you select the block. The blue label on the top right corner of the block displays the family. As the block is a superset of functionalities available on different devices, not all parameters will be relevant to your model.

Parameters

General

Allow use of 16 HRPWMs (for C28044) instead of 6 PWMs — Enable HRPWM when PWM resolution is too low
`off` (default) | `on`

Enable all 16 high-resolution PWM modules (HRPWM) on the C28044 digital signal controller when the PWM resolution is too low.

Note Select this parameter only if you are using a C28044 processor.

For example, the Spectrum Digital eZdsp™ F28044 board has a system clock of 100 MHz (200-kHz switching). At these frequencies, conventional PWM resolution is too low, approximately 9 bits or 10 bits. By comparison, the HRPWM resolution for the same board is 14.8 bits.

Dependencies

When you enable this parameter:

- Use the HRPWM parameters under the ePWMA tab to make additional configuration changes.
- Most of the configuration parameters under the ePWMB tab are unavailable.
- Your model can contain up to 16 C280x/C2803x/C2833x ePWM blocks, provided you configure each one for a separate module. (For example, **Module** is `ePWM1`, `ePWM2`, and so on.)
- Select this parameter only if you are using a C28044 processor. To enable HRPWM for other processors, first determine how many HRPWM modules are available by consulting the Texas Instruments documentation for your processor. Then use the HRPWM parameters under the ePWMA tab to enable and configure the HRPWM.

Module — Indicates which ePWM module to use
`ePWM1` (default) | `ePWM2` | `ePWM3` | **…**

Select the appropriate ePWM module. `ePWM`*x*, where *x* can be 1,2,3....

Note Number of module available will vary for different processors.

ePWMLink TBPRD — Indicates TBPRD linking of ePWM module
`Not Linked` (default) | `ePWM2` | `ePWM3` | **…**

Select an ePWM module to which you want to link the current ePWM module for timer period. When you link the two modules, the timer period value in the linked ePWM module sets the value of the **Timer period** parameter in the current module. The **Timer period units**, **Specify timer period via** , and the **Timer period** parameters do not show when you select linking to another ePWM module.

However, the linking has no effect when you link an ePWM module to a module that does not exist in your model.

Note This parameter is available only with some TI C2000™ processors.

Timer period units — Indicates time period units
`Clock cycles` (default) | `Seconds`

Specify the units of the **Timer period** or **Timer initial period** parameters in `Clock cycles` or `Seconds`. When the parameter is set to `Seconds`, the software converts the **Timer period** or **Timer initial period** from a value in seconds to a value in clock cycles. For best results, select `Clock cycles`. Doing so reduces the number of calculations and rounding errors.

Dependencies

- If you set **Timer period units** to `Seconds`, enable support for floating-point numbers. In the model window, select **Modeling > Model Settings**.
- In the Configuration Parameters dialog box, select **Code Generation > Interface**. Under **Software Environment**, enable **floating-point numbers**.

Specify timer period via — Configure source of timer period value
`Specify via dialog` (default) | `Input port`

When you select `Input port`, the **Timer period** parameter changes to **Timer initial period** and creates a timer period input port, **T**, on the block.

Timer period — Indicates period of ePWM counter
`64000` (default)

Set the period of the ePWM counter waveform. The resultant ePWM waveform period depends on the settings of the **Action when counter=** parameters on the **ePWMx** tab.

When you enable the HRPWM, you can enter a high-precision floating-point value. The time-base period high resolution register (TBPRDHR) stores the high-resolution portion of the timer period value.

The timer period is calculated based on the **Counting mode** selection and **Timer period units**, as shown.

Count Mode	Timer period units	Calculation	Example
Up or Down	Clock cycles	The value entered in clock cycles is used to calculate time-base period (TBPRD) for the ePWM timer register. The period of the ePWM timer T_{CTR} = (TBPRD + 1) * TBCLK. Where T_{CTR} is the timer period in seconds, and TBCLK is the time-base clock.	For ePWM clock (EPWMCLK) frequency = 200 MHz and TBCLK = 5 ns. EPWMCLK will be equal to SYSCLKOUT or SYSCLKOUT/2 depending on the **EPWM clock divider (EPWMCLKDIV)** parameter setting. When the timer period is entered in clock cycles TBPRD = 9999, and the ePWM timer period is calculated as T_{CTR} = 50 µs. For the default action settings on the **ePWMx** tab, the ePWM period = 50 µs.
	Seconds	The value entered in seconds is used to calculate the time-base period (TBPRD) for the ePWM timer register. The TBPRD value entered in the register is TBPRD = (T_{CTR} / TBCLK) – 1. Where, T_{CTR} is the timer period in seconds and TBCLK is the time-base clock. For the default action settings on the **ePWMx** tab, the ePWM period is the same as the timer period (in seconds) entered.	For EPWMCLK frequency = 200 MHz and TBCLK = 5 ns. When the timer period is entered in seconds T_{CTR} = 50 µs and the time based period is calculated as TBPRD = 9999. For the default action settings in the **ePWMx** tab, the ePWM period = 50 µs.
Up-Down	Clock cycles	The value entered in clock cycles is used to calculate the time-base period (TBPRD) for the ePWM timer register. The period of the ePWM timer, T_{CTR} = 2 * TBPRD * TBCLK. Where T_{CTR} is the timer period in seconds and TBCLK is the time-base clock.	For EPWMCLK frequency = 200 MHz and TBCLK = 5 ns. When the timer period is entered in clock cycles, TBPRD = 10000, and the ePWM timer period is calculated as T_{CTR} = 100 µs. For the default action settings on the **ePWMx** tab, the ePWM period = 100 µs.

Count Mode	Timer period units	Calculation	Example
	Seconds	The value entered in seconds is used to calculate the time-base period (TBPRD) for the ePWM timer register. The TBPRD value entered in the register is TBPRD = T_{CTR} / TBCLK. Where T_{CTR} is the timer period in seconds and TBCLK is the time-base clock. For the default action settings on the **ePWMx** tab, the ePWM period is two times the timer period (in seconds) entered.	For EPWMCLK frequency = 200 MHz and TBCLK = 5 ns. When the timer period is entered in seconds T_{CTR} = 50 µs, and the time based period is calculated as TBPRD = 10000. For the default action settings on the **ePWMx** tab, the ePWM period = 100 µs.

Timer initial period — Indicates initial ePWM period of the waveform
`64000` (default) | `0` | `...`

The initial period of the waveform from the time the PWM peripheral starts operation until the ePWM input port, **T**, receives a new value for the period. Use **Timer period units** to measure the period in clock cycles or in seconds. The timer period is calculated similar to the **Timer period** parameter.

Dependencies

To enable this parameter, set the **Specify timer period via** parameter to `Input port`.

Reload for time base period register (PRDLD) — Event at which counter period register is reloaded
`Counter equals to zero` (default) | `Counter equals to zero or SYNC event` | `SYNC event` | `Immediate without using shadow`

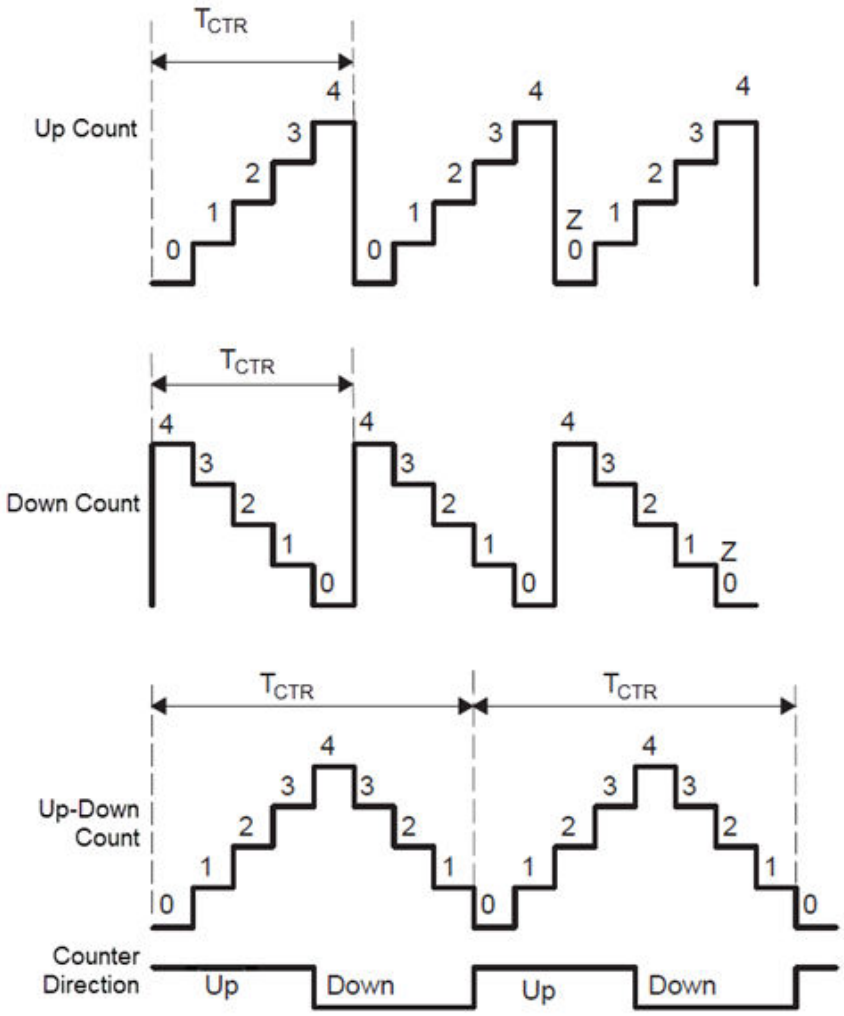
This parameter provides an option to select the appropriate event to update the time period register with a new value.

- `Counter equals to zero` - The counter period refreshes when the value of the counter is 0.
- `Counter equals to zero or SYNC event` - When counter is 0 or when there is a synchronization event.
- `SYNC event` - The parameter is in synchronization.
- `Immediate without using shadow` - The counter period refreshes immediately.

Counting mode — Indicates counting mode of ePWM counter
`Up-Down` (default) | `Up` | `Down`

Specify the counting mode. This PWM module can operate in three distinct counting modes: `Up`, `Down`, and `Up-Down`.

This illustration shows the waveforms that correspond to these three modes:



Dependencies

The Down option is not compatible with HRPWM. To avoid an error when you build the model, do not set the **Counting mode** parameter to Down and select the **Enable HRPWM (Period)** parameter.

Synchronization action — Specify source of phase offset
`Disable` (default) | `Set counter to phase value specified via dialog` | `Set counter to phase value specified via input port`

Specify the source of a phase offset to apply to the time-base synchronization input signal EPWMxSYNCI from the **SYNC** input port.

- `Set counter to phase value specified via dialog` - Specify this option to create the **Phase offset value** parameter.

Note

- The ePWM block expects the phase value in the range of 0 to 360 deg values scaled with clock cycles as input to the phase values. Hence for the negative values you need to add 360 deg to

bring it to the range of 0 to 360 deg phase. Example, for -90 deg value, you will have to provide (360 - 90) = 270 deg scaled with clock cycles as input.

- For any values greater than 360 deg you need to consider the logic to wrap the value to the scale of 0 to 360 deg. For example, 120 + (360 - 90) = 390 should be wrapped as (390 - 360) = 30 deg before scaling it with clock cycles and providing it as input to ePWM.

- `Set counter to phase value specified via input port` - Specify this option to create a phase input port, **PHS**, on the block.
- `Disable` - Specify this option to prevent the application of phase offsets to the TB module.

Counting direction after phase synchronization — Counter direction after phase synchronization
`Count down after sync` (default) | `Count up after sync`

Configure the timer to count up or down, following synchronization. This parameter corresponds to the phase direction (PHSDIR) field of the Time-base Control Register (TBCTL).

Dependencies

This parameter appears when **Counting mode** is `Up-Down` and **Synchronization action** is `Set counter to phase value specified via dialog` or `Input port`.

Phase offset value (TBPHS) — Specify Phase offset value (TBPHS)
`0` (default)

The specified offset value is loaded in the time base counter on a synchronization event.

To enable this parameter, select the `Set counter to phase value specified via dialog` in the **Synchronization action** parameter.

Enter the `Phase offset value (TBPHS)` in **TBCLK** cycles from 0 to 65535. While using HRPWM, you may enter decimal values.

Note

- The ePWM block expects the phase value to be in the range of 0 to 360 degrees scaled with clock cycles. For the negative values you need to add 360 degrees to bring it to the range of 0 to 360 degrees phase range. For example, for -90 degrees value, you will have to provide (360 - 90) = 270 degrees scaled with clock cycles as an input.
- For any values greater than 360 degrees you need to consider the logic to wrap the value to the scale of 0 to 360 degrees. For example, 120 + (360 - 90) = 390 should be wrapped as (390 - 360) = 30 degrees before scaling it with clock cycles and providing it as an input.

This parameter corresponds to the Time-Base Phase Register (TBPHS).

Specify software synchronization via input port (SWFSYNC) — Indicates phase offset value
`off` (default) | `on`

Create an input port, **SYNC**, for a time-base synchronization input signal, EPWMxSYNCI. Select this parameter to achieve precise synchronization across multiple ePWM modules by daisy-chaining multiple time-base (TB) submodules.

Enable digital compare A event1 synchronization (DCAEVT1) — Synchronize ePWM time based on DCAEVT1 digital compare event
`off` (default) | `on`

Select this parameter to synchronize this PWM module to the time base of another PWM module. Fine-tune the synchronization between the two modules using the **Phase offset value**. Enabling HRPWM disables this option.

Note This parameter is available only for specific `C28x` devices.

Enable digital compare B event1 synchronization (DCBEVT1) — Synchronize ePWM time based on DCBEVT1 digital compare event
`off` (default) | `on`

Select this parameter to synchronize this PWM module to the time base of another PWM module. Fine-tune the synchronization between the two modules using the **Phase offset value**. This option is not compatible with `HRPWM`. Enabling `HRPWM` disables this option.

Note This parameter is available only for specific `C28x` devices.

Synchronization output (SYNCO) — Event which generates a time base synchronization output signal
`Disable` (default) | `Pass through (EPWMxSYNCI or SWFSYNC)` | `Counter equals to zero (CTR=Zero)` | `Counter equals to compare B (CTR=CMPB)` | `Counter equals to compare C (CTR=CMPC)` | `Counter equals to compare D (CTR=CMPD)` | `Enable the DCBEVT1 sync event to the SYNCO signal` | `Enable the DCAEVT1 sync event to the SYNCO signal` | `Enable the counter equals to compare D (CTR=CMPD) event to set the SYNCO signal` | `Enable the counter equals to compare C (CTR=CMPC) event to set the SYNCO signal` | `Enable the counter equals to compare B (CTR=CMPB) event to set the SYNCO signal` | `Enable the counter equals to zero (CTR=ZERO) event to set the SYNCO signal` | `Enable the SWFSYNC event to set the SYNCO signal`

Time-base counter synchronization allows for increased flexibility of synchronization of the ePWM modules. The clock synchronization scheme allows ePWM modules to operate as a single system when required. Additionally, this synchronization scheme can be extended to the capture peripheral submodules (eCAP). In Type 4 ePWM, there are two types of time-base counter synchronization scheme available. For more information, see “Time-Base Counter Synchronization”.

You can configure additional SYNCO options in Configuration Parameters. For more, see “C28x-ePWM” on page 1-145

This parameter corresponds to the SYNCOSEL field in the Time-Base Control Register (TBCTL).

Note The parameters for synchronization output (SYNCO) vary based on the processor selected.

The following parameters are available only for specific `C28x` devices.

- `Pass through (EPWMxSYNCI or SWFSYNC)` — synchronization input pulse or software-forced synchronization pulse, respectively. You can use this option to achieve precise synchronization across multiple ePWM modules by daisy-chaining multiple time-base (TB) submodules.
- `Counter equals to zero (CTR=Zero)` — Time-base counter equal to zero (TBCTR = 0x0000)
- `Counter equals to compare B (CTR=CMPB)` — Time-base counter equal to counter-compare B (TBCTR = CMPB)
- `Counter equals to compare C (CTR=CMPC)` — Time-base counter equal to counter-compare C (TBCTR = CMPC)
- `Counter equals to compare D (CTR=CMPD)` — Time-base counter equal to counter-compare D (TBCTR = CMPD)
- `Disable` — Disable the EPWMxSYNCO output (the default)

The following parameters are only specific to F2838x, F28002x and F28003x processors.

- `Enable the DCBEVT1 sync event to the SYNCO signal` - Select to synchronize digital compare event B to the synchronization output.
- `Enable the DCAEVT1 sync event to the SYNCO signal` - Select to synchronize digital compare event A to the synchronization output.
- `Enable the counter equals to compare D (CTR=CMPD) event to set the SYNCO signal` - enables the counter equals to compare D (CTR=CMPD) event to set the SYNCO signal
- `Enable the counter equals to compare C (CTR=CMPC) event to set the SYNCO signal` - enables the counter equals to compare C (CTR=CMPC) event to set the SYNCO signal
- `Enable the counter equals to compare B (CTR=CMPB) event to set the SYNCO signal` - enables the counter equals to compare B (CTR=CMPB) event to set the SYNCO signal
- `Enable the counter equals to zero (CTR=ZERO) event to set the SYNCO signal` - enables the counter equals to zero (CTR=ZERO) event to set the SYNCO signal
- `Enable the SWFSYNC event to set the SYNCO signal` - enables the software-forced synchronization pulse (SWFSYNC) event to set the SYNCO signal

Peripheral synchronization event (PWMSYNCSEL) — Time-Base Counter Peripheral Synchronization
`Counter equals to period (CTR=PRD)` (default) | `Counter equals to zero (CTR=Zero)` | `Counter is incrementing and equals to compare C register (CTRU=CMPC)` | `Counter is decrementing and equals to compare C register (CTRD=CMPC)` | `Counter is incrementing and equals to compare D register (CTRU=CMPD)` | `Counter is decrementing and equals to compare D register (CTRD=CMPD)`

Each ePWM module has a peripheral synchronization output (SYNCPER). This output signal is used to synchronize the CMPSS to the EPWM. For more information, see “Time-Base Counter Peripheral Synchronization”

Time base clock (TBCLK) prescaler divider — Time base clock (TBCLK) prescaler divider corresponding to CLKDIV
1 (default) | 2 | 4 | **…**

Use the **Time base clock (TBCLK) prescaler divider** (CLKDIV) and the **High speed time base clock (HSPCLKDIV) prescaler divider** (HSPCLKDIV) to configure the Time-base clock speed (TBCLK) for the ePWM module. Calculate TBCLK using this equation:

TBCLK in Hz = PWM clock in Hz/(HSPCLKDIV * CLKDIV)

For example, the default values of both CLKDIV and HSPCLKDIV are 1, and the default frequency of PWM clock is 100 MHz, so:

TBCLK in Hz = 100 MHz/(1 * 1) = 100 MHz

TBCLK in seconds = 1/TBCLK in Hz = 1/100 MHz = 0.01 µs

The choices for the **Time base clock (TBCLK) prescaler divider** are: 1, 2, 4, 8, 16, 32, 64, and 128.

The **Time block clock (TBCLK) prescaler divider** parameter corresponds to the CLKDIV field of the Time-base Control Register (TBCTL).

The PWM clock is the SYSCLKOUT or a clock derived from SYSCLKOUT using the PWM clock divider. For a few TI C2000 processors, there is a PWM clock divider that divides the SYSCLKOUT to derive the PWM module clock. Check the technical reference manual of your processor for more details.

The frequency of SYSCLKOUT depends on the oscillator frequency and the configuration of the PLL-based clock module. Changing the value of SYSCLOCKOUT affects the timing of all the ePWM modules. If there is a PWM clock prescale available in the processor, changing its value also affects the PWM timing.

High speed time base clock (HSPCLKDIV) prescaler divider — High speed time base clock (HSPCLKDIV) prescaler divider
1 (default) | 2 | 4 | ...

The choices for the **High speed time base clock (HSPCLKDIV) prescaler divider** are: 1, 2, 4, 6, 8, 10, 12, and 14.

Selecting **Enable high resolution PWM (HRPWM – period)** sets the value of this parameter to 1.

This parameter corresponds to the HSPCLKDIV field of the Time-base Control Register (TBCTL).

Enable swap module A and B — Swap output signals for module A and B
off (default) | on

Swap the ePWMA and ePWMB outputs. When you select this parameter, the block outputs the ePWMA signals on the ePWMB outputs and the ePWMB signals on the ePWMA outputs.

This parameter, sets the SWAPB field in the HRPWM Configuration Register (HRCNFG). This option works in for ePWMs that support HRPMW as it uses the HRCNFG register.

Note This parameter is available only for specific C28x devices.

ePWMA and ePWMB

You can perform waveform generation configuration in ePWMA/ePWMB tab along with Counter compare tab.

The ePWM hardware module can generate waveforms for A and B channels followed by Deadband module and HRPWM module for waveform generation.

During waveform generation you can generate waveform for ePWMA and ePWMB but these waveform options can be overridden by the options selected in Deadband configuration. For example,

you can choose not to configure the eWPMB in waveform generation module but use eWPMA waveform to create Deadband outputs for both ePWMA and ePWMB channels.

Enable ePWM#x — Enables ePWMx output signals
`off` (default) | `on`

Enables the ePWMA and/or ePWMB output signals for the ePWM module selected on the **General** tab. In this case, # represents the ePWM module and x represents A or B. By default, **Enable ePWM#A** is enabled, and **Enable ePWM#B** is disabled.

Each ePWM module has two outputs, ePWMA and ePWMB. The ePWMA output tab and ePWMB output tab include the same settings, although the default values vary in some cases.

Dependencies

When you select **Enable ePWM#A** or **Enable ePWM#B**, enable support for floating-point numbers by browsing to **Configuration Parameters** > **Code Generation** > **Interface** > **Software Environment**.

CMPx initial value — Initial value of CMPx
32000 (default)

This field appears when you set **CMPx source** to `Input port`. In this case, x represents A or B. Enter the initial pulse width of CMPA or CMPB that the PWM peripheral uses when it starts operation. Subsequent inputs to the **WA** or **WB** ports change the CMPA or CMPB pulse width.

Action when counter=ZERO, Action when counter=ZERO, Action when counter=CMPA on up-count (CAU), Action when counter=CMPA on down-count (CAD), Action when counter=CMPB on up-count (CBU), Action when counter=CMPB on down-count (CBD) — Determine Action Qualifier (AQ) submodule behavior
`Set` (default)

This group of parameters on the **ePWMA output** and **ePWMB output** tabs determine the behavior of the Action Qualifier (AQ) submodule. The AQ module determines which events are converted into one of the various action types, producing the required switched waveforms of the ePWM#A and ePWM#B output signals.

These parameters can be specified as `Do nothing`, `Clear`, `Set`, or `Toggle`.

The default values vary between the **ePWMA** and **ePWMB** tabs.

This table shows the default value for each of these tabs when you set the **Counting mode** to `Up` or `Up-Down`:

Action when counter =	ePWMA	ePWMB
ZERO	Set	Clear
period (PRD)	Clear	Set
CMPA on up-count (CAU)	Clear	Set
CMPA on down-count (CAD)	Set	Do nothing
CMPB on up-count (CBU)	Do nothing	Clear
CMPB on down-count (CBD)	Do nothing	Set

This table shows the defaults for each of these tabs when you set **Counting mode** to `Down`:

Action when counter =	ePWMA	ePWMB
ZERO	Do nothing	Do nothing
period (PRD)	Clear	Clear
CMPA on down-count (CAD)	Set	Do nothing
CMPB on down-count (CBD)	Do nothing	Set

For a detailed discussion on the AQ submodule, consult the *TMS320x280x Enhanced Pulse Width Modulator (ePWM) Module Reference Guide* (SPRU791), available on the Texas Instruments website.

Compare value reload condition, Add continuous software force input port, Continuous software force initial logic, Reload condition for software force — Determines how action-qualifier (AQ) submodule handles S/W force event
`Load on counter equals to zero (CTR=Zero)` (default)

These parameters determine how the action-qualifier (AQ) submodule handles the S/W force event, an asynchronous event initiated by software (CPU) via control register bits.

Compare value reload condition determines if and when to reload the action-qualifier S/W force register from a shadow register. You can specify one of the following `Load on counter equals to zero (CTR=Zero)` (default), `Load on counter equals to period (CTR=PRD)`, `Load on either`, and `Freeze`.

Add continuous software force input port creates an input port, **SFA**, which you can use to control the software force initial logic. Send one of the following values to **SFA** as an unsigned integer data type:

- 0 = `Forcing disable`: Do nothing(default).
- 1 = `Forcing low`: Clear low
- 2 = `Forcing high`: Set high

Continuous software force initial logic - If you did not create the **SFA** input port, you can use to select which type of software force initial logic to apply. You can specify one of the following:

- `Forcing disable`: Do nothing (default).
- `Forcing low`: Clear low
- `Forcing high`: Set high

Reload condition for software force

You can specify one of the following `Counter equals to zero (CTR=Zero)` (default), `Counter equals to period (CTR=PRD)`, `Either period or zero`, and `Immediate`.

Inverted version of ePWM#A — Invert ePWM#A signal
`off` (default) | `on`

Invert the ePWM#A signal and output it on the ePWM#B outputs.

This parameter sets the SELOUTB field in the HRPWM Configuration Register (HRCNFG) and will work for ePWMs with HRPWM support as it uses the HRCNFG register.

Note This parameter is available only for ePWM modules which have HRPWM submodule.

Enable high resolution PWM (HRPWM) — Enables high resolution PWM settings
`off` (default) | `on`

Dependencies

- This parameter is available only for **ePWMx** tabs for C280x and C2833x processors and in **HRPWM** tab for the rest of the processors. For more information, refer to HRPWM tab section.
- Similarly, parameters such as **High resolution PWM (HRPWM) loading mode**, **High resolution PWM (HRPWM) control mode**, **High resolution PWM (HRPWM) edge control mode** and **Use scale factor optimizer (SFO) software** feature in **ePWMx** tabs for C280x and C2833x processors and in **HRPWM** tab for rest of the processors. For more information, refer to HRPWM tab section.

Counter Compare

ePWMLink CMPx — Linking of ePWM module
`Not Linked` (default) | `ePWM1` | `ePWM2` | `...`

In ePWMx, where x ranges from 1 to 12

Select an ePWM module to which you want to link the current ePWM module for counter value. In this case for CMPx, x represents A, B, C, or D. When you link the counter value of an ePWM module with another, the **CMPx value** of the linked ePWM module is used in the current module. The **CMPx**, **Specify CMPx via**, and **CMPx value** parameters do not appear when you select another ePWM module for linking.

However, the linking has no effect when you link an ePWM module to a module that does not exist in your model.

Note This parameter is available only with some of the TI's C2000 processors.

CMPx Units — Units used by compare registers
`Clock cycles` (default) | `Percentages`

The four compare registers CMPA, CMPB, CMPC, and CMPD are compared with the time-base counter value to generate appropriate events. CMPA and CMPB events are used for controlling the PWM duty cycle by selecting appropriate actions on the **ePWMA** and **ePWMB** tabs. These events can also be used to generate an interrupt to the CPU and/or a start of conversion pulse to the ADC. You can refer to the **Event Trigger** tab to select the events to be triggered.

Compare registers **CMPC** and **CMPD** are present only in the latest processors.

Notes

- The term *clock cycles* refers to the time-base clock on the processor. See the **TB clock prescaler divider** topic for an explanation of time-base clock speed calculations.
- Percentages use additional computation time in generated code and can decrease accuracy of the results.

Dependencies

If you set **CMPx units** to `Percentages`, enable support for floating-point numbers by browsing to **Configuration Parameters** > **Code Generation** > **Interface** > **Software Environment**. In this case, x represents A, B, C, or D.

Specify CMPx Via — Pulse width source
`Specify via dialog` (default) | `Input port`

Specify the source of the pulse width. If you select `Specify via dialog` (the default), enter a value for the **CMPx value** parameter. If you select `Input port`, the block creates a new Input port with name WA, WB, WC or WD on the block. If you select `Input port`, make sure to set the **CMPx initial value** parameter. In this case, x represents A, B, C, or D.

CMPx Value/CMPx Initial Value — Pulse width value
`32000` (default)

CMPx Value field appears when you set **CMPx source** to `Specify via dialog`. **CMPx Initial Value** field appears when you set **CMPx source** to `Input port`. Enter a value that specifies the pulse width, in the units specified in **CMPx units**. In this case, x represents A, B, C, or D.

Reload for compare x Register (SHDWxMODE) — Time at which the counter period is reset
`Counter equals to zero (CTR=Zero)` (default) | `Counter equals to period (CTR=PRD)` | `Counter equals to Zero or period (CTR=Zero or CTR=PRD)` | `...`

The time at which the counter period is reset. In this case, x represents A, B, C, or D.

- `Counter equals to zero (CTR=Zero)` — Refreshes the counter period when the value of the counter is 0.
- `Counter equals to period (CTR=PRD)` — Refreshes the counter period when the value of the counter is period.
- `Counter equals to zero or period (CTR=Zero or CTR=PRD)` — Refreshes the counter period when the value of the counter is 0 or period.
- `Freeze` — Refreshes the counter period when the value of the counter is freeze.
- `Counter equals to zero or SYNC event` — Refreshes the counter period when the value of the counter is zero or SYNC.
- `Counter equals to period or SYNC event` — Refreshes the counter period when the value of the counter is 0 or SYNC.
- `Counter equals to zero or period or SYNC event` — Refreshes the counter period when the value of the counter is 0 or period or SYNC.
- `SYNC event` — Refreshes the counter period when the value of the counter is SYNC.
- `Immediate without using shadow` — Refreshes the counter period immediately.

Deadband Unit

Use deadband for ePWM#A, Use deadband for ePWM#B — Enables deadband area
`off` (default) | `on`

Enables a deadband area of rising edge delay or falling edge delay cycles without signal overlap between pairs of ePWM output signals. Ensure that the corresponding **ePWMx (x=A or B)** are enabled on the ePWMx tabs to generate the DB for the same.

Enable half-cycle clocking — Double deadband resolution
`off` (default) | `on`

Enable half-cycle clocking to double the deadband resolution. This option clocks the deadband counters at TBCLK*2 I.e it will divide the clock value in seconds by 2, which in turn doubles the frequency.

When you disable this option, the deadband counters use full-cycle clocking (TBCLK*1).

Note This parameter is available only specific C28x devices.

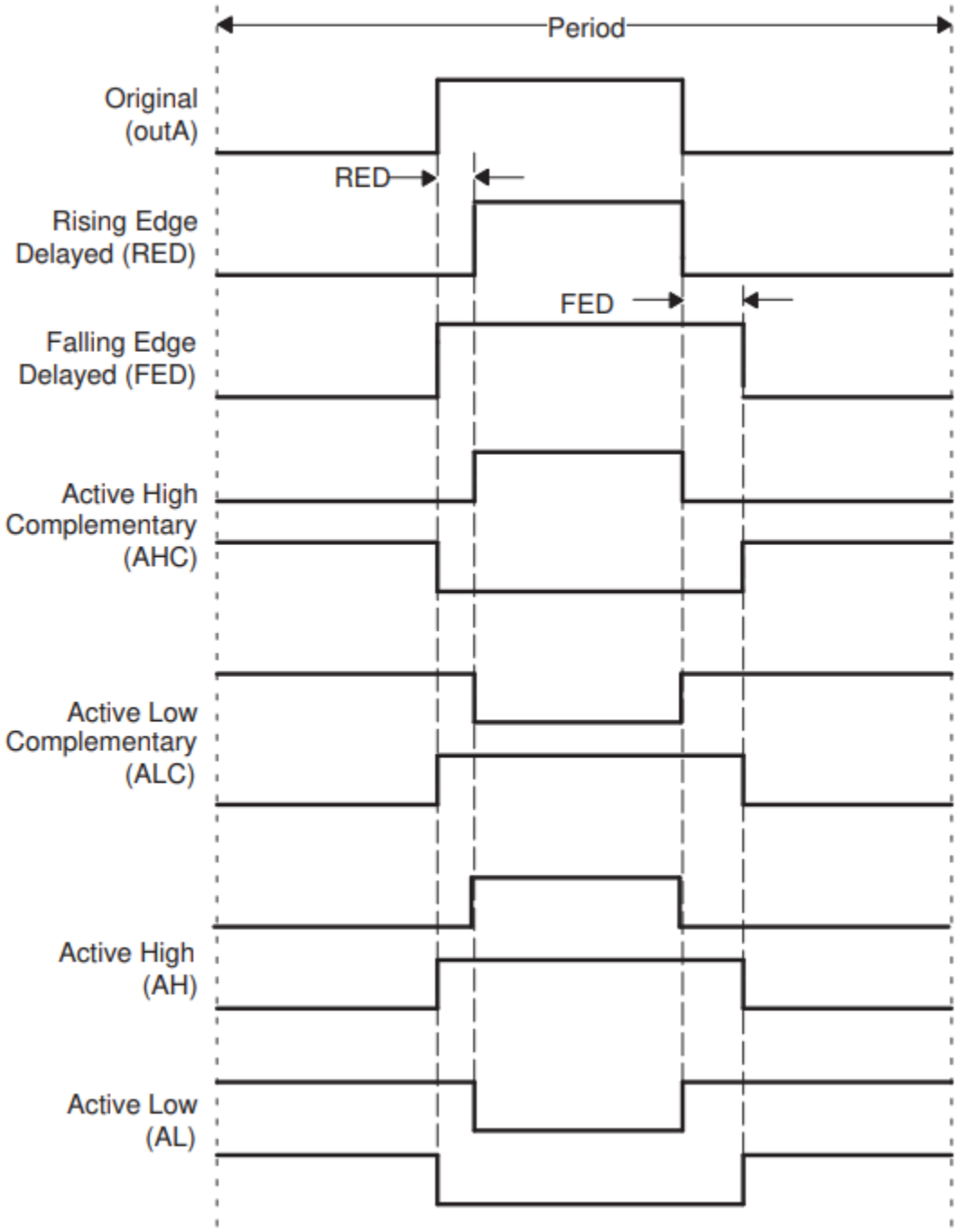
Deadband Polarity — ePWM output state at deadband time
`Active high (AH)` (default) | `Active low (AL)` | `Active high complementary (AHC)` | `Active low complementary (ALC)`

When you enable only one ePWM for deadband polarity, you will see the options `Positive` and `Negative`.

When you enable both the ePWM for deadband polarity, then you will see the options `Active high (AH)`, `Active low (AL)` , `Active high complementary (AHC)` or `Active low complementary (ALC)`.

AT the deadband time, both the ePWMA and ePWMB outputs have to set to an inactive state. Depending on your hardware settings, the inactive states can correspond to a high or a low logic value. Active high means that the system is active when the ePWM output is set to a high logic value. Active low means that the system is active when the ePWM output is set to a low logic value. Use the Complementary option when the B signal needs to be the inverse of A. For more information, refer to the ePWM technical reference guide of your processor.

This diagram shows the waveforms for typical cases where 0% < duty < 100%.



Signal source for rising edge (RED) — Raising edge source
ePWMxA (default) | ePWMxB

Select the signal source to which rising edge delay (RED) has to be applied. You can either select ePWMxA or ePWMxB for raising edge.

Signal source for falling edge (FED) — Falling edge source
`ePWMxA` (default) | `ePWMxB`

Select the signal source to which falling edge delay (FED) has to be applied. You can either select ePWMxA or ePWMxB for falling edge.

Deadband period units — Indicates deadband period units
`Clock cycles` (default) | `Seconds`

Specify the units of the **Deadband period** as `Clock cycles` or `Seconds`. When **Deadband period units** is set to `Seconds`, the software converts the **Deadband period** from a value in seconds to a value in clock cycles. For best results, select `Clock cycles`. Doing so reduces calculations and rounding errors.

Dependencies

- If you set **Deadband period units** to `Seconds`, enable support for floating-point numbers. In the model window, select **Simulation > Model Configuration Parameters**.

  In the Configuration Parameters dialog box, select Code Generation > Interface. Under **Software Environment**, enable **floating-point numbers**.
- This parameter is available only with some of the TI's C2000™ processors.

Deadband period source — Source of control logic
`Specify via dialog` (default) | `Input port`

Select the source of the **Deadband period** as `Specify via dialog` or `Input port`.

Deadband Rising edge (RED) period (0-16383) — RED delay time value
`0` (default)

The value you enter in the field specifies the deadband delay in time-base clock.

The outcome of the **Deadband rising edge period (0-16383)** parameter depends on the `Deadband period units` and `Deadband period source`.

The following table illustrates the deadband rising edge period result based on the selection of `deadband unit`, `deadband source` and the `processor` selected.

Deadband period units	Deadband period source	Deadband rising edge(RED) period
Clock cycles	Specify via dialog	Deadband Rising edge (RED) period (0~16383)
		Deadband Rising edge (RED) period (0~1023)
Clock cycles	Input port	Deadband Rising edge (RED) initial period (0~16383)
		Deadband Rising edge (RED) initial period (0~1023)
Seconds	Specify via dialog	Deadband Rising edge (RED) period
		Deadband Rising edge (RED) period

Deadband period units	Deadband period source	Deadband rising edge(RED) period
Seconds	Input port	Deadband Rising edge (RED) initial period: Deadband Rising edge (RED) initial period:

Deadband falling edge (FED) period — FED delay time value
`0` (default)

The value you enter in the field specifies the dead band delay in time-base clock.

The parameter **Deadband falling edge period** outcome depends on the `Deadband period units` and `Deadband period source`. The **Deadband falling edge period** values are similar to **Deadband rising edge period**. For more information refer to deadband rising edge (RED) period parameter.

Event Trigger

Enable ADC start of conversion for module A — Start of ADC conversion event
`off` (default) | `on`

When you select this option, ADC Start of Conversion Event (ePWMSOCxA) is generated when the event selected in the **Start of conversion for module A event selection** parameter occurs.

Number of event for start of conversion for Module A (SOCA) to be generated — Event start number
`First event` (default) | `Second event` | `Third event` | `...`

When you select **Enable ADC start of conversion for module A**, this field specifies the number of the event that triggers ADC Start of Conversion for Module A (SOCA).

`First event` triggers ADC start of conversion with every event (the default). `Second event` triggers ADC start of conversion with every second event. `Third event` triggers ADC start of conversion with every third event.

Event triggers ranges from `First event` to `Fifteenth event` based on the processor selected.

Start of conversion for module A event selection — Indicates counter match condition that triggers ADC start of conversion event
`Counter equals to zero (CTR=Zero)` (default) | `Digital Compare Module A Event 1 start of conversion (DCAEVT1.soc)` | `...`

This parameter specifies the counter match condition that triggers an ADC start of conversion event. The choices are:

- `Digital Compare Module A Event 1 start of conversion (DCAEVT1.soc)(For specific C28x devices only)` - When the ePWM asserts a DCAEVT1 or DCBEVT1 digital compare event. Use this feature to synchronize the selected PWM module to the time base of another PWM module. Fine-tune the synchronization between the two modules using the **Phase offset value** parameter.
- `Counter equals to zero (CTR=Zero)` - When the ePWM counter reaches zero (the default for few processors).

- `Counter equals to period (CTR=PRD)` - When the ePWM counter reaches the period value.
- `Counter equals to zero or period (CTR=Zero or CTR=PRD)` - When the time base counter reaches zero (TBCTR = 0x0000) or when the time base counter reaches the period (TBCTR = TBPRD).
- `Counter is incrementing and equals to the compare x register (CTRU=CMPx)` - The ePWM counter reaches the compare value x on the way up. In this case, x represents A, B, C, or D.
- `Counter is decrementing and equals to the compare x register (CTRD=CMPx)` - The ePWM counter reaches the compare value x on the way down. In this case, x represents A, B, C, or D.

Enable ADC start of conversion for module B — Start of ADC conversion event
`off` (default) | `on`

When you select this option, ADC Start of Conversion Event (ePWMSOCxB) is generated when the event selected in the **Start of conversion for module B event selection** parameter occurs.

Number of event for start of conversion for Module B (SOCB) to be generated — Event start number
`First event` (default) | `Second event` | `Third event` | `...`

When you select **Enable ADC start of conversion for module B**, this field specifies the number of the event that triggers ADC Start of Conversion for Module B (SOCB).

`First event` triggers ADC start of conversion with every event (the default). `Second event` triggers ADC start of conversion with every second event. `Third event` triggers ADC start of conversion with every third event.

Event triggers ranges from `First event` to `Fifteenth event` based on the processor selected.

Start of conversion for module B event selection — Indicates the counter match condition that triggers an ADC start of conversion event
`Counter equals to zero (CTR=Zero)` (default) |

When you select **Enable ADC start of conversion for module B**, this field specifies the counter match condition that triggers an ADC start of conversion event. The choices are:

- `Digital Compare Module B Event 1 start of conversion (DCBEVT1.soc) (For specific C28x devices only)` - When the ePWM asserts a DCAEVT1 or DCBEVT1 digital compare event. Use this feature to synchronize the selected PWM module to the time base of another PWM module. Fine-tune the synchronization between the two modules using the **Phase offset value**.
- `Counter equals to zero (CTR=Zero)` - When the ePWM counter reaches zero (the default for few processors).
- `Counter equals to period (CTR=PRD)` - When the ePWM counter reaches the period value.
- `Counter equals to zero or period (CTR=Zero or CTR=PRD)` - When the time base counter reaches zero (TBCTR = 0x0000) or when the time base counter reaches the period (TBCTR = TBPRD).
- `Counter is incrementing and equals to the compare x register (CTRU=CMPx)` - The ePWM counter reaches the compare value x on the way up. In this case, x represents A, B, C, or D.

- `Counter is decrementing and equals to the compare x register (CTRD=CMPx)` - The ePWM counter reaches the compare value x on the way down. In this case, x represents A, B, C, or D.

Enable ePWM interrupt — Generates ePWM interrupts
`Disabled` (default) | `Enabled`

Select this option to generate ePWM interrupts based on different events defined by **Number of event for interrupt to be generated** and **Interrupt counter match event condition**. By default, the software clears (disables) this option.

Number of event for interrupt to be generated — Specifies the number of the event that triggers the ePWM interrupt
`First event` (default) | `Second event to fifteenth event`

When you select **Enable ePWM interrupt**, this field specifies the number of the event that triggers the ePWM interrupt: `First event` triggers ePWM interrupt with every event (the default), `Second event` triggers ePWM interrupt with every second event, and `Third event` triggers ePWM interrupt with every third event.

Note Event triggers ranges from `First event` to `Fifteenth event` based on the processor selected.

Interrupt counter match event condition — Indicates the counter match condition that triggers ePWM interrupt
`Counter equals to zero (CTR=Zero)` (default)

When you select **Enable ePWM interrupt**, this field specifies the counter match condition that triggers ePWM interrupt. The choices are the same as for **Module A counter match event condition**. The choices are:

- `Counter equals to zero (CTR=Zero)` - When the ePWM counter reaches zero (the default for few processors).
- `Counter equals to period (CTR=PRD)` - When the ePWM counter reaches the period value.
- `Counter equals to zero or period (CTR=Zero or CTR=PRD)` - When the time base counter reaches zero (TBCTR = 0x0000) or when the time base counter reaches the period (TBCTR = TBPRD).
- `Counter is incrementing and equals to the compare x register (CTRU=CMPx)` - The ePWM counter reaches the compare value x on the way up. In this case, x represents A, B, C, or D.
- `Counter is decrementing and equals to the compare x register (CTRD=CMPx)` - The ePWM counter reaches the compare value x on the way down. In this case, x represents A, B, C, or D.

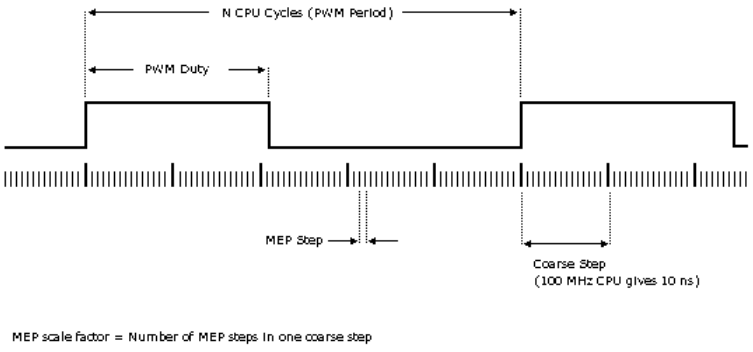
HRPWM Tab

During HR (High Resolution) configuration for F2837xD processor, the ePWM modules only exists on ePWM1 and SFO only runs on CPU1. Therefore if you want to use HRPWM, then ePWM1 must be allocated to CPU1 and if SFO is enabled, it must run on CPU1.

For F2838xD device, even though CPU and EWPM1 still do the calibration for HRPWM, each EPWM module clocks its own HRPWM, and use the same values of HRMSTEP calculated by the SFO. This allows HRPWM to function correctly with CPU2.

Enable high resolution period on ePWM#A (HRPWM - period), Enable high resolution period on ePWM#B (HRPWM - period) — Indicates the effective resolution for conventionally generated PWM
`off` (default) | `on`

When the effective resolution for the conventionally generated PWM is insufficient, consider using High Resolution PWM (HRPWM). The resolution of PWM is normally dependent upon the PWM frequency and the underlying system clock frequency. To address this limitation, HRPWM uses **Micro Edge Positioner (MEP)** technology to position edges more finely by dividing each coarse system clock. The accuracy of the subdivision is on the order of `150ps`. The following figure shows the relationship between one system clock and edge position in terms of **MEP** steps:



When this parameter is enabled, the block accepts decimal values for the timer period of the ePWM Module. The Extension Register for the HRPWM Period (TBPRDHR) provides an 8 bit representation of the decimal part of the **Timer period** value. This parameter enables the **Enable high resolution PWM (HRPWM - duty)** parameter, and displays the **HRPWM loading mode**, **HRPWM control mode**, and **HRPWM edge control** mode parameters.

Selecting Enable HRPWM (Period) forces **TB clock prescaler divider** and **High Speed TB clock prescaler divider** to `1`. These settings match the HRPWM time base clock with the SYSCLKOUT frequency.

The `Down` option in the **Counting mode** parameter is not compatible with HRPWM. To avoid an error when you build the model, do not set the **Counting mode** parameter to `Down` and select the **Enable HRPWM (Period)** parameter checkbox.

Note This parameter is available only for specific C28x devices.

Enable HRPWM Duty on ePWM#A(HRPWM - duty) — Indicates decimal part of the compare value
`off` (default) | `on`

When this parameter is enabled, decimal values will be accepted for the Compare A value (CMPA) of the ePWM Module. The Extension Register for the HRPWM Compare A (CMPAHR) provides an 8 bit representation of the decimal part of the compare value.

This parameter also enables **HRPWM control mode**.

Note This parameter only appears for specific C28x devices.

High resolution PWM (HRPWM) load mode on ePWM#A — Determines when to transfer CMPAHR shadow value to active register
`Counter equals to period (CTR=PRD)` (default) | `Counter equals to zero (CTR=ZERO)` | `Counter equals to either zero or period (CTR=ZERO or CTR=PRD)`

This parameter appears when **Enable high resolution PWM (HRPWM - period)** or **Enable high resolution PWM (HRPWM - duty)** is selected. Determine when to transfer the value of the CMPAHR shadow to the active register:

- `Counter equals to zero (CTR=ZERO)` — Transfers the value when the time base counter equals zero (TBCTR = 0x0000).
- `Counter equals to period (CTR=PRD)` — Transfers the value when the time base counter equals the period (TBCTR = TBPRD).
- `Counter equals to either zero or period (CTR=ZERO or CTR=PRD)` — Transfers the value when either case is true.

This parameter configures the HRLOAD shadow mode bit in the HRPWM configuration register (HRCNFG).

High resolution PWM (HRPWM) control mode on ePWM#A — Indicates the control mode
`Duty control mode` (default) | `Phase control mode`

This parameter appears when **Enable high resolution PWM (HRPWM - period)** or **Enable high resolution PWM (HRPWM - duty)** is selected. Select which register controls the Micro Edge Positioner (MEP) step size. The **High resolution PWM (HRPWM) control mode** option configures the CTLMODE control mode bits.

- `Duty control mode` — Uses the Extension Register for HRPWM Duty (CMPAHR) or the Extension Register for HRPWM Period (TBPRDHR) to control the MEP edge position.
- `Phase control mode` — Uses the Time Base Phase High Resolution Register (TBPHSHR) to control the MEP edge position.

High resolution PWM (HRPWM) Edge Control mode — Indicates the edge control mode
`Both Edge` (default) | `Rising Edge` | `Falling Edge`

Select the register that controls the MEP precise position control on the edge type.

- `Rising Edge` — MEP control of rising edge
- `Falling Edge` — MEP control of falling edge
- `Both Edge` — MEP control of both edges

The **High resolution PWM (HRPWM) edge control mode** option configures the EDGMODE edge mode bits in the HRPWM configuration register (HRCNFG).

Dependencies

This parameter appears when **Enable high resolution PWM (HRPWM - period)** or **Enable high resolution PWM (HRPWM - duty)** is selected.

Use scale factor optimizer (SFO) software — Indicates scale factor
`off` (default) | `on`

Enable scale factor optimizing (SFO) software with HRPWM. This software dynamically determines the scaling factor for the MEP step size. The step size varies depending on operating conditions such as temperature and voltage. The SFO software reduces variability due to these conditions. For more information, see the Scale Factor Optimizing Software (SFO) section of the *TMS320x2802x, 2803x Piccolo High Resolution Pulse Width Modulator (HRPWM) Reference Guide*.

Dependencies

- **Use scale factor optimizer (SFO) software** parameter is enabled, read-only and visible, only if the **Enable auto convert**, and **Enable high resolution PWM (HRPWM - period)** parameters are selected.
- **Use scale factor optimizer (SFO) software** parameter is editable, only if the **Enable auto convert** parameter is unselected.

Enable auto convert — Indicates auto convert
`off` (default) | `on`

Apply the scaling factor calculated by the SFO software to the controlling period or duty cycle. (Use the **HRPWM duty mode** to select controlling period or duty cycle.) This parameter sets the AUTOCONV field in the HRPWM Configuration Register (HRCNFG).

Dependencies

- **Enable auto convert** parameter only appears for specific C28x devices.
- **Enable auto convert** parameter is enabled, read-only and visible, only if **Enable high resolution PWM (HRPWM - period)** parameter is selected.
- **Enable auto convert** parameter is enabled, and editable, only if **Enable high resolution PWM (HRPWM - duty)** parameter is selected.

PWM Chopper Control Tab

Chopper module enable — Enables chopper module
`off` (default) | `on`

Select to enable the chopper module.

Chopper frequency divider — Determines frequency of chopper clock
1 (default) | 2 | . . .

Set the prescaler value that determines the frequency of the chopper clock. The system clock speed is divided by this value to determine the chopper clock frequency. Choose an integer value in the range 1 to 8.

Chopper clock cycles width of first pulse — High-energy first pulse to turn on hard and fast power switch
1 (default) | 2 | . . .

Choose an integer value in the range 1 to 16 to set the width of the first pulse. This feature provides a high-energy first pulse to turn on the hard and fast power switch.

Chopper pulse duty cycle — Determines duty cycles of second and subsequent pulses
12.5% (default) | 25% | . . .

The duty cycles of the second and subsequent pulses are also programmable. The duty cycle can be varied in steps of `12.5%` from `12.5%` to `87.5%`.

Trip Zone Unit

Trip zone source — Determines source of control logic for trip zone signals
`Specify via dialog` (default) | `Input port`

Specify the source of the control logic for the trip zone signals. Select `Specify via dialog` (the default) to enable specific trip zone signals in the block dialog. Choose `Input port` to enable specific trip-zone signals using a block input port **TZSEL**.

The **Trip Zone unit** tab lets you specify parameters for the Trip-zone (TZ) submodule. Each ePWM module receives TZ signals from the GPIO MUX. The number of Trip zone signals vary based on C28x processor families. These signals can be used to force the ePWM output into a specific state based on an event like an external fault. Use the settings on this tab to program the ePWM outputs to respond to external events.

If you select `Input port`, use this bit operation to determine the value of the 16-bit integer to send to the **TZSEL** input port:

`TZSEL INPUT VALUE = (DCBEVT1*2` ^{15} `+ DCAEVT1*2` ^{14} `+ OSHT6*2`^{13} `+ OSHT5*2`^{12} `+ OSHT4*2`^{11} `+ OSHT3*2`^{10} `+ OSHT2*2`^{9} `+ OSHT1*2`^{8} `+ DCBEVT2*2` ^{7} `+ DCAEVT2*2` ^{6} `+ CBC6*2`^{5} `+ CBC5*2`^{4} `+ CBC4*2`^{3} `+ CBC3*2`^{2} `+ CBC2*2`^{1} `+ CBC1*2`^{0}`)`

The software uses the higher 8 bits for the **One shot TZ1-TZ6** (OSHT1–6) and the lower 8 bits for **Cyclic TZ1-TZ6** (CBC1–6). You can set up a group of TZ sources (1~6), use a bit operation to combine them into an integer, and then feed the integer to TZSEL.

For example, to enable One Shot TZ6 (OSHT6) and One Shot TZ5 (OSHT5) as trip zone sources, set OSHT6 and OSHT5 to 1 and leave the remaining values as 0.

TZSEL INPUT VALUE = (1*2^{15} + 1*2^{14} + 1*2^{13} + 1*2^{12} + 0*2^{11} ...)

TZSEL INPUT VALUE = (8192 + 4096 + 0 ...)

TZSEL INPUT VALUE = 12288

When the block receives this value, it applies it to the TZSEL register as a binary value: `0011000000000000`.

For more information, see the Trip-Zone Submodule Control and Status Registers section of the TMS320x28xx, 28xxx Enhanced Pulse Width Modulator (ePWM) Module Reference Guide.

Enable one-shot trip zone# (TZ#) — Enables corresponding trip zone signal in one-shot mode
`off` (default) | `on`

Select this parameter to enable the corresponding trip zone signal in one-shot mode. In this mode, when the trip event is active, the trip zone module performs the corresponding action on the EPWM#A/B output immediately and latches the condition. You can unlatch the condition using software control.

Dependencies

This option is available only when the **Trip zone source** is `Specify via dialog`.

Enable one-shot digital compare A event 1 (DCAEVT1), Enable one-shot digital compare B event 1 (DCBEVT1) — Enables corresponding event signal as OST trip source for event 1
`off` (default) | `on`

Select these parameter to enable the corresponding event signal as an OST trip source for event 1. In this mode, if the digital compare A or digital compare B event 1 is active, the trip zone module performs the corresponding action on the EPWM#A/B output immediately and latches the condition. You can unlatch the condition using the software control.

Dependencies

- This option is available only when the **Trip zone source** is `Specify via dialog`.
- This parameter is available only for specific C28x processors.

Enable cyclic trip zone# (TZ#) — Enables corresponding trip zone signal in cycle-by-cycle mode
`off` (default) | `on`

Select this parameter to enable the corresponding trip zone signal in cycle-by-cycle mode. In this mode, when the trip event is active, the trip zone module performs the corresponding action on the EPWM#A/B output immediately and latches the condition. In cycle-by-cycle mode, the trip zone module automatically clears condition when the ePWM Counter reaches zero. Therefore, in cycle-by-cycle mode, every ePWM cycle resets or clears the trip event.

Dependencies

This option is available only when the **Trip zone source** is `Specify via dialog`.

Enable cyclic digital compare A event 2 (DCAEVT2), Enable cyclic digital compare B event 2 (DCBEVT2) — Enables corresponding event signal as cyclic trip source for event 2
`off` (default) | `on`

Select these parameters to enable the corresponding event signal as a cyclic trip source for event 2. In this mode, if the digital compare A or digital compare B event 2 is active, the Trip zone module performs the corresponding action on the EPWM#A/B output immediately and latches the condition. In Cycle-by-Cycle Mode, the Trip zone module automatically clears condition when the ePWM Counter reaches zero. Therefore, in Cycle-by-Cycle Mode, every ePWM cycle resets or clears the trip event.

Dependencies

- This option is available only when the **Trip zone source** is `Specify via dialog`.
- This parameter is available only for specific C28x processors.

Enable trip-zone one-shot interrupt (OST) — Enables trip zone one-shot interrupt
`off` (default) | `on`

Generate an interrupt when any of the enabled one shot (OST) triggering events occur.

Note `TZFLG.INT` is cleared in the post processing of ISR. However, if the interrupt flag is cleared when either `CBC` or `OST` is set, then another interrupt pulse will be generated. Clearing all flag bits will prevent further interrupts. This bit is cleared by writing the appropriate value to the `TZCLR` register.

Enable Trip-zone Cycle-by-Cycle interrupt (CBC) — Enables trip-zone cycle-by-cycle interrupt
`off` (default) | `on`

Generate an interrupt when any of the enabled cyclic or cycle-by-cycle (CBC) triggering events occur.

Note `TZFLG.INT` is cleared in the post processing of ISR. However, if the interrupt flag is cleared when either `CBC` or `OST` is set, then another interrupt pulse will be generated. Clearing all flag bits will prevent further interrupts. This bit is cleared by writing the appropriate value to the `TZCLR` register.

Digital comparator output A event 1 interrupt enable (DCAEVT1), Digital comparator output A event 2 interrupt enable (DCAEVT2), Digital comparator output B event 1 interrupt enable (DCBEVT1), Digital comparator output B event 2 interrupt enable (DCBEVT2) — Enables digital comparator output
`off` (default) | `on`

Generate an interrupt when Digital Comparator Output A or Digital Comparator Output B for event 1 or 2 occurs.

Note These parameters are available only for specific C28x processors.

ePWM#A forced (TZ) to, ePWM#B forced (TZ) to, ePWM#A forced (DCAEVT#) to, ePWM#B forced (DCBEVT#) to — Determines action to consider on ePWM output
`No action` (default) | `High` | `Low` | `Hi-Z (High Impedance)`

These parameters decide the actions you can take on the ePWM outputs in a trip zone condition. The trip zone module overrides and forces the ePWM#A and/or ePWM#B (TZ or DCAEVTx) output to one of these states: `No action` (the default), `High`, `Low`, or `Hi-Z (High Impedance)`.

Digital Compare

Source for digital compare A high signal (DCAH), Source for digital compare B high signal (DCBH) — Source for digital compare high signal
`GPTRIP1SEL/(TZ1) for DCAH/GPTRIP1SEL/(TZ1) for DCBH` (default) | `GPTRIP3SEL/(TZ3) for DCAH/GPTRIP7SEL for DCBH` | `...`

Select the appropriate TZ or COMP signal to generate high logic value for the Digital compare A/B high signal. Use the **Digital compare output A event # selection**, **Digital compare output B event # selection**, **DCAEVT# source select**, **DCBEVT# source select** parameters to determine the impact of DCAH/DCBH on DCAEVT# and DCBEVT#.

Each digital compare (DC) submodule receives three TZ signals (TZ1 to TZ3) from the GPIO MUX, and three COMP signals from the COMP (For specific C28x devices only). These signals indicate fault or trip conditions that are external to the ePWM submodule. Use the settings in this tab to output specific DC events in response to those external signals. These DC events feed directly into the time base, trip zone, and event trigger submodules.

For more information, see the *Digital Compare (DC) Submodule* section of the *Piccolo Enhanced Pulse Width Modulator (ePWM) Module Reference Guide*.

Source for digital compare A low signal (DCAL), Source for digital compare B low signal (DCBL) — Source for digital compare low signal
`GPTRIP1SEL/(TZ1) for DCAL/GPTRIP1SEL/(TZ1) for DCBL` (default) | `TRIPIN1(ECCDBLERR) for DCAL/Select From DCBLTRIPSEL for DCBL` | `...`

Select the appropriate TZ or COMP signal to generate low logic value for the Digital compare A/B low signal. Use the **Digital compare output A event # selection**, **Digital compare output B event # selection**, **DCAEVT# source select**, **DCBEVT# source select** options to determine the impact of DCAL/DCBL on DCAEVT# and DCBEVT#.

Digital compare output A event # selection (DCAEVT#), Digital Compare output B event # selection (DCBEVT#) — Qualify signals that generate DC events
`Event disabled` (default) | `DCAH=low and DCAL=don’t care/DCBH=low and DCBL=don’t care` | `...`

Qualify the signals that generate DC events, such as DCAEVT# or DCBEVT#. To disable this feature, choose the `Event disabled` option. Based on the source selection for the signals, the event can be triggered. Event selection can be based on the combination of conditions High, Low and Don't care.

DCAEVT# source select, DCBEVT# source select — Digital compare event source
`Event disabled` (default) | `DCAH=low and DCAL=don’t care/DCBH=low and DCBL=don’t care` | `...`

This parameter controls two separate aspects of triggering DC events:

- Triggering filtered or unfiltered DC event
  - The DC event can be a filtered or unfiltered signal that can be passed to other submodules.
  - Configures EVT1SRCSEL and EVT2SRCSEL in both DCACTL and DCBCTL registers.
  - Options that begin with `DCAEVT# with sync` or `DCAEVT# with async` do not apply filtering to DC events. Qualified DC signals pass directly to trigger DC events without any filtering.
  - Options that begin with `DCEVTFILT sync` apply filtering to DC events. Qualified signals pass through filtering circuits before triggering DC events. For more information, refer to the *Event Filtering* section of the *Piccolo Enhanced Pulse Width Modulator (ePWM) Module Reference Guide*.
- Trigger the DC event synchronously or asynchronously
  - The DC event signal can be synchronized with TBCLK or can remain asynchronously.
  - Configures EVT1SRCSEL and EVT2SRCSEL in both DCACTL and DCBCTL registers.
  - Options that end with `async` trigger DC events asynchronously. When the qualified or filtered signals exist, the DC submodule triggers the DC event immediately.
  - Options that end with `sync` trigger DC events synchronously. Once the qualified or filtered signals exist, the DC submodule triggers the DC event in sync with the TBCLK signal.

  Note The following fields appear when you select `DCEVTFILT with sync` or `DCEVTFILT with async` for the **DCAEVTX source select** or **DCBEVTX source select**.

  For more details about the following parameters, refer to the Technical Reference Manual for Enhanced Pulse Width Modulator (ePWM) Module Reference Guide.

Pulse select — Indicates blanking window which filters out event occurrences
`Counter equals to zero (CTR=Zero)` (default) | `Counter equals to period (CTR=PRD)`

The blanking window which filters out event occurrences on the signal while active. Set this parameter is aligned to either a CTR = PRD pulse or a CTR = Zero pulse.

Blanking window inverted — Indicates the inverted blanking window
`off` (default) | `on`

Select this parameter to enable the inverted blanking window.

Blanking window offset — Indicates offset blanking window
`0` (default)

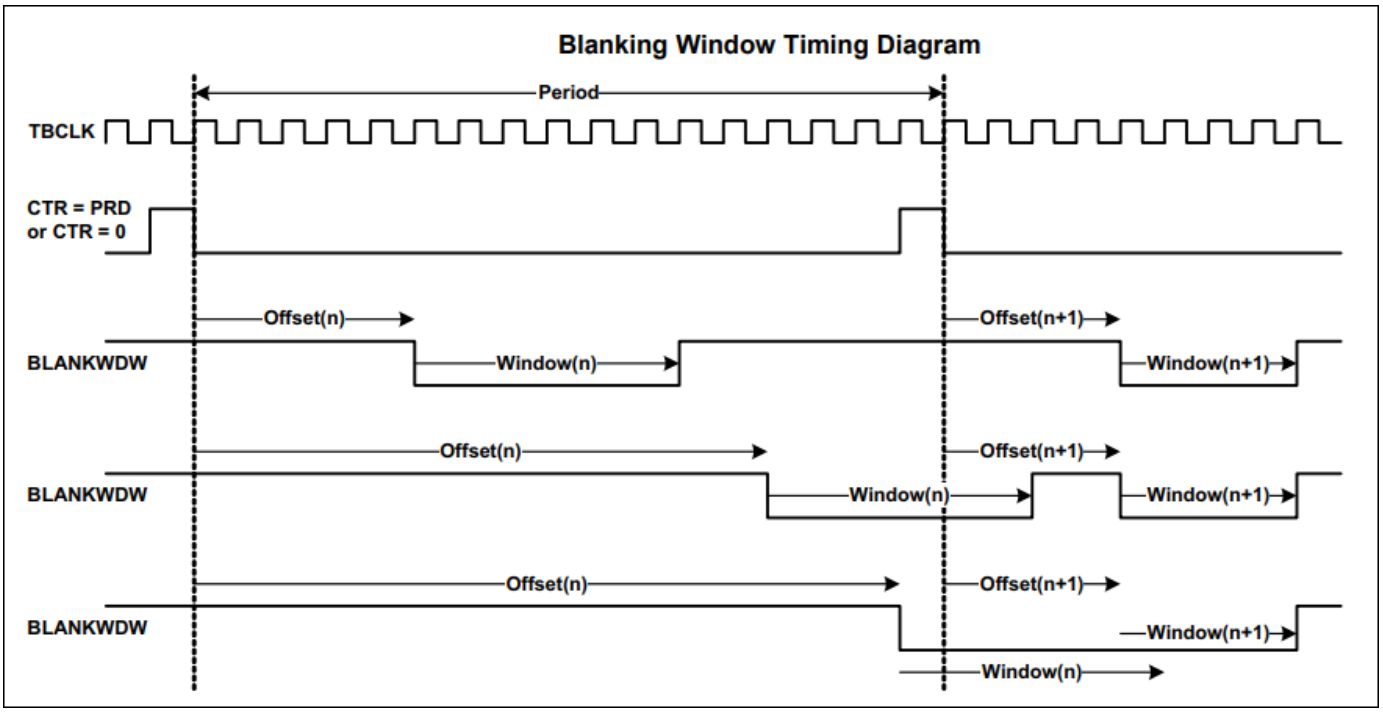
The number of TBCLK cycles to determine the point after the **Pulse select** event to start the blanking window for filtering

Blanking window width — Indicates duration of blanking window
`0` (default)

The number of TBCLK cycles to determine the point after the **Pulse select** event to start the blanking window for filtering.

The blanking window will start based on the occurrence **Pulse select** event with additional TBCLK cycles as offset as per the **Blanking offset window**.

During the blanking window all the events are ignored. The events can trigger sync or async events before and after the blanking window is applied.



Filter source select — Indicates source select for filter
`Filtered version of DCAEVT1 (DCAEVT1FILT)` (default) | `Filtered version of DCAEVT2 (DCAEVT2FILT)` | `Filtered version of DCBEVT1 (DCBEVT1FILT)` | `Filtered version of DCBEVT2 (DCBEVT2FILT)`

Use this parameter to select a source for filtering, which forms the DCEVTFILT signal.

Enable counter capture — Indicates counter capture
`off` (default) | `on`

Enabling this option allows to capture the TBCTR value of the trip event.

References

For more information, consult the following references, available at the Texas Instruments Web site:

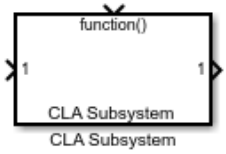
- Enhanced Pulse Width Modulator (ePWM) Module Reference Guide.
- High Resolution Pulse Width Modulator Reference Guide.
- Piccolo Enhanced Pulse Width Modulator (ePWM) Module Reference Guide.
- Piccolo High Resolution Pulse Width Modulator (HRPWM) Reference Guide.
- Piccolo Technical Reference Manual.
- Delfino Technical Reference Manual.
- Concerto Technical Reference Manual.
- Using the ePWM Module for 0% - 100% Duty Cycle Control Application Report.
- Configuring Source of Multiple ePWM Trip-Zone Events.
- DSPs Data Manual.
- Digital Signal Processor Data Manual.
- Digital Signal Controllers (DSCs) Data Manual.

See Also

C2802x/C2803x/C2805x/C2806x/F28M3x/F2807x/F2837xD/F2837xS/F28004x ADC | “Overview of Time-Base Synchronization in ePWM Type 4” | “C28x-ePWM” on page 1-145

CLA Subsystem

Group blocks to execute algorithm inside CLA

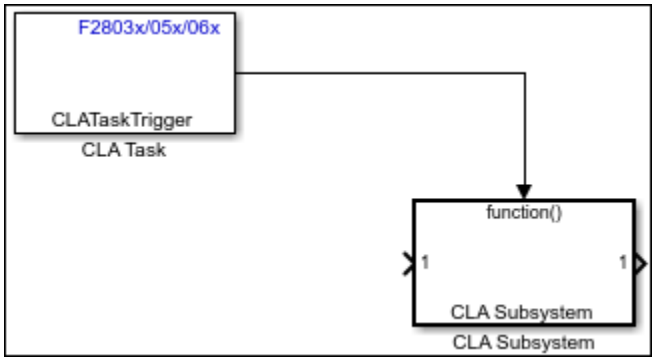


Libraries:
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x

Description

A CLA Subsystem block contains a subset of blocks within a model or system. CLA Subsystem is a triggered subsystem which is caused by a CLA Task block.

CLA Subsystem block can only be used along with CLA Task block.



CLA Subsystem requires `tic2000demospkg` package to be loaded in the model with a valid TI C2000 based hardware board.

- Open **Configuration Parameters** > **Hardware Implementation** > **Hardware board** and change the parameter to a valid `TI C2000 board`.
- After selecting `TI C2000 hardware board`, click **Manage packages** > **Refresh** > **Load** from Embedded coder dictionary app to load **tic2000demospkg**.

Note

- To know more about how to use the subsystem for CLA, refer “Overview of CLA Configuration for C2000 Processors Using Subsystem”.
- CLA Subsystem block is derived from Subsystem block with additional parameter configurations specific to CLA. It is recommended to use the options provided in this block for successful usage of CLA Subsystem.

Ports

Input

In — Signal input to a subsystem
scalar | vector | matrix

Placing an Inport block in a subsystem adds an external input port to the Subsystem block. The port label matches the name of the Inport block.

Use Inport blocks to get signals from the local environment.

Data Types: `single` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Output

Out — Signal output from a subsystem
scalar | vector | matrix

Placing an Outport block in a subsystem adds an output port from the Subsystem block. The port label on the Subsystem block is the name of the Outport block.

Use Outport blocks to send signals to the local environment.

Data Types: `single` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Parameters

Parameters on the Code Generation tab require a Simulink Coder™ or Embedded Coder license.

Main

Show port labels — Display options for port labels

`FromPortIcon` (default) | `FromPortBlockName` | `SignalName`

Select how to display port labels on the Subsystem block icon.

`none`
: Do not display port labels.

`FromPortIcon`
: If the corresponding port icon displays a signal name, display the signal name on the Subsystem block. Otherwise, display the port block name or the port number if the block name is a default name.

`FromPortBlockName`
: Display the name of the corresponding port block on the Subsystem block.

`SignalName`
: If the signal connected to the port is named, display the name of the signal on the Subsystem block. Otherwise, display the name of the corresponding port block.

Programmatic Use

Parameter: `ShowPortLabels`

Type: character vector
Value: `'FromPortIcon'` | `'FromPortBlockName'` | `'SignalName'`
Default: `'FromPortIcon'`

Read/Write permissions — Levels of access to contents of subsystem

`ReadWrite` (default) | `ReadOnly` | `NoReadOrWrite`

Control user access to the contents of the subsystem.

`ReadWrite`

Enable opening and modification of subsystem contents.

`ReadOnly`

Enable opening but not modification of the subsystem. If the subsystem resides in a block library, you can create and open links to the subsystem and can make and modify local copies of the subsystem but cannot change the permissions or modify the contents of the original library instance.

`NoReadOrWrite`

Disable opening or modification of subsystem. If the subsystem resides in a library, you can create links to the subsystem in a model but cannot open, modify, change permissions, or create local copies of the subsystem.

Note You do not receive a response if you attempt to view the contents of a subsystem whose **Read/Write permissions** parameter is set to `NoReadOrWrite`. For example, when double-clicking such a subsystem, Simulink does not open the subsystem and does not display any messages.

Programmatic Use

Parameter: `Permissions`
Type: character vector
Value: `'ReadWrite'` | `'ReadOnly'` | `'NoReadOrWrite'`
Default: `'ReadWrite'`

Name of error callback function — Name of function to be called if error occurs

`''` (default) | function name

Enter name of a function to be called if an error occurs while Simulink is executing the subsystem.

Simulink passes two arguments to the function: the handle of the subsystem and a character vector that specifies the error type. If no function is specified, Simulink displays a generic error message if executing the subsystem causes an error.

Programmatic Use

Parameter: `ErrorFcn`
Type: character vector
Value: `''` | `'<function name>'`
Default: `''`

Permit hierarchical resolution — Resolution for workspace variable names

`All` (default) | `ExplicitOnly` | `None`

Select whether to resolve names of workspace variables referenced by this subsystem.

For more information, see “Symbol Resolution” and “Symbol Resolution Process”.

`All`

Resolve all names of workspace variables used by this subsystem, including those used to specify block parameter values and Simulink data objects (for example, `Simulink.Signal` objects).

`ExplicitOnly`

Resolve only names of workspace variables used to specify block parameter values, data store memory (where no block exists), signals, and states marked as “must resolve”.

`None`

Do not resolve any workspace variable names.

Programmatic Use

Parameter: `PermitHierarchicalResolution`
Type: character vector
Value: `'All'` | `'ExplicitOnly'` | `'None'`
Default: `'All'`

Code Generation

Function packaging — Code format

`Nonreusable function` (default) | `Inline`

Select the code format to be generated for CLA subsystem.

Note CLA Subsystem block doesn’t support `Auto` and `Reusable function` code format.

`Inline`

Simulink Coder and Embedded Coder inline the subsystem unconditionally and entire algorithm is inlined to the CLA task (.cla) file. For more, see “Method 2 - Inline Code Generation for CLA Subsystem”

`Nonreusable function`

In this subsystem, the code for the algorithm in CLA is generated as a separate `.C` file along with data. For more, see “Method 1 - Nonreusable Function Code Generation for CLA Subsystem (Recommended)”.

Subsystems with this setting generate functions that might have arguments depending on the **Function interface** parameter setting. You can name the generated function and file using parameters **Function name** and **File name (no extension)**. These functions are not reentrant.

Dependencies

- This parameter requires Simulink Coder for code generation.

Programmatic Use

Parameter: `RTWSystemCode`
Type: character vector
Value: | `'Inline'` | `'Nonreusable function'` |

Default: `'Nonreusable function'`

Function name options — How to name generated function

`Auto` (default) | `Use subsystem name` | `User specified`

Select how Simulink Coder names the function it generates for the subsystem.

If you have an Embedded Coder license, you can control function names with options on the Configuration Parameter **Code Generation > Identifiers** pane.

`Auto`

Assign a unique function name using the default naming convention, *`model_subsystem`*`()`, where *`model`* is the name of the model and *`subsystem`* is the name of the subsystem (or that of an identical one when code is being reused).

`Use subsystem name`

Use the subsystem name as the function name. By default, the function name uses the naming convention *`model_subsystem`*.

`User specified`

Enable the **Function name** field. Enter any legal C or C++ function name, which must be unique.

Dependencies

- This parameter requires a Simulink Coder license.
- To enable this parameter, set **Function packaging** to `Nonreusable function` or `Reusable function`.

Programmatic Use

Parameter: `RTWFcnNameOpts`
Type: character vector
Value: `'Auto'` | `'Use subsystem name'` | `'User specified'`
Default: `'Auto'`

Function name — Name of function for subsystem code

`''` (default) | function name

Specify a unique, valid C or C++ function name for subsystem code.

Use this parameter if you want to give the function a specific name instead of allowing the Simulink Coder code generator to assign its own autogenerated name or use the subsystem name.

Dependencies

- This parameter requires a Simulink Coder license.
- To enable this parameter, set the **Function name options** parameter to `User specified`.

Programmatic Use

Parameter: `RTWFcnName`
Type: character vector
Value: `''` | `'<function name>'`
Default: `''`

File name options — How to name generated file

`User specified` (default)

Select how Simulink Coder names the separate file for the function it generates for the CLA subsystem. Ensure that you set the **File name options** parameter to `User specified` for CLA Subsystem.

`User specified`

This option enables the **File name (no extension)** text entry field. The code generator uses the name you enter as the file name. Enter any file name, but do not include the `.c` or `.cpp` (or any other) extension. This file name need not be unique.

Note While a subsystem source file name need not be unique, you must avoid giving nonunique names that result in cyclic dependencies (for example, `sys_a.h` includes `sys_b.h`, `sys_b.h` includes `sys_c.h`, and `sys_c.h` includes `sys_a.h`).

Dependencies

- This parameter requires a Simulink Coder license.
- To enable this parameter, set **Function packaging** to `Nonreusable function`.

Programmatic Use

Parameter: `RTWFileNameOpts`
Type: character vector
Value: `'User specified'`
Default: `'User specified'`

File name (no extension) — Name of generated file

`sample_cla` (default) | file name

The algorithm inside CLA is generated in the source file. (For example `sample_cla.c` which includes `sample_cla.h`). The file name that you specify does not have to be unique. However, avoid giving non-unique names that result in cyclic dependencies (for example, `sys_a.h` includes `sys_b.h`, `sys_b.h` includes `sys_c.h`, and `sys_c.h` includes `sys_a.h`).

Dependencies

- This parameter requires a Simulink Coder license.
- To enable this parameter, set **File name options** to `User specified`.

Programmatic Use

Parameter: `RTWFileName`
Type: character vector
Value: `''` | `'<file name>'`
Default: `''`

Function interface — Select to use arguments with generate function

`void_void` (default)

Select to use arguments with generated function. Ensure that you set the **Function interface** parameter to `void_void` for CLA Subsystem.

`void_void`

Generate a function without arguments and pass data as global variables. For example:

```
void subsystem_function(void)
```

For more information, see:

- “Reduce Global Variables in Nonreusable Subsystem Functions” (Embedded Coder)
- “Generate Modular Function Code for Nonvirtual Subsystems” (Embedded Coder)

Dependencies

- To enable this parameter, set **Function packaging** to `Nonreusable function`.

Programmatic Use

Parameter: `FunctionInterfaceSpec`
Type: character vector
Value: `'void_void'`
Default: `'void_void'`

Function with separate data — Control code generation for subsystem

on (default) | off

Generate subsystem function code in which the internal data for an atomic subsystem is separated from its parent model and is owned by the subsystem.

 off

Do not generate subsystem function code in which the internal data for an atomic subsystem is separated from its parent model and is owned by the subsystem.

 on

Generate subsystem function code in which the internal data for an atomic subsystem is separated from its parent model and is owned by the subsystem. The subsystem data structure is declared independently from the parent model data structures. A subsystem with separate data has its own block I/O and `DWork` data structure. As a result, the generated code for the subsystem is easier to trace and test. The data separation also tends to reduce the maximum size of global data structures throughout the model, because they are split into multiple data structures.

Note It is recommended that the **Function with separate data** parameter is set to `on`.

For details on how to generate modular function code for an atomic subsystem, see “Generate Modular Function Code for Nonvirtual Subsystems” (Embedded Coder).

For details on how to apply memory sections to atomic subsystems, see “Override Default Memory Placement for Subsystem Functions and Data” (Embedded Coder).

Dependencies

- To enable this parameter, set **Function packaging** to `Nonreusable function`.

Programmatic Use

Parameter: `FunctionWithSeparateData`
Type: character vector
Value: `'off' | 'on'`
Default: `'on'`

Memory section for initialize/terminate functions — Select how to apply memory sections

`C28xFunction` (default)

Select how Embedded Coder applies memory sections to the subsystem initialization and termination functions. For CLA Subsystem, select `C28xFunction`. With this selection the initialize and terminate functions are protected with CLA complier tag such that it is compiled only by C28x compiler.

Note Ensure that the model is loaded with **tic2000demospkg** package where C28xFunction is defined.

For example, if name of the initialize function is `subsystem_initialize` then

```
#ifndef __TMS320C28XX_CLA__
void subsystem_initialize()
{
//body of the function
}
#endif
```

Dependencies

- To enable this parameter, set **Function packaging** to `Nonreusable function`.

Programmatic Use

Parameter: `RTWMemSecFuncInitTerm`
Type: character vector
Value: `C28xFunction`
Default: `C28xFunction`

Memory section for execution functions — Select how to apply memory sections

`ClaFunction` (default)

Select how Embedded Coder applies memory sections to the subsystem execution functions. For CLA Subsystem, select `ClaFunction`. With this selection the execution functions are protected with CLA complier tag such that it is compiled only by CLA compiler.

Note Ensure that the model is loaded with **tic2000demospkg** package where ClaFunction is defined.

For example, if name of the execution function is `subsystem` then

```
#ifdef __TMS320C28XX_CLA__
void subsystem()
{
//body of the function
}
#endif
```

Dependencies

- To enable this parameter, set **Function packaging** to `Nonreusable function`.

Programmatic Use
Parameter: `RTWMemSecFuncExecute`
Type: character vector
Value: `ClaFunction`
Default: `ClaFunction`

Memory section for constants — Select how to apply memory sections

`ClaDataRAM` (default)

Select how Embedded Coder applies memory sections to the subsystem constants. For CLA Subsystem, select `ClaDataRAM`. Memory section for constants parameter allows the constants to be stored in CLA data RAM.

Ensure that the model is loaded with **tic2000demospkg** package where ClaDataRAM is defined.

Dependencies

- To enable this parameter, set **Function packaging** to `Nonreusable function` and select the **Function with separate data** parameter.

Programmatic Use
Parameter: `RTWMemSecDataConstants`
Type: character vector
Value: `ClaDataRAM`
Default: `ClaDataRAM`

Memory section for internal data — Select how to apply memory sections

`ClaDataRAM` (default)

Select how Embedded Coder applies memory sections to the subsystem internal data. For CLA Subsystem, select `ClaDataRAM`. Memory section for internal data parameter allows the internal data to be stored in CLA data RAM.

Dependencies

- To enable this parameter, set **Function packaging** to `Nonreusable function` and select the **Function with separate data** parameter.

Programmatic Use
Parameter: `RTWMemSecDataInternal`
Type: character vector

Value: `ClaDataRAM`
Default: `ClaDataRAM`

Memory section for parameters — Select how to apply memory sections

`ClaDataRAM` (default)

Select how Embedded Coder applies memory sections to the subsystem parameters. For CLA Subsystem, select `ClaDataRAM`. Memory section for parameters allows the parameters to be stored in CLA data RAM

Dependencies

- To enable this parameter, set **Function packaging** to `Nonreusable function` and select the **Function with separate data** parameter.

Programmatic Use
Parameter: `RTWMemSecDataParameters`
Type: character vector
Value: `ClaDataRAM`
Default: `ClaDataRAM`

Subsystem Reference

Subsystem file name — File name for referenced subsystem

string | character vector

Dependencies

To access this parameter, click the **Convert** button.

For more information on how to convert a subsystem to a referenced subsystem, see “Convert an Existing Subsystem to a Referenced Subsystem”.

Version History

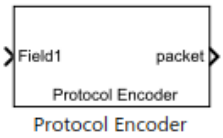
Introduced in R2021b

See Also

C28x CLA Task | “Overview of CLA Configuration for C2000 Processors Using Subsystem” | Subsystem

Protocol Encoder

Encode input data into a uint8 byte stream by specifying the packet structure



Libraries:
C2000 Microcontroller Blockset / Target Communication

Description

The Protocol Encoder encodes input data into a uint8 byte stream as per the specified packet structure based on the communication protocol. You can use this block to encode the separate fields into a packet, by specifying the header, terminator, name of packet fields in sequence and their size. You can also generate checksum bytes for the packet for validating the packet.

Ports

Input

Field1 — First field to be encoded by the block
scalar

First field to be encoded by the block. The name of input port is defined using the block parameter `Field name`. Each subsequent field that you create while defining the packet structure results in a new input port, with the `Field name` value appearing as the name of input port.

The block can have from 1 to `N` input ports. `N` is the number of fields in the packet structure that you specify in the **Specify packet fields sequentially** pane.

Output

Packet — uint8 byte stream packet
vector

The uint8 byte stream packet that contains the encoded data from multiple signals as specified in the packet structure.

Data Types: `uint8`

Parameters

Header — Header of the packet
character array | array of numeric values less than 255

Specify the header that indicates the beginning of the encoded data. The encoded output data begins with the header.

The numeric array specified in this parameter is the `uint8` integer representation of the corresponding ASCII characters.

Few examples of header values are: ‘START’, '$’, 36, [36,37].

Specify packet fields sequentially

Field name — Name of the field in the packet structure
`Field1` (default) | string

Specify the name of the field to be included in the packet structure. This name appears as the name of the input port.

To add a new packet field and specify its properties, click **Add**. After adding a field, click **Move Up** or **Move Down** to change its sequence in the structure.

To delete a field and its properties, select the row and click **Delete**.

Input field length — Length of the input field as per the data type
1 (default) | numeric scalar

Length of the input field as per the data type that you enter in the next column (`Input field data type`). For example, if the `Input field data type` of a field is ‘uint16’ and the length is 1, the encoder converts the input into two `uint8` bytes in the output packet.

Input field data type — Data type of the field
`uint8` (default) | `int8` | `int16` | `uint16` | `int32` | `uint32` | `int64` | `uint64` | `single` | `double`

Select the data type of the field.

Note `double` is considered as 8 bytes. If your hardware does not support 8 bytes `double`, selecting this data type might give incorrect results.

Byte order — Endianess of the field in the packet
`Little endian` (default) | `Big endian`

Select the endianess of the packet field to describe the order in which bytes are unpacked.

Specify logic to generate checksum for validation — Enable the logic for checksum validation and select the logic
`XOR of bytes` | `2's complement of sum of bytes` | `Custom algorithm`

Specify the logic to generate checksum for validation using standard algorithms or a custom algorithm.

If the logic is selected as `XOR of bytes`, the block calculates checksum as a single byte which is XOR of all bytes excluding header, terminator (if applicable).

If the logic is selected as `2's complement of sum of bytes`, the block calculates checksum as a single byte which is the 2's complement of sum off all bytes excluding header and terminator (if applicable).

If you select `Custom algorithm`, the additional parameters – **Checksum size** and **File path** – are enabled. In this case, you can provide your own custom logic to validate packet. The checksum bytes returned by the custom logic should be of size specified in the **Checksum size** field and these bytes are appended to the packet. For more information, see “Function Template for Custom Checksum Logic in Protocol Encoder Block” on page 2-277.

For the encoded packet in all the above three cases, checksum bytes are appended either at the end (if terminator is not available) or before the terminator (if terminator is specified).

Checksum size — Size of checksum bytes specified in the function used for custom algorithm
numeric scalar

Specify size of the checksum bytes generated by the function used for custom algorithm. For more details, see “Function Template for Custom Checksum Logic in Protocol Encoder Block” on page 2-277.

Dependencies

This parameter is enabled only if you select `Custom algorithm` option under **Specify logic for checksum validation** parameter.

Data Types: `uint8`

File path — Path of the function (.m) used for custom checksum logic
MATLAB path

Specify the path of the function (.m) in which you defined the custom algorithm for generating checksum bytes. The file must be saved in a directory in the MATLAB path.

Dependencies

This parameter is enabled only if you select `Custom algorithm` option under **Specify logic for checksum validation** parameter.

Data Types: `uint8`

Terminator — Terminator of the packet
<none> (default) | `CR ('\r')` | `LF ('\n')` | `CR/LF ('\r\n')` | `NULL ('\0')` | `Custom Terminator`

Specify the terminator that indicates the end of the encoded data. The last byte/bytes in the encoded output is the terminator.

If you select `Custom Terminator`, you can specify your own terminator value.

Custom terminator — Custom terminator of the packet
character array | array of numeric values less than 255

Specify the custom terminator that indicates the end of the encoded data. The last byte/bytes in the encoded output is the terminator.

The numeric array specified in this parameter is the `uint8` integer representation of the corresponding ASCII characters.

Few examples of custom terminator values are: ‘END’, '#’.

Data Types: `uint8`

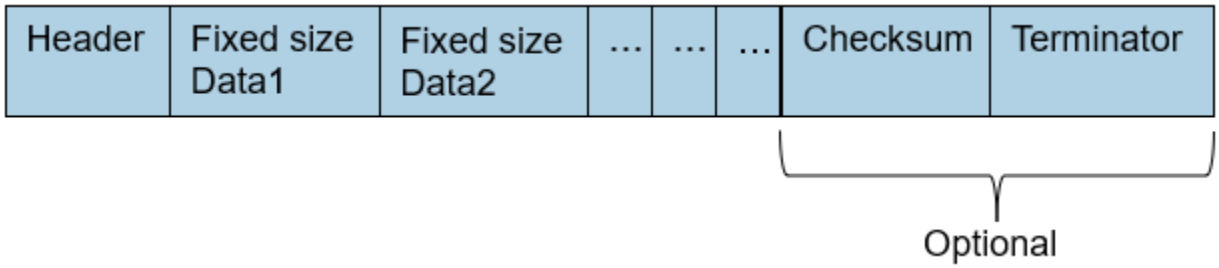
More About

Encode Data from Fixed-Sized Packet

The Protocol Encoder block can be used to encode input data into a fixed size packet (`uint8` byte stream).

The block converts the input data into uint8 bytes as per the byte order specified, and appends the data with header, and optional checksum and terminator values.

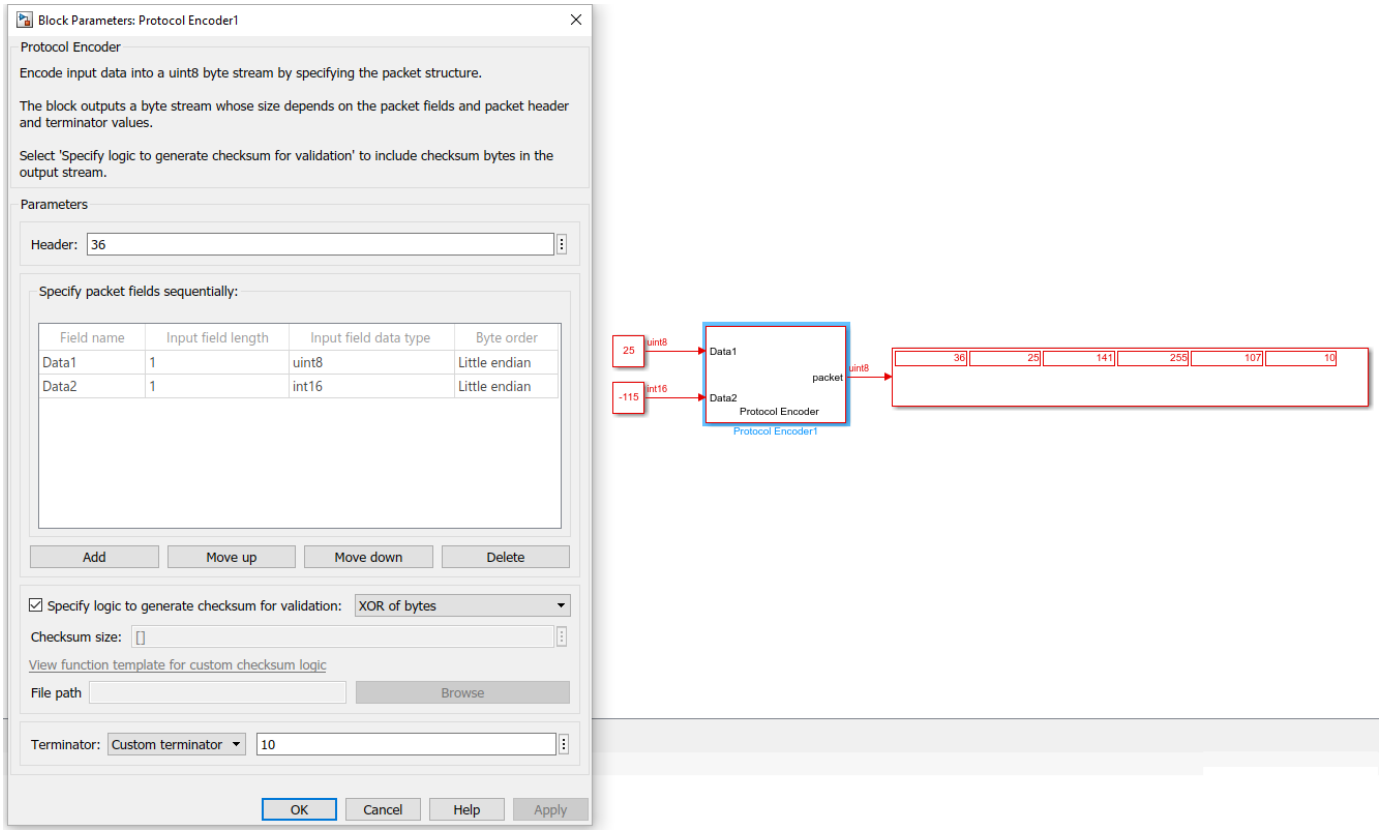
The general structure of a fixed size packet look like this, with optional checksum and terminator fields



For example, suppose that you need to encode input into packet as specified in this structure:

Header (36)	Data1 Data type: uint8	Data2 Data type: int16	Checksum (XOR based)	Terminator (10)

In this case, you specify the block parameters as shown in the below figure. The output displayed when you simulate the model conforms to the packet structure.



Function Template for Custom Checksum Logic in Protocol Encoder Block

```
function csBytes = generateChecksumBytes(payload)
%#codegen
end
```

Create the function as specified above. The function must return checksum bytes (csBytes) as `uint8` data type, whose size must be specified in the **Checksum size** field available in the block mask. The input to the function is the payload containing all the data bytes excluding header and terminator, as `uint8` datatype.

Save the function and specify the path of the function in the **File path** field in the block mask. Ensure that the file is in the MATLAB path.

Monitor Signals and Tune Parameters

The Protocol Encoder block is available with Simulink Support Package for Arduino® Hardware and Embedded Coder Support Package for Texas Instruments C2000 Processors. However, to monitor signals and tune parameters for protocol encoding in Simulink models to be run on TI's C2000-based targets, use the Universal Measurement and Calibration Protocol (XCP)-based External mode simulation.

Version History

Introduced in R2021b

See Also

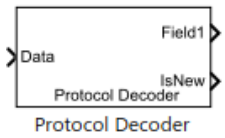
Protocol Decoder

Topics

“Encode and Decode Serial Data Using C2000-based Hardware” on page 2-316

Protocol Decoder

Decode a uint8 byte stream by specifying the packet structure



Libraries:
C2000 Microcontroller Blockset / Target Communication

Description

The Protocol Decoder block decodes a uint8 byte stream as per the specified packet structure based on the communication protocol. You can use this block to decode packet into separate fields, by specifying the header, terminator, name of packet fields in sequence and their size. You can also validate the packet using the checksum logic that you specify.

Ports

Input

Data — uint8 byte stream packet
vector

The uint8 byte stream packet that contains the encoded data from multiple signals as specified in the packet structure.

The input packet can be partially received over multiple sample times. The Protocol Decoder block has the capability to buffer such partial data until the complete packet is received.

Data Types: `uint8`

Length — Length of data stream packet at Data input port
numeric scalar

Optional input port to include the exact length of valid data stream. Use this option when you know the exact length of the valid data in the input data stream

This option is useful when you have a communication channel receive peripheral that outputs partially received data that contains trailing zeros. Such peripherals also output the length of the actual number of valid data bytes received. You can connect the length output of the peripheral directly with the **Length** input port of Protocol Decoder block, so that trailing zeros in the input byte stream do not affect the decoding logic.

Dependencies

This input port appears only if you select the **Is input data stream length available** parameter.

Data Types: `uint16`

Output

Field1 — First field decoded by the block
scalar

Name of the first field decoded by the block. The name of the output port is defined using the block parameter, `Field name`. Each subsequent field that you create while defining the packet structure results in a new output port, with the `Field name` value appearing as the name of output port.

The block can have from 1 to `N` output ports. `N` is the number of fields in the packet structure that you specify in the **Specify packet fields sequentially** pane.

Data Types: `string`

IsNew — New data indicator
`0` | `1`

New data indicator returned as a logical. A value of 1 indicates that a new data is available since the last sample was received by the block. This output can be used to trigger subsystems to process new data received from the Protocol Decoder block.

If `IsNew` = 0, the block outputs decoded fields from the last complete packet decoded by the block. If no last sample is available, the block outputs zeros.

Data Types: `Boolean`

IsValid — Checksum validation indicator
`0` | `1`

Checksum validation indicator returned as a logical. A value of 1 indicates that the input packet received by the block is valid when validated using the checksum logic that you specified. If `IsValid` = 0, the block outputs zeros.

Dependencies

This output port appears only if you select the **Specify logic for checksum validation** parameter.

Data Types: `Boolean`

Variable field length — Actual data length of the last field that is variable-sized
numeric scalar

Actual data length of the last field that is variable-sized. The maximum size of the variable size field can be specified using **Specify maximum length if last field has variable size** parameter. The block outputs the variable-sized data as a fixed size data with size that is equal to the maximum length specified. The output contains actual data followed by trailing zeros. The size of actual data can be determined using this **Variable Field Length** output.

Dependencies

This output port appears only if you select the **Specify maximum length if last field has variable size** parameter. For more details, see “Decode Variable-sized Packet” on page 2-286.

Data Types: `double`

Variable field lengths — Actual data lengths of comma separated variables
array

Actual data lengths of comma separated variables that are decoded from the input stream.

This output value (1-by-`N` array) corresponds to the actual data length of the fields in the packet, where N is the number of comma separated variables in the packet.

When **Parse comma separated variables** is selected, the `Output field length` parameter value corresponds to the maximum size of the field. The actual size of the field is available as the **Variable field lengths** output.

If the actual field length of n^{th} field is less than the specified `Output field length` parameter value on the block, the output corresponding to the field is actual data followed by trailing zeros. The n^{th} element of the **Variable field lengths** output provides the actual length corresponding to that variable in that array.

If the actual field length of n^{th} field is greater than the specified `Output field length` parameter value on the block, the output corresponding to the field is truncated and outputted, given the actual packet size does not exceed maximum packet size. The n^{th} element of the **Variable field lengths** output provides the actual length corresponding to that variable in that array.

Dependencies

This output port appears only if you select the **Parse comma separated variables** parameter. For more details, see “Decode Comma-separated Variable” on page 2-287.

Data Types: `double`

Parameters

Parse comma separated variables — Parse input data that contains comma separated variables
array

Specify that the input data to be parsed contains comma separated variables.

Data Types: `uint8`

Header — Header of the packet
character array | array of numeric values less than 255

Specify the header that indicates the beginning of the data. The simulation disregards data that occurs before the header. The header data is not sent to the output port.

The numeric array specified in this parameter is the `uint8` integer representation of the corresponding ASCII characters.

Few examples of header values are: ‘START’, '$’, 36, [36,37].

Specify packet fields sequentially

Field name — Name of the field in the packet structure
`Field1` (default) | string

Specify the name of the field included in the packet structure. This name appears as the name of the output port.

To add a new packet field and specify its properties, click **Add**. After adding a field, click **Move Up** or **Move Down** to change its sequence in the structure.

To delete a field and its properties, select the row and click **Delete**.

Output field length — Length of the field in the packet structure as per the data type
1 (default) | numeric scalar

Length of the field in the packet structure as per the data type that you enter in the next column (`Output field data type`). For example, if the `Output field data type` of a field is `uint16` and the length is 1, the decoder reads two bytes from the packet as per the packet structure and combine these bytes to output a `uint16` value.

If you select **Parse comma separated variables** parameter, the `Output field length` value corresponds to the maximum length that the field can have.

Data Types: `double`

Output field data type — Data type of the field
`uint8` (default) | `int8` | `int16` | `uint16` | `int32` | `uint32` | `int64` | `uint64` | `single` | `double`

Select the output data type of the field.

Note `double` is considered as 8 bytes. If your hardware does not support 8 bytes `double`, selecting this data type might give incorrect results.

Note If you select the **Parse comma separated variables** parameter, you cannot edit this parameter (the **Output field data type** is set to `uint8` always).

Byte order — Endianess of the field in the packet
`Little endian` (default) | `Big endian`

Select the endianess of the packet field to describe the order in which bytes are transmitted.

Note If you select the **Parse comma separated variables** parameter, you cannot edit this parameter (the **Byte order** is set to `Little endian` always).

Specify maximum length if last field has variable size — Enable last field to output variable-sized data and specify maximum length of the field
numeric scalar

Enable last field of the packet to output variable-sized data and specify the maximum length of the variable length field, which is present as the last field in the packet. For more details, see “Function Template for Custom Checksum Logic in Protocol Decoder Block” on page 2-289.

Specify logic for checksum validation — Enable the logic for checksum validation and select the logic
`XOR of bytes` | `2's complement of sum of bytes` | `Custom algorithm`

Specify the logic to generate checksum for validation using standard algorithms or a custom algorithm.

If you select `Custom algorithm`, the additional parameters – **Checksum size** and **File path** – are enabled.

Checksum bytes are expected to be in the last in the packet (before terminator, if terminator is available).

If you select the logic as `XOR of bytes`, checksum byte is expected to be the last byte before terminator. The logic calculates the XOR of all bytes excluding header, terminator and checksum byte, and compare it with the checksum byte.

If you select the logic as `2's complement of sum of bytes`, checksum byte is expected to be the last byte before terminator. The logic calculates the 2's complement of sum off all bytes excluding header, terminator and checksum byte, and compare it with the checksum byte.

If you select `Custom algorithm`, you can provide your own custom logic to validate packet. In this case, checksum bytes are extracted as per the size specified in the **Checksum size** field. For more information, see “Function Template for Custom Checksum Logic in Protocol Decoder Block” on page 2-289.

Checksum size — Number of bytes of checksum data
numeric scalar

Specify the number of bytes of checksum data. The blocks extracts the checksum bytes as per the specified size and passes it to the function used for custom algorithm, for validating the packet. For more details, see “Function Template for Custom Checksum Logic in Protocol Decoder Block” on page 2-289. Checksum bytes are expected to be in the last in the packet (before terminator, if terminator is available).

Dependencies

This parameter is enabled only if you select `Custom algorithm` option under **Specify logic for checksum validation** parameter.

Data Types: `uint8`

File path — Path of the function (.m) used for custom checksum logic
MATLAB path

Specify the path of the function (.m) in which you defined the custom algorithm for checksum logic. The file must be saved in a directory in the MATLAB path. For more details, see “Function Template for Custom Checksum Logic in Protocol Decoder Block” on page 2-289.

Dependencies

This parameter is enabled only if you select `Custom algorithm` option under **Specify logic for checksum validation** parameter.

Data Types: `uint8`

Terminator — Terminator of the packet
`<none>` (default) | `CR ('\r')` | `LF ('\n')` | `CR/LF ('\r\n')` | `NULL ('\0')` | `Custom Terminator`

Specify the terminator that indicates the end of the data. The terminator data is not sent to the output port.

If you select `Custom Terminator`, you can specify your own terminator value.

Data Types: `uint8`

Custom terminator — Custom terminator of the packet
character array

Custom terminator that indicates the end of the data. The terminator data is not sent to the output port.

The numeric array specified in this parameter is the `uint8` integer representation of the corresponding ASCII characters.

Few examples of custom terminator values are: 'END', '#'

Is input data stream length available — Enable additional input for data stream length
`off` (default) | `on`

Enable optional input port to include the exact length of valid data stream.

More About

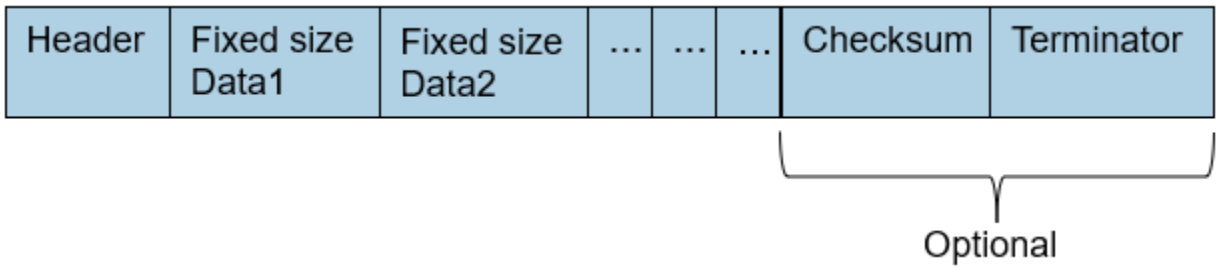
Decode Fixed-Sized Packet

The Protocol Decoder block can be used to decode fixed size input packet (`uint8`) into separate fields.

The block extracts the separate fields as per the packet structure specified, and convert them to output data of the data type specified in the `Output field data type` parameter as per the `Byte order`.

The input packet can be partially received in over multiple sample times. The Protocol Decoder block has the capability to buffer such partial data until the complete packet is received.

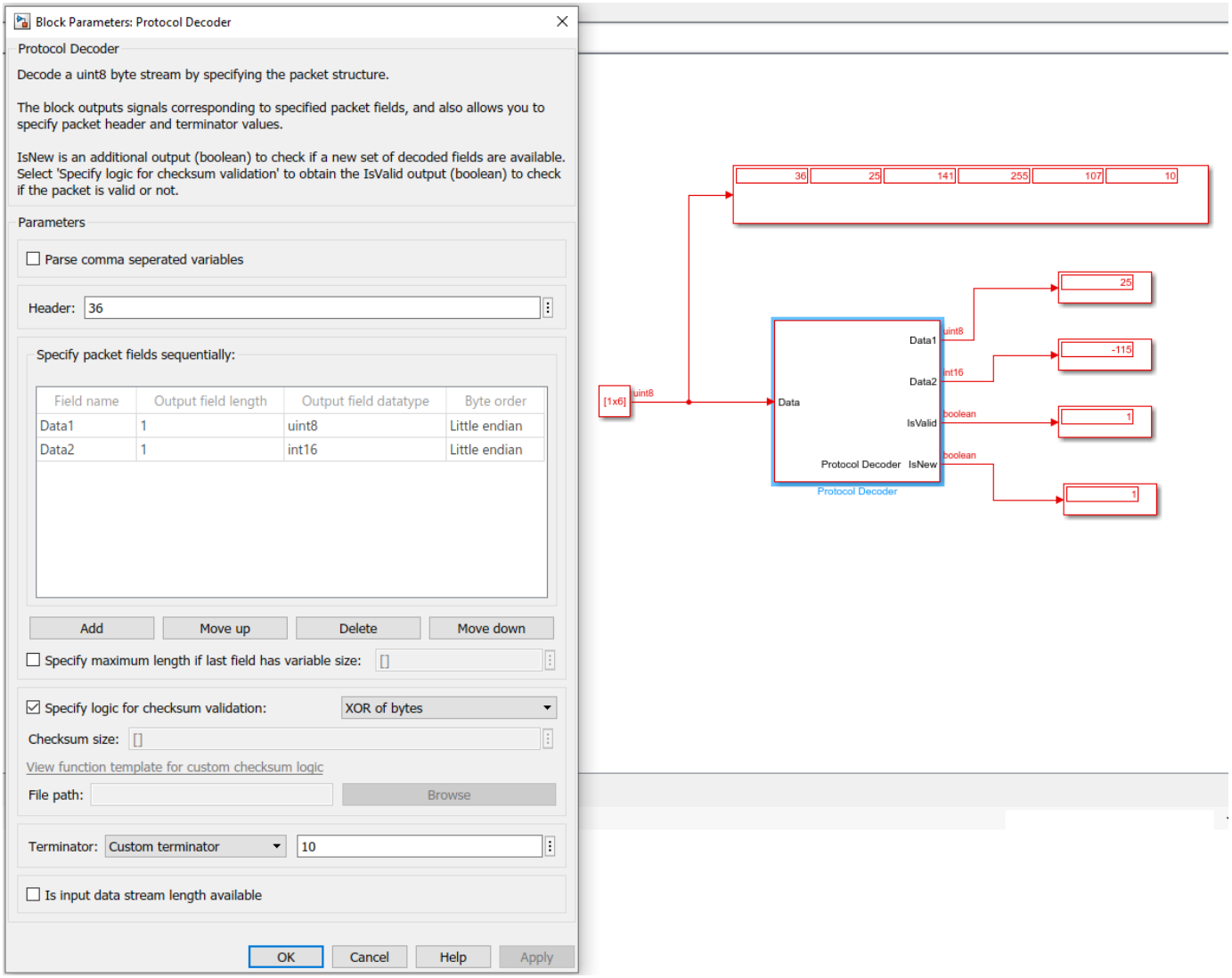
The general structure of a fixed size packet look like this, with optional checksum and terminator fields:



For example, suppose that you need to decode input packet as specified in this structure:

Header (36)	Data1 Data type: uint8	Data2 Data type: int16	Checksum (XOR based)	Terminator (10)

In this case, you specify the block parameters as shown in the below figure. The output displayed when you simulate the model conforms to the packet structure:



The block follows this sequence for decoding the data:

1. Extract packet based on header, packet size and terminator (if applicable). The packet size is determined by adding these:

   Sum of values specified using `Output field length` parameter (in bytes) **+** Value of `Checksum size` parameter (if applicable).

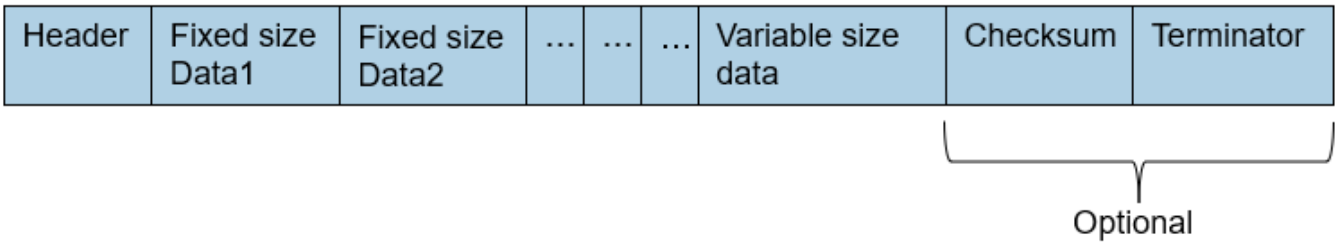
   The block waits for the header, to start decoding. Once it receives the header, it reads the fixed data fields as per the packet size. If the terminator is specified, the blocks checks if the subsequent bytes following the fixed size bytes are terminator bytes. If the subsequent bytes are not terminator bytes, the packet is discarded.
2. Validate the data received from the previous step using the checksum logic specified (if applicable).
3. Split the fields in the packet as per the length, datatype, and endianness specified, and provide them as output.

Decode Variable-sized Packet

If the last field in the packet is variable-sized, you select the parameter **Specify maximum length if last field has variable size**.

The input packet can be partially received in over multiple sample times. The Protocol Decoder block has the capability to buffer such partial data until the complete packet is received.

The general structure of a packet containing variable-sized field may look like this, with optional checksum and terminator fields:



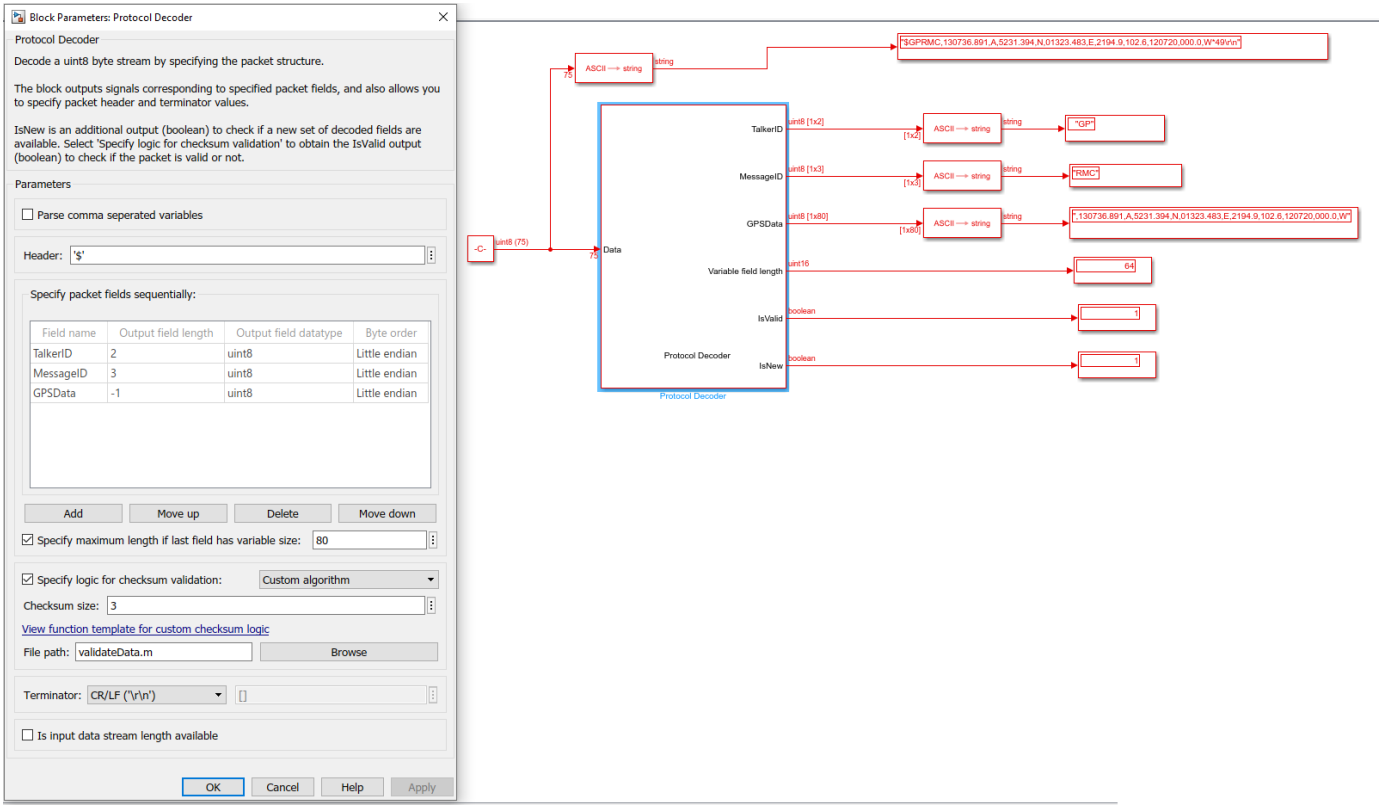
For example, NMEA (National Marine Electronics Association) sentences given by a GPS device can be considered as a packet that contains both fixed size fields and variable size field. The general structure of these sentences looks like this:

Header ('$')	Talker ID (2 bytes)	Message ID (3 bytes)	Variable size data (maximum 80 bytes)	*	Checksum (2 bytes)	Terminator (CR LF)

Here:

- Header ($): Start of the packet
- Talker ID: Identify the talker (GP, GN, and so on)
- MessageID: Type of message (RMC, GGA, and so on)
- Variable size data: Data fields depending on the Message ID
- Checksum: Checksum bytes are followed after asterisk *. The checksum is a two-digit hexadecimal number calculated by the bit-wise exclusive OR of ASCII codes of all characters between $ and *, not inclusive.
- Terminator: CR LF, indicating end of the message

This figure shows the corresponding entries for the various parameters in the block and the output in the Simulink model. The **Checksum size** in the block mask is specified as 3, which includes asterisk * and the 2-digit checksum value. The last row, `GPSData`, denotes the variable sized data with default values for its **Output field length**, **Output field data type**, and **Byte order** parameters, which cannot be edited. The variable size output (`GPSData` in the block) contains actual data appended with zeros. The additional output port that gets enabled, **Variable field length**, can be used to determine the actual size of the field.



The custom algorithm required for checksum validation is saved in another file, `validateData.m`, and it is called using the **File path** field in the block. The logic defined for the above case, in `validateData.m`, looks like this:

```
function isValid = validateData(packet, checksumBytes)
%#codegen
isValid = false;
calculated_checksum = uint8(0);
% Checksum bytes from NMEA starts with '*', followed by 2 bytes which is
% XOR of all bytes in hexadecimal format.
if ~checksumBytes(1) == uint8('*')
    return;
end
for i = 1:numel(packet)
    calculated_checksum = bitxor(calculated_checksum ,packet(i));
end
if checksumBytes(2:3) == dec2hex(calculated_checksum)
    isValid = true;
end
end
```

Note If you specify that the data contains variable-sized field, the **Terminator** selection is mandatory (select a value except `none`).

Decode Comma-separated Variable

To decode comma-separated variable, you enable the **Parse comma separated variables** parameter.

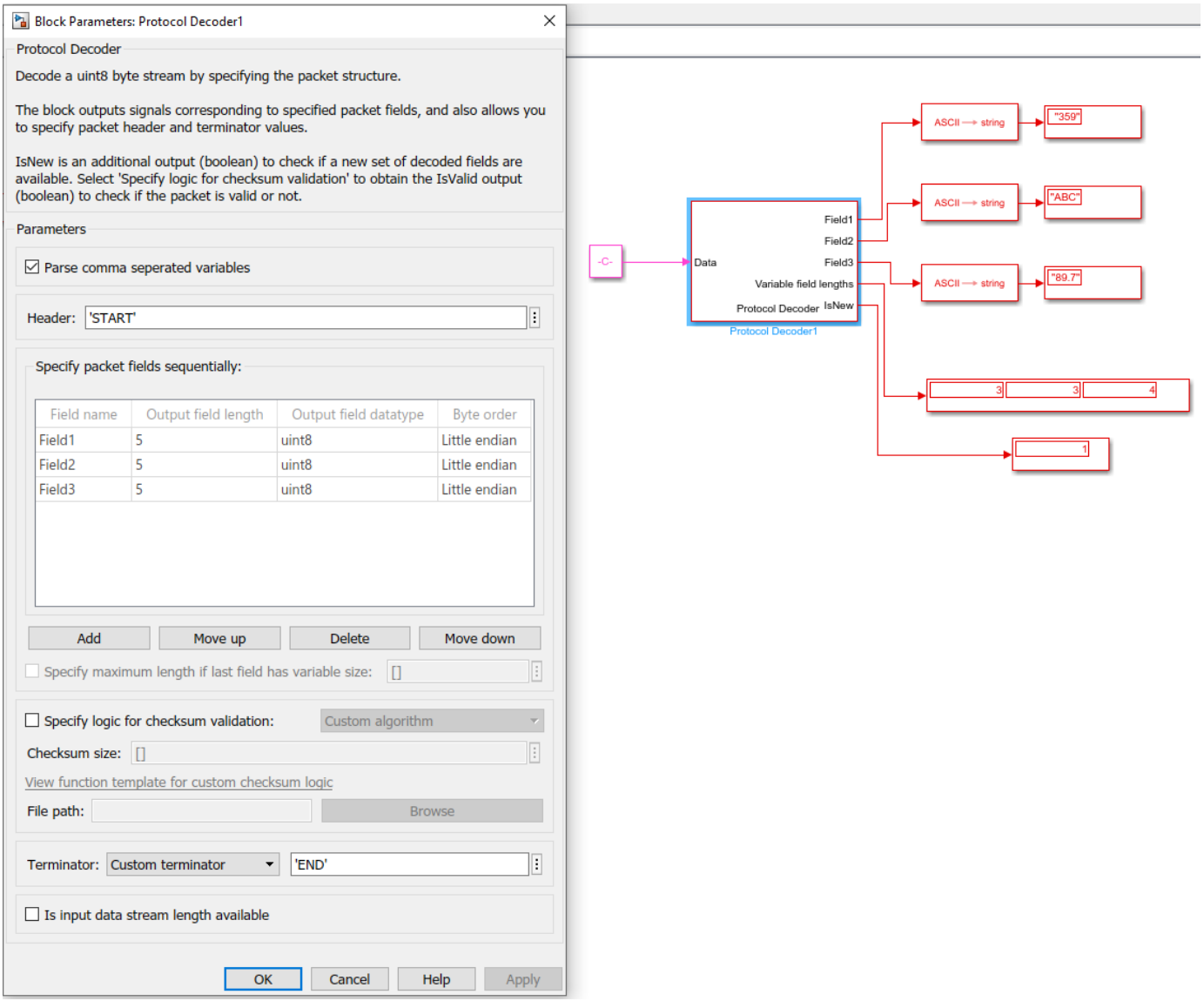
The input packet can be partially received in over multiple sample times. The Protocol Decoder block has the capability to buffer such partial data until the complete packet is received.

The fields in the packet can be variable sized. In the **Output field length** column, ensure that the maximum possible length of the field is given. The block outputs fixed size fields based on this length.

For example, consider the packet below:

Header (START)	Field1 Maximum size: 5 bytes	,	Field2 Maximum size: 5 bytes	,	Field3 Maximum size: 5 bytes	Terminator (END)

This figure shows the corresponding entries for the various parameters in the Protocol Decoder block and the output in the Simulink model.



The block follows this sequence for decoding the data:

1. Check for a packet which has the specified header and terminator and does not exceed the maximum packet size.

   The maximum packet size is determined by adding these:

   Sum of `Output field length` + `Checksum size` (if applicable) + number of fields -1.

2. Validate the packet once the packet satisfying above condition is obtained, using the checksum logic (if applicable).
3. Parse the individual fields considering comma as the separator. The additional output port that gets enabled, **Variable field lengths**, can be used to determine the actual size of the fields.

   a. If the n^{th} field has a length that is the same as specified in **Output field length** in the block, the block outputs the value as it is. In this case, the n^{th} element of the **Variable field lengths** is equal to the field length specified.

   b. If the n^{th} field has a length that is less than its specified `Output field length` in the block, the block outputs the actual value appended with zeros. In this case, the n^{th} element of the **Variable field lengths** is equal to the actual field length obtained.

   c. If the n^{th} field has a length that is greater than specified output data length, the block outputs truncated value as per the specified length. In this case, the n^{th} element of the **Variable field lengths** is equal to the actual field length obtained.

Note If you want to parse comma-separated variable, the **Terminator** selection is mandatory (select a value except `none`).

Function Template for Custom Checksum Logic in Protocol Decoder Block

```
function isValid  = validateChecksumByte(payload,checksumByte)
%#codegen
end
```

Create the function as specified above to validate packet using checksum. The function returns a boolean, `isValid`, which determines if the packet is valid. If the `isValid` is false, output field values given by the block are `0`'s. Payload contains all the data bytes excluding header, terminator, and checksum bytes, in `uint8` data type. The checksum bytes (checksumByte) are `uint8` bytes whose size is specified in the **Checksum size** field available in the block mask.

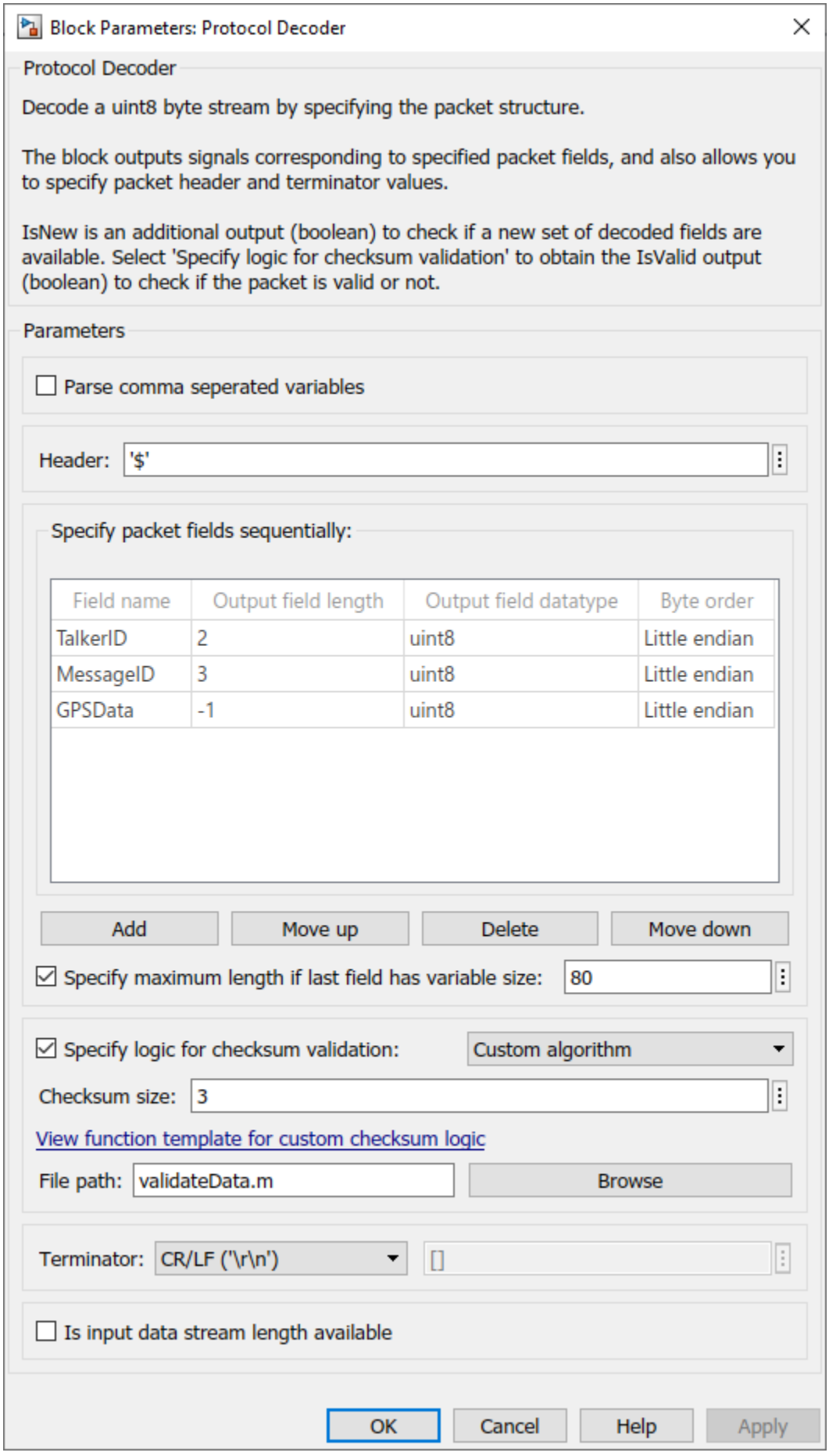
Save the function and specify the path of the function in the **File path** field in the block mask. Ensure that the file is in the MATLAB path.

Example

Consider the NMEA packet with the following structure:

Header ('$')	Talker ID (2 bytes)	Message ID (3 bytes)	Variable size data (maximum 80 bytes)	*	Checksum (2 bytes)	Terminator (CR LF)

You define the block parameters for this case like this:


Block Parameters: Protocol Decoder
Protocol Decoder
Decode a uint8 byte stream by specifying the packet structure.
The block outputs signals corresponding to specified packet fields, and also allows you to specify packet header and terminator values.
IsNew is an additional output (boolean) to check if a new set of decoded fields are available. Select 'Specify logic for checksum validation' to obtain the IsValid output (boolean) to check if the packet is valid or not.
Parameters
Parse comma seperated variables
Header: '$'
Specify packet fields sequentially:
Field name
Output field length
Output field datatype
Byte order
TalkerID
2
uint8
Little endian
MessageID
3
uint8
Little endian
GPSData
-1
uint8
Little endian
Add
Move up
Delete
Move down
Specify maximum length if last field has variable size:
80
Specify logic for checksum validation:
Custom algorithm
Checksum size:
3
View function template for custom checksum logic
File path:
validateData.m
Browse
Terminator:
CR/LF ('\r\n')
[]
Is input data stream length available
OK
Cancel
Help
Apply

Checksum bytes are followed after asterisk *. The checksum is a two-digit hexadecimal number calculated by the bit-wise exclusive OR of ASCII codes of all characters between the $ and *, not inclusive. The **Checksum size** in the block mask is specified as 3, which includes asterisk * and 2-digit checksum value.

Consider the following input (obtained from a GPS receiver) to the Protocol Decoder:

```
[uint8(['$GPVTG,77.52,T,,M,0.004,N,0.008,K,A*06']),uint8([13,10])]
```

The first input to validate the logic is the packet excluding header($), terminator (CR(ASCII-13)(LF - ASCII-10)), and checksum bytes:

```
uint8(['GPVTG,77.52,T,,M,0.004,N,0.008,K,A'])
```

The second input is the checksum bytes *06, which are the last 3 bytes before terminator. In this case, the validation logic can be created like this:

```
function isValid = validateData(packet, checksumBytes)
%#codegen
isValid = false;
calculated_checksum = uint8(0);
% Checksum bytes from NMEA starts with '*', followed by 2 bytes which is
% XOR of all bytes in hexadecimal format.
if ~checksumBytes(1) == uint8('*')
    return;
end
for i = 1:numel(packet)
    calculated_checksum = bitxor(calculated_checksum ,packet(i));
end
if checksumBytes(2:3) == dec2hex(calculated_checksum)
    isValid = true;
end
end
```

Monitor Signals and Tune Parameters

The Protocol Decoder block is available with Simulink Support Package for Arduino Hardware and Embedded Coder Support Package for Texas Instruments C2000 Processors. However, to monitor signals and tune parameters for protocol decoding in Simulink models to be run on TI's C2000-based targets, use the Universal Measurement and Calibration Protocol (XCP)-based External mode simulation.

Version History

Introduced in R2021b

See Also

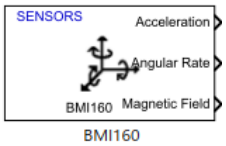
Protocol Encoder

Topics

"Encode and Decode Serial Data Using C2000-based Hardware" on page 2-316

BMI160

Measure linear acceleration, angular rate, and temperature from BMI160 sensor



Libraries:
C2000 Microcontroller Blockset / Sensors

Description

The BMI160 block outputs the values of linear acceleration and angular rate along x-, y- and z- axes as measured by the BMI160 sensor connected to TI's C2000 board. The block also outputs the temperature as read by the BMI160 sensor. If you connect the BMM150 as a secondary sensor to BMI160, the BMI160 block also outputs magnetic field along x-, y- and z- axes as measured by the BMM150 sensor.

The block supports Single tap, Double tap, High g detection, Any motion, Slow motion, Flat detection, and Data ready interrupts.

Interrupt source output is a `1`-by-`7` vector with elements corresponding to the source of single tap, double tap, high g detection, any motion, slow motion, flat detection, and data ready interrupts respectively. `1` indicates that it is the source, `0` indicates that it is not a source, and `-1` indicates that the interrupt is not active.

Tap source output is a `1`-by-`4` vector, with the first three elements indicating whether the single tap / double tap occurred in X, Y and Z axis, and the fourth element indicating whether it is on positive or negative direction of the axis. High g source output is a `1`-by-`4` vector, with the first three elements indicating whether the High g interrupt occurred in X,Y and Z axis and the fourth element indicating whether it is on positive or negative direction of the axis. Any motion source output is a `1`-by-`4` vector which shows on which of the X,Y and Z axis did Any motion interrupt occur and whether it is on positive or negative direction of the axis. First byte denotes whether the event occurred on X axis, second byte denotes whether the event occurred on Y axis, third byte denotes whether the event occurred on Z axis and fourth byte denotes the direction. For direction byte `0` denotes negative side and `1` denotes positive and for other bytes `1` denotes that axis is the source and `0` denotes that axis is not the source.

When interrupts are enabled use external interrupt block and develop respective function call subsystem.

Ports

Output

Acceleration — Linear acceleration measured by BMI160 sensor
row vector

Linear acceleration (in m/s^2) measured by BMI160 sensor connected to C2000 board, along the x-, y- and z- axes, specified as a row vector [x,y,z].

Dependencies

This output port appears only if you select the **Acceleration (m/s^2)** parameter.

Data Types: `double`

Angular Rate — Angular rate measured by BMI160 sensor
row vector

Angular rate (in rad/s) measured by BMI160 sensor connected to C2000 board, along the x-, y- and z-axes, specified as a row vector [x,y,z].

Dependencies

This output port appears only if you select the **Angular rate (rad/s)** parameter.

Data Types: `double`

Magnetic Field — Magnetic field strength measured by a secondary BMM150 sensor
row vector

Magnetic field strength (in µT) measured by a BMM150 sensor that is connected as a secondary sensor to BMI160 sensor, along the x-, y- and z- axes, specified as a row vector [x,y,z].

Dependencies

This output port appears only if you select the **Enable secondary magnetometer** and **Magnetic Field (µT)** parameters.

Data Types: `double`

Temperature — Temperature measured by BMI160 sensor
scalar

Temperature (in ℃) measured by BMI160 sensor connected to C2000 board.

Dependencies

This output port appears only if you select the **Temperature (°C)** parameter.

Data Types: `double`

Acceleration Status — Status of acceleration value
`0` | `1`

Status of acceleration `0` indicates that the data read is new and `1` indicates that the data read is not new.

Dependencies

This output port appears only if you select the **Acceleration Status** parameter.

Data Types: `int8`

Angular Rate Status — Status of angular rate value
`0` | `1`

Status of angular rate `0` indicates that the data read is new and `1` indicates that the data read is not new.

Dependencies

This output port appears only if you select the **Angular Rate Status** parameter.

Data Types: `int8`

Magnetic Field Status — Status of magnetic field value
`0 | 1`

Status of magnetic field `0` indicates that the data read is new and `1` indicates that the data read is not new.

Dependencies

This output port appears only if you select the **Magnetic Field Status** parameter.

Data Types: `int8`

Interrupt Source — Status of interrupt source
`0 | 1 | -1`

Interrupt source is a 1-by-7 vector which represents Single tap, Double tap, High g detection, Any motion, Slow motion, Flat detection, and Data ready interrupts.

Status of interrupt source `1` indicates that the interrupt enabled and it is the source of the generated interrupt, `0` indicates that the interrupt is enabled and it is not the source of the generated interrupt, and `-1` indicates that the interrupt is not enabled.

Dependencies

This output port appears only if you select one of these parameters.

- **Single tap**
- **Double tap**
- **High g detection**
- **Any motion**
- **Flat detection**
- **Data ready**

Data Types: `double`

Tap event Source — Status of tap event source
row vector

Tap event source is a 1-by-4 vector which represents Single tap or Double tap interrupt. The first three elements indicates if the interrupt occurred in X, Y and Z axis, and the fourth element indicates if it is on positive or negative direction of the axis. For direction, byte `0` denotes negative side, `1` denotes positive side and for other bytes `1` denotes that axis is the source and `0` denotes that axis is not the source.

Dependencies

This output port appears only if you select the **Interrupt source** parameter.

Data Types: `int8`

High g event Source — Status of high g event source
row vector

High g event source is a 1-by-4 vector which represents High g interrupt. The first three elements indicates if the interrupt occurred in X, Y and Z axis, and the fourth element indicates if it is on positive or negative direction of the axis. For direction, byte `0` denotes negative side, `1` denotes positive side and for other bytes `1` denotes that axis is the source and `0` denotes that axis is not the source.

Dependencies

This output port appears only if you select the **Interrupt source** parameter.

Data Types: `int8`

Any motion event Source — Status of any motion event source
row vector

Any motion event source is a 1-by-4 vector which represents Any motion interrupt. The first three elements indicates if the interrupt occurred in X, Y and Z axis, and the fourth element indicates if it is on positive or negative direction of the axis. For direction, byte `0` denotes negative side, `1` denotes positive side and for other bytes `1` denotes that axis is the source and `0` denotes that axis is not the source.

Dependencies

This output port appears only if you select the **Interrupt source** parameter.

Data Types: `int8`

Parameters

I2C module — Module for communication
`I2C_A` (default)

The I2C module to be used for communication to the BMI160 sensor. The number of I2C modules supported varies across different C2000 processors. You can find the supported I2C modules corresponding to the processor (which you selected for the Hardware Board parameter in the Simulink model) by opening the Configuration Parameters dialog box and checking the I2C specific tabs under **Target hardware resources**.

BMI160 I2C address — I2C address of BMI160 sensor
`0x69` (default) | `0x68`

The I2C address used by BMI160 sensor communicating with the C2000 processor. The default parameter value (`0x69`) corresponds to the value mentioned in the `Schematics` section of the BOOSTXL-SENSORS BoosterPack™ Plug-in Module User's Guide.

Enable secondary magnetometer — Enable read data from BMM150 connected as a secondary sensor to BMI160
`on` (default) | `off`

If this option is selected, the block can read magnetometer data from a BMM150 sensor that is connected as a secondary sensor to BMI160.

BMM150 I2C address — I2C address of BMM150 sensor
`0x13` (default) | `0x10` | `0x11` | `0x12`

The I2C address used by BMM150 sensor connected as a secondary sensor to BMI160 sensor. The default parameter value (`0x13`) corresponds to the value mentioned in the `Schematics` section of the BOOSTXL-SENSORS BoosterPack™ Plug-in Module User's Guide.

Dependencies

This parameter appears only if you select the **Enable secondary magnetometer** parameter.

Select outputs

Acceleration (m/s²) — Set output port for reading acceleration
`on` (default) | `off`

Select this parameter to set **Acceleration** as one of the output ports.

Angular rate (rad/s) — Set output port for reading angular rate
`on` (default) | `off`

Select this parameter to set **Angular Rate** as one of the output ports.

Magnetic Field (µT) — Set output port for reading magnetic field
`on` (default) | `off`

Select this parameter to set **Magnetic Field** as one of the output ports.

Temperature (°C) — Set output port for reading temperature
`off` (default) | `on`

Select this parameter to set **Temperature** as one of the output ports.

Acceleration status — Set output port for obtaining acceleration status
`off` (default) | `on`

Select this parameter to set **Acceleration Status** as one of the output ports.

Angular rate status — Set output port for obtaining angular rate status
`off` (default) | `on`

Select this parameter to set **Angular Rate Status** as one of the output ports.

Magnetic Field Status — Set output port for obtaining magnetic field status
`off` (default) | `on`

Select this parameter to set **Magnetic Field Status** as one of the output ports.

Interrupt source — Set output port for obtaining interrupt source status
`off` (default) | `on`

Select this parameter to set **Interrupt source** as one of the output ports.

Tap source — Set output port for obtaining tap event status
`off` (default) | `on`

Select this parameter to set **Tap event source** as one of the output ports.

High g source — Set output port for obtaining high g event status
`off` (default) | `on`

Select this parameter to set **High g event source** as one of the output ports.

Any motion source — Set output port for obtaining any motion event status
`off` (default) | `on`

Select this parameter to set **Any motion event source** as one of the output ports.

Acceleration (m/s^2) — Set output port for reading acceleration
`on` (default) | `off`

Select this parameter to set **Acceleration** as one of the output ports.

Angular rate (rad/s) — Set output port for reading angular rate
`on` (default) | `off`

Select this parameter to set **Angular Rate** as one of the output ports.

Magnetic Field (µT) — Set output port for reading magnetic field
`on` (default) | `off`

Select this parameter to set **Magnetic Field** as one of the output ports.

Temperature (°C) — Set output port for reading temperature
`off` (default) | `on`

Select this parameter to set **Temperature** as one of the output ports.

Acceleration status — Set output port for obtaining acceleration status
`off` (default) | `on`

Select this parameter to set **Acceleration Status** as one of the output ports.

Angular rate status — Set output port for obtaining angular rate status
`off` (default) | `on`

Select this parameter to set **Angular Rate Status** as one of the output ports.

Magnetic Field Status — Set output port for obtaining magnetic field status
`off` (default) | `on`

Select this parameter to set **Magnetic Field Status** as one of the output ports.

Data type — Output data type for values from BMI160 sensor
`single` (default) | `double`

Specify the output data type for the values read from BMI160 sensor. The default data type for TI's C2000 processors is `single`. Use this parameter to change the values to `double`, if required.

Sample time — Time interval to read data
`-1` (default) | positive integer

Specify how often this block reads the data from the BMI160 sensor. When you set this parameter to -1, Simulink determines the best sample time for the block based on the block context within the model.

Accelerometer Settings

Accelerometer range — Full scale for measuring linear acceleration
`±2g` (default) | `±4g` | `±8g` | `±16g`

Select the full scale for measuring linear acceleration (the range of acceleration that the sensor needs to measure).

Accelerometer output data rate — Rate at which accelerometer data is sampled
`12.5 Hz` (default) | `25 Hz` | `50 Hz` | `100 Hz` | `200 Hz` | `400 Hz` | `800 Hz` | `1600 Hz`

Select the output data rate at which accelerometer data is sampled, which also determines the bandwidth.

Enable low pass filter — Enable low pass filter to read accelerometer data from BMI160 sensor
`off` (default) | `on`

Select this option to enable the low-pass filter for the acceleration values read from BMI160 sensor.

Accelerometer filter mode — Select the filter mode for low pass filter for accelerometer values
`Normal` (default) | `OSR2` | `OSR4`

Select the filter mode for low pass filter for accelerometer values. The 3dB Cutoff frequency of the accelerometer depends on the value of this parameter and the ODR that you selected using the **Accelerometer output data rate** parameter.

Gyroscope settings

Gyroscope range — Full scale for measuring angular rate
`125 dps` (default) | `250 dps` | `500 dps` | `1000 dps` | `2000 dps`

Select the full scale for measuring angular rate (the range of angular rate that the sensor needs to measure).

Gyroscope output data rate — Rate at which gyroscope data is sampled
`25 Hz` (default) | `50 Hz` | `100 Hz` | `200 Hz` | `400 Hz` | `800 Hz` | `1600 Hz` | `3200 Hz`

Select the output data rate at which gyroscope data is sampled, which also determines the bandwidth.

Enable low pass filter — Enable low pass filter to read gyroscope data from BMI160 sensor
`off` (default) | `on`

Select this option to enable the low-pass filter for the angular rate values read from BMI160 sensor.

Gyroscope filter mode — Select the filter mode for low pass filter for gyroscope values
`Normal` (default) | `OSR2` | `OSR4`

Select the filter mode for low pass filter for gyroscope values. The 3dB Cutoff frequency of the gyroscope depends on the value of this parameter and the ODR that you selected using the **Gyroscope output data rate** parameter.

Magnetometer settings

Magnetometer output data rate — Rate at which magnetometer data is sampled
`25 Hz` (default) | `0.78125 Hz` | `1.5625 Hz` | `3.125 Hz` | `6.25 Hz` | `12.5 Hz` | `50 Hz` | `100 Hz` | `200 Hz` | `400 Hz` | `800 Hz`

Select the output data rate at which magnetometer data is sampled.

Generate interrupts on

Note Selecting one of these parameters (**Single tap**, **Double tap**, **High g detection**, **Any motion**, **Slow motion**, **Flat detection**, **Data ready**) overrides the availability of other status values at the output. The selection of parameters for other status outputs (`Acceleration status`, `Angular rate status`, and `Magnetic field status`) is disabled.

Single tap — Enable interrupt when single tap is triggered
`off` (default) | `on`

If this option is selected, an interrupt is generated on pin INT1 or INT2 of the sensor when single tap is triggered and following conditions are valid.

- The tap occurs during the Shock Time threshold.
- No tap occurs during the Quiet Time threshold.

Quiet Time threshold — Quiet time duration
`30 ms` (default) | `20 ms`

Select the quiet time threshold for generating single tap or double tap interrupt. The tap must not occur during the specified quiet time threshold for generating single tap or double tap interrupt.

Dependencies

This parameter appears only if you select the **Single tap** parameter or **Double tap** parameter.

Shock Time threshold — Shock time duration
`50 ms` (default) | `75 ms`

Select the shock time threshold for single tap or double tap interrupt. The tap must occur during the specified shock time threshold for generating single tap or double tap interrupt.

Dependencies

This parameter appears only if you select **Single tap** parameter or **Double tap** parameter.

Amplitude threshold — Amplitude threshold for detecting a tap
`0.1` (default)

Specify the amplitude threshold value ranging from 0.03125g - 1.96875g for detecting a single tap or double tap. When the value crosses the specified amplitude threshold value, a tap occurs. Amplitude threshold has minimum and maximum range which depends on the accelerometer range.

Dependencies

This parameter appears only if you select the **Single tap** parameter or **Double tap** parameter.

Interrupt generate pin — Pin to generate single tap interrupt
`INT1` (default) | `INT2`

Select the pin on which the single tap interrupt, which can be used by the C2000 board, gets generated on the sensor.

Dependencies

This parameter appears only if you select the **Single tap** parameter.

Double tap — Enable interrupt when double tap is triggered
`off` (default) | `on`

If this option is selected, an interrupt is generated on pin INT1 or INT2 of the sensor when double tap is triggered and following conditions are valid.

- The first tap occurs during the Shock Time threshold.
- No tap occurs during the Quiet Time threshold.
- The second tap occurs during the Duration Time threshold.

Duration Time threshold — Time threshold for second tap
`50 ms` (default) | `100 ms` | `150 ms` | `200 ms` | `250 ms` | `375 ms` | `500 ms` | `700 ms`

Select the duration time threshold for double tap interrupt. The second tap must occur during the specified duration time threshold for generating double tap interrupt.

Dependencies

This parameter appears only if you select **Double tap** parameter.

Interrupt generate pin — Pin to generate double tap interrupt
`INT1` (default) | `INT2`

Select the pin on which the double tap interrupt, which can be used by the C2000 board, gets generated on the sensor.

Dependencies

This parameter appears only if you select the **Double tap** parameter.

High g detection — Enable interrupt when high g detection is triggered
`off` (default) | `on`

If this option is selected, an interrupt is generated on pin INT1 or INT2 of the sensor when high g interrupt is detected.

Time threshold(2.5 ms to 640 ms) — Time duration for high g detection
`2.5` (default) |

Specify the time threshold value ranging from 2.5 ms to 640 ms for high g interrupt. The high g detection must occur during the specified time threshold for generating high g interrupt.

Dependencies

This parameter appears only if you select **High g detection** parameter.

Amplitude threshold — Amplitude threshold for detecting high g interrupt
`0.1` (default)

Specify the amplitude threshold value ranging from 0.00391g - 1.99546g for detecting a high g interrupt. When the value crosses the specified amplitude threshold value, high g interrupt is detected. Amplitude threshold has minimum and maximum range which depends on the accelerometer range.

Dependencies

This parameter appears only if you select the **High g detection** parameter.

Interrupt generate pin — Pin to generate high g interrupt
`INT1` (default) | `INT2`

Select the pin on which the high g interrupt, which can be used by the C2000 board, gets generated on the sensor.

Dependencies

This parameter appears only if you select the **High g detection** parameter.

Any motion — Enable interrupt when any motion interrupt is triggered
`off` (default) | `on`

If this option is selected, an interrupt is generated on pin INT1 or INT2 of the sensor when any motion interrupt is detected.

Time threshold — Time duration for any motion detection
`1` (default) | `2` | `3` | `4`

Select the time threshold value for any motion interrupt. The any motion detection must occur during the specified time threshold.

Dependencies

This parameter appears only if you select **Any motion** parameter.

Amplitude threshold — Amplitude threshold for detecting any motion interrupt
`0.1` (default)

Specify the amplitude threshold value ranging from 0.00195g - 0.999g for detecting any motion interrupt. When the value crosses the specified amplitude threshold value, any motion interrupt is detected. Amplitude threshold has minimum and maximum range which depends on the accelerometer range.

Dependencies

This parameter appears only if you select **Any motion** parameter.

Interrupt generate pin — Pin to generate any motion interrupt
`INT1` (default) | `INT2`

Select the pin on which the any motion interrupt, which can be used by the C2000 board, gets generated on the sensor.

Dependencies

This parameter appears only if you select the **Any motion** parameter.

Slow motion — Enable interrupt when slow motion interrupt is triggered
`off` (default) | `on`

If this option is selected, an interrupt is generated on pin INT1 or INT2 of the sensor when slow motion interrupt is detected.

Time threshold — Time duration for slow motion detection
`1` (default) | `2` | `3` | `4`

Select the time threshold value for slow motion interrupt. The slow motion detection must occur during the specified time threshold.

Dependencies

This parameter appears only if you select **Slow motion** parameter.

Amplitude threshold — Amplitude threshold for detecting slow motion interrupt
`0.1` (default)

Specify the amplitude threshold value ranging from 0.00195g - 0.999g for detecting slow motion interrupt. When the value crosses the specified amplitude threshold value, slow motion interrupt is detected.

Dependencies

This parameter appears only if you select **Slow motion** parameter.

Interrupt generate pin — Pin to generate slow motion interrupt
`INT1` (default) | `INT2`

Select the pin on which the slow motion interrupt, which can be used by the C2000 board, gets generated on the sensor.

Dependencies

This parameter appears only if you select the **Slow motion** parameter.

Flat detection — Enable interrupt when flat detection interrupt is triggered
`-1` (default) | positive integer

If this option is selected, an interrupt is generated on pin INT1 or INT2 of the sensor when flat detection interrupt is detected.

Theta threshold (0.7˚ to 44.8˚) — Theta threshold for detecting flat detection interrupt
`5` (default)

Specify the amplitude threshold value ranging from 0.7° - 44.8° for detecting flat detection interrupt. When the value crosses the specified amplitude threshold value, flat detection interrupt is detected.

Dependencies

This parameter appears only if you select **Flat detection** parameter.

Time threshold — Time duration for flat detection interrupt
`640` (default) | `0` | `1280` | `2560`

Select the time threshold value for flat detection interrupt. The flat detection interrupt must occur during the specified time threshold.

Dependencies

This parameter appears only if you select **Flat detection** parameter.

Interrupt generate pin — Pin to generate flat detection interrupt
`INT1` (default) | `INT2`

Select the pin on which the flat detection interrupt, which can be used by the C2000 board gets generated on the sensor.

Dependencies

This parameter appears only if you select the **Flat detection** parameter.

Data ready — Enable interrupt when data is ready
`off` (default) | `on`

If this option is selected, an interrupt is generated on pin INT1 or INT2 of the sensor when data is ready, allowing you to trigger other subsystems to perform any action.

Interrupt generate pin — Pin to generate data ready interrupt
`INT1` (default) | `INT2`

Select the pin on which the data ready interrupt, which can be used by the C2000 board, gets generated on the sensor.

Dependencies

This parameter appears only if you select the **Enable data ready interrupt** parameter.

Data type — Output data type for values from BMI160 sensor
`single` (default) | `double`

Specify the output data type for the values read from BMI160 sensor. The default data type for C2000 board is `single`. Use this parameter to change the values to `double`, if required.

Sample time — Time interval to read data
`-1` (default) | positive integer

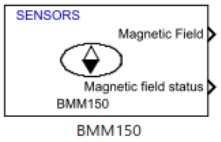
Specify how often this block reads the data from the BMI160 sensor. When you set this parameter to -1, Simulink determines the best sample time for the block based on the block context within the model.

Version History

Introduced in R2021b

BMM150

Measure magnetic field from BMM150 sensor



Libraries:
C2000 Microcontroller Blockset / Sensors

Description

The BMM150 block outputs the values of magnetic field along x-, y- and z- axes as measured by the BMM150 sensor connected to C2000 board.

Ports

Output

Magnetic Field — Magnetic field strength measured by a BMM150 sensor
row vector

Magnetic field strength (in µT) measured by a BMM150 sensor connected to C2000 board, along the x-, y- and z- axes, specified as a row vector [x,y,z].

Data Types: `double`

Magnetic Field Status — Status of magnetic field value
`0` | `1`

Status of magnetic field `0` indicates that the data read is new and `1` indicates that the data read is not new.

Dependencies

This output port appears only if you select the **Status** parameter.

Data Types: `int8`

Parameters

I2C module — Module for communication
`I2C_A` (default)

The I2C module to be used for communication to the BMM150 sensor. The number of I2C modules supported varies across different C2000 processors. You can find the supported I2C modules corresponding to the processor (which you selected for the Hardware Board parameter in the Simulink model) by opening the Configuration Parameters dialog box and checking the I2C specific tabs under **Target hardware resources**.

I2C address — I2C address of BMM150 sensor
`0x13` (default) | `0x10` | `0x11` | `0x12`

The I2C address used by BMM150 sensor communicating with the C2000 processor. The default parameter value (`0x13`) corresponds to the value mentioned in the `Schematics` section of the BOOSTXL-SENSORS BoosterPack™ Plug-in Module User's Guide.

Status — Set output port for obtaining magnetic field status
`on` (default) | `off`

Select this parameter to set **Magnetic Field Status** as one of the output ports.

Preset value — Preset value for the operating condition of BMM150 sensor
`Low power` (default) | `Regular` | `Enhanced` | `High accuracy`

Specify the preset value for the operating condition of BMM150 sensor while reading the values. The selection of one of the four preset values affects the electrical characteristics (like supply current) and the required output accuracy with respect to ODR and output noise.

Data type — Output data type for values from BMM150 sensor
`single` (default) | `double`

Specify the output data type for the values read from BMM150 sensor. The default data type for TI's C2000 processors is `single`. Use this parameter to change the values to `double`, if required.

Sample time — Time interval to read data
`-1` (default) | positive integer

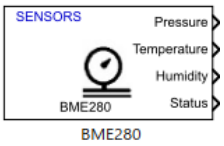
Specify how often this block reads the data from the BMM150 sensor. When you set this parameter to -1, Simulink determines the best sample time for the block based on the block context within the model.

Version History

Introduced in R2021b

BME280

Measure barometric air pressure, relative humidity, and temperature from BME280 sensor



BME280

Libraries:
C2000 Microcontroller Blockset / Sensors

Description

The BME280 block outputs the values of barometric air pressure and relative humidity as measured by the BME280 sensor connected to C2000 board. The block also outputs the temperature as read by the BME280 sensor.

Ports

Output

Pressure — Barometric air pressure measured by BME280 sensor
scalar

Barometric air pressure (in Pascal (Pa)) measured by a BME280 sensor that is connected to C2000 board.

Dependencies

This output port appears only if you select the **Pressure (Pa)** parameter.

Data Types: `double`

Temperature — Temperature measured by BME280 sensor
scalar

Temperature (in ℃) measured by BME280 sensor connected to C2000 board.

Dependencies

This output port appears only if you select the **Temperature (°C)** parameter.

Data Types: `double`

Humidity — Relative humidity measured by BME280 sensor
scalar

Relative humidity (in %) measured by BME280 sensor connected to C2000 board.

Dependencies

This output port appears only if you select the **Humidity (%)** parameter.

Data Types: `double`

Status — Status of read values
`0` | `1`

Status of read values from the BME280 sensor, to indicate if the data read is the new value or not. The **Status** value of `0` indicates that the data read is new and `1` indicates that the data read is not new.

Dependencies

This output port appears only if you select the **Status** parameter.

Data Types: `uint8`

Parameters

I2C module — Module for communication
`I2C_A` (default)

The I2C module to be used for communication to the BME280 sensor. The number of I2C modules supported varies across different C2000 processors. You can find the supported I2C modules corresponding to the processor (which you selected for the Hardware Board parameter in the Simulink model) by opening the Configuration Parameters dialog box and checking the I2C specific tabs under **Target hardware resources**.

I2C address — I2C address of BME280 sensor
`0x77` (default) | `0x76`

The I2C address used by BME280 sensor communicating with the C2000 processor. The default parameter value (`0x77`) corresponds to the value mentioned in the `Schematics` section of the BOOSTXL-SENSORS BoosterPack™ Plug-in Module User's Guide.

Pressure (Pa) — Set output port for reading pressure
`on` (default) | `off`

Select this parameter to set **Pressure** as one of the output ports.

Temperature (°C) — Set output port for reading temperature
`on` (default) | `off`

Select this parameter to set **Temperature** as one of the output ports.

Humidity (%) — Set output port for reading humidity
`on` (default) | `off`

Select this parameter to set **Humidity** as one of the output ports.

Status — Set output port for obtaining status of values read
`off` (default) | `on`

Select this parameter to set **Status** as one of the output ports.

Filter coefficient — Filter coefficient for IIR filter
`0` (default) | `2` | `4` | `8` | `16`

Specify filter coefficient for the IIR filter while reading pressure and temperature values from BME280 sensor. Selecting a non-zero value for filter coefficient helps to improve the step response and remove short-term fluctuations in the measured values.

Stand by time — Standby time to read values
`0.5` (default) | `10` | `20` | `62.5` | `125` | `250` | `500` | `1000`

Specify the standby time (in ms) when the processor stays idle (inactive duration) while reading values from BME280 sensor. The selection of this value affects the total cycle time.

Pressure oversampling factor — Oversampling factor for the measured pressure value
`1` (default) | `2` | `4` | `8` | `16`

Specify the oversampling factor to reduce the noise for the pressure value read from BME280 sensor. This value is a multiplication factor that affects measurement rate and current consumption.

Humidity oversampling factor — Oversampling factor for the measured humidity value
`1` (default) | `2` | `4` | `8` | `16`

Specify the oversampling factor to reduce the noise for the humidity value read from BME280 sensor. This value is a multiplication factor that affects measurement rate and current consumption.

Temperature oversampling factor — Oversampling factor for the measured humidity value
`1` (default) | `2` | `4` | `8` | `16`

Specify the oversampling factor to reduce the noise for the temperature value read from BME280 sensor. This value is a multiplication factor that affects measurement rate and current consumption.

Data type — Output data type for values from BME280 sensor
`single` (default) | `double`

Specify the output data type for the values read from BME280 sensor. The default data type for TI's C2000 processors is `single`. Use this parameter to change the values to `double`, if required.

Sample time — Time interval to read data
`-1` (default) | positive integer

Specify how often this block reads the data from the BME280 sensor. When you set this parameter to -1, Simulink determines the best sample time for the block based on the block context within the model.

Version History

Introduced in R2021b

Read Data from IMU and Environmental Sensors

This example shows how to use C2000™ Microcontroller Blockset to read data from the BMI160 Inertial Measurement Unit (IMU) sensor and BME280 Environmental sensor that are part of the BOOSTXL-SENSORS BoosterPack™ plug-in module.

The example also shows how to read data from another BMM150 geomagnetic sensor connected as a breakout board to F28379D LaunchPad.

Required Hardware

- Texas Instruments™ F28379D LaunchPad
- BOOSTXL-SENSORS BoosterPack™ plug-in module, which includes built-in BMI160 Inertial Measurement Sensor, BMM150 Geomagnetic Sensor and BME280 Environmental Sensor.
- BMM150 Geomagnetic Sensor on an I2C-based board to be connected as a breakout board to F28379D LaunchPad.

Introduction

This example provides three Simulink models that use *TI Delfino F28379D Launchpad* as the Hardware board:

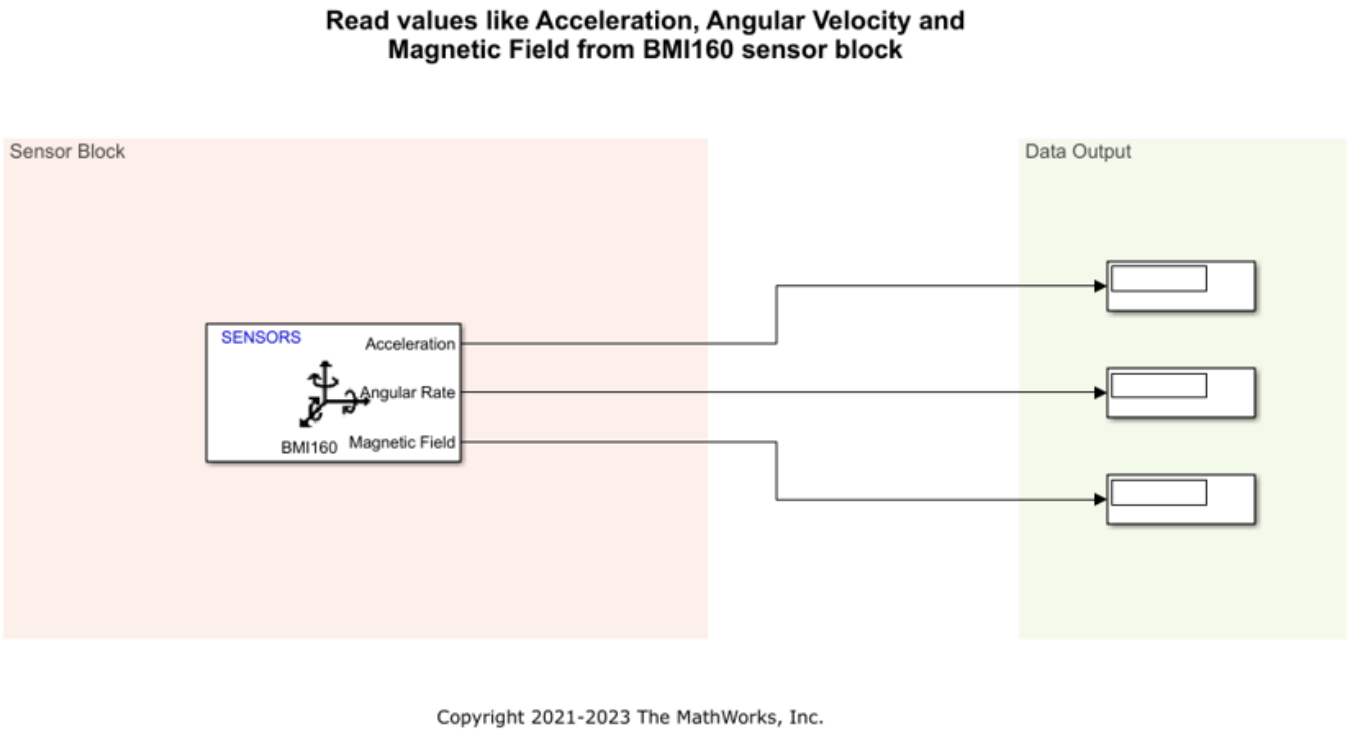
- c28x_i2c_bmi160_sensor helps you to read acceleration, angular rate and magnetic field as measured from both BMI160 and BMM150 sensors in the BOOSTXL-SENSORS module
- c28x_i2c_bme280_sensor helps you to read digital pressure and relative humidity as measured from a BME280 sensor in the BOOSTXL-SENSORS module
- c28x_i2c_bmm150_sensor helps you to read magnetic field as measured from a BMM150 sensor connected as a breakout board to F28379D LaunchPad

Model Configuration for Reading Data from BMI160 Sensor and BME280 Sensor

The model provided in this example for reading data from BMI160 sensor uses the corresponding block, BMI160 provided with the blockset.

To open the model, run this command at the MATLAB prompt:

```
open_system('c28x_i2c_bmi160_sensor');
```

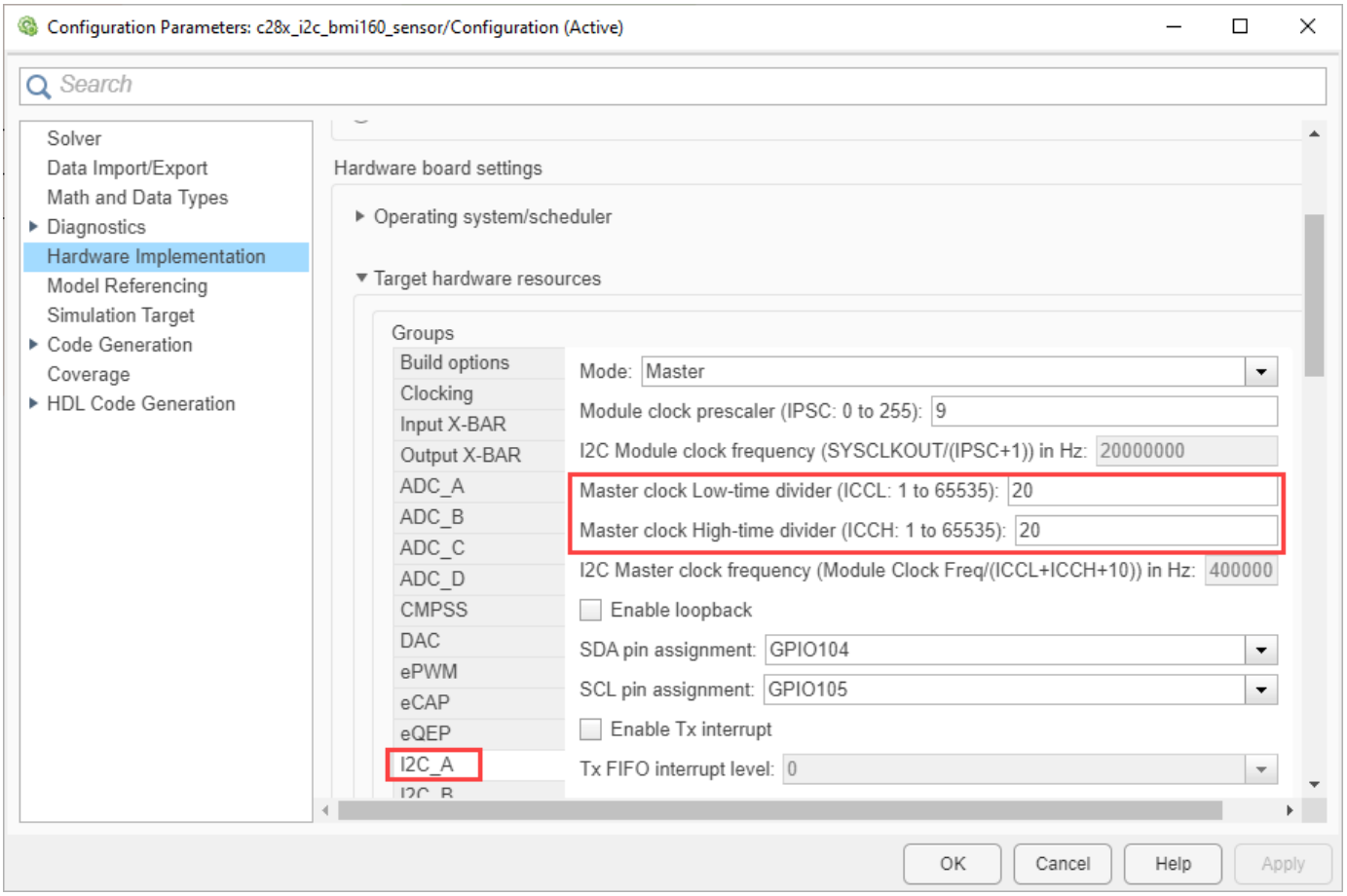


In the BMI160 block in the model, the **I2C module** parameter is set to `I2C_A`. Therefore, to change the clock frequency, if required, change the settings for the same. To do this:

1. Go to Hardware tab, and click **Hardware Settings** to open the Configuration Parameters dialog box.

2. Go to **Hardware Implementation > Target hardware resources**, and select **I2C_A** tab.

3. Edit the values to change the clock frequency as required.

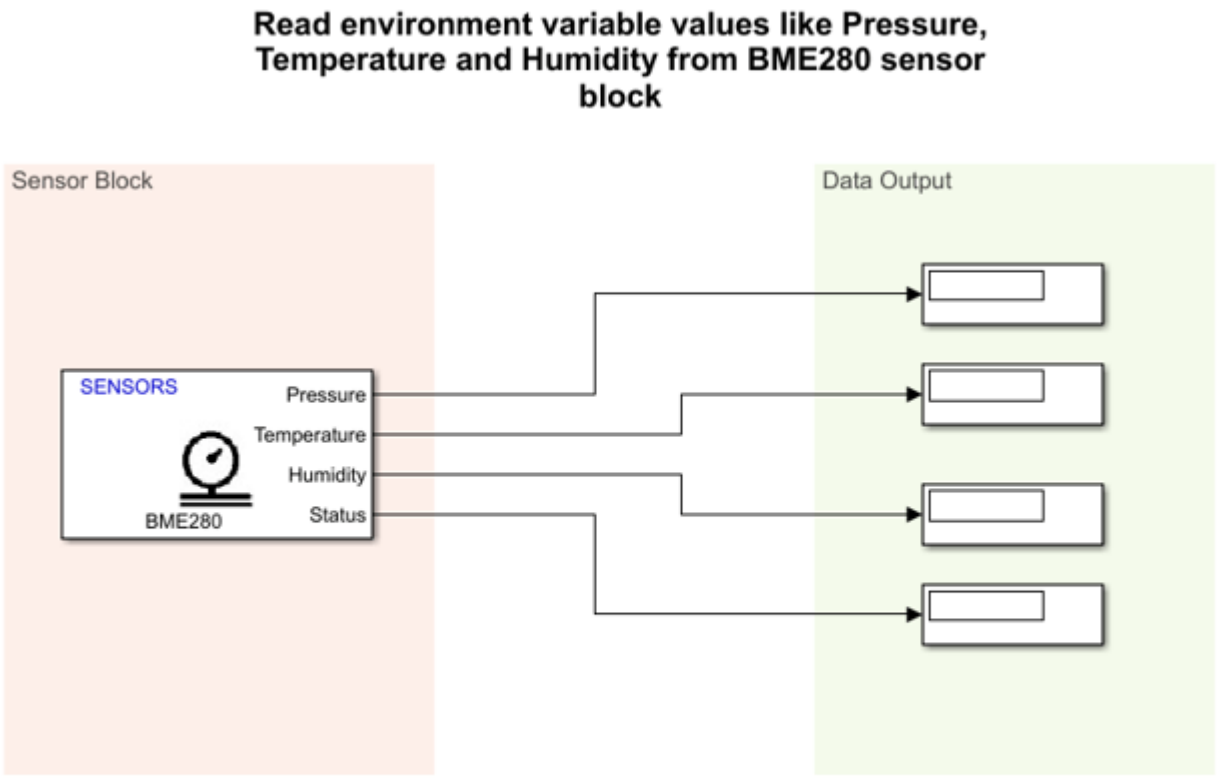


4. Click **Apply and then *OK**.

The other model provided in this example for reading data from BME280 sensor uses the corresponding block, BME280 provided with the blockset.

To open the model, run this command at the MATLAB prompt:

```
open_system('c28x_i2c_bme280_sensor');
```



In the BME280 block in the above model, the **I2C module** parameter is set to `I2C_A`. Therefore, to change the clock frequency, if required, change the settings by following the same steps as described for BMI160.

You can also use the different options under **Advanced settings** inside the BME280 block to change the values for the filtering and sampling factors.

Note: The **I2C address** parameter value selected in each of the blocks (BMI160 and BME280) in the two models corresponds to the information provided in the **Schematics** section of the **BOOSTXL-SENSORS BoosterPack Plug-in Module User's Guide**.

Complete Hardware Connections and Read Data from BMI160 Sensor

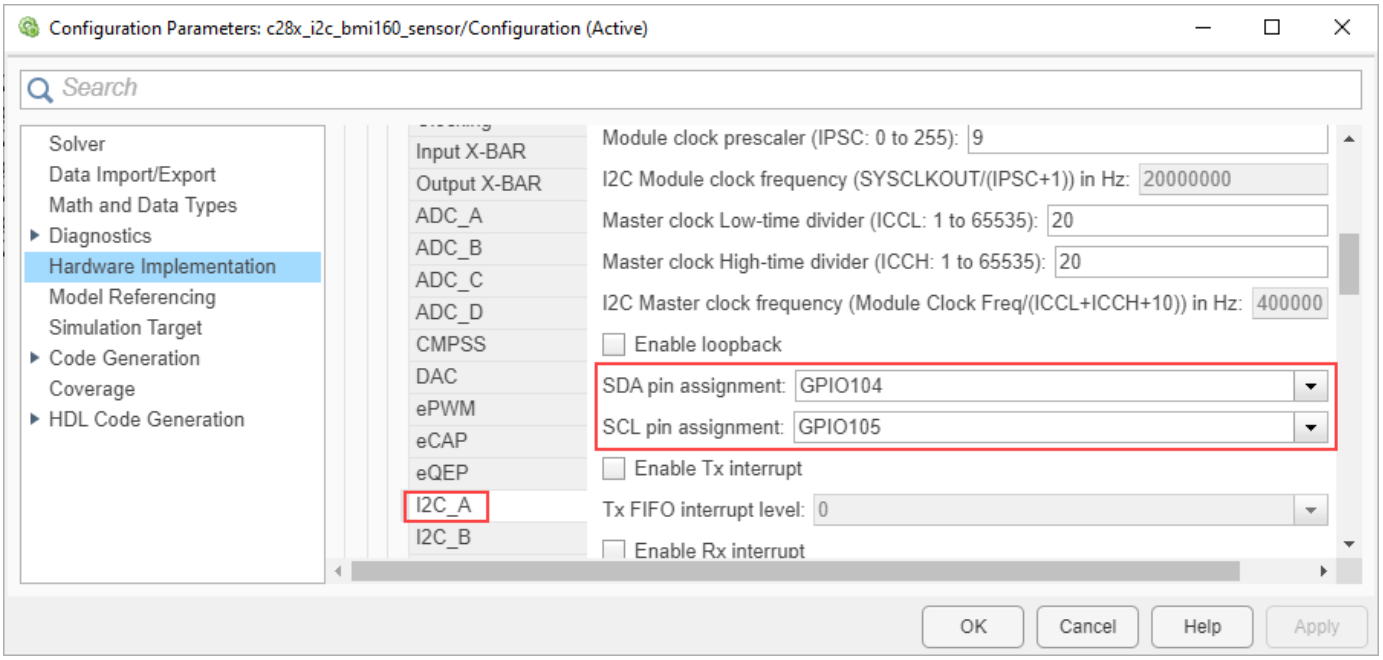
After you complete the configurations settings for the `c28x_i2c_bmi160_sensor` model, perform these steps:

1. Connect the BOOSTXL-SENSORS plug-in module to the F28379D LaunchPad. Connect GPIO104 and GPIO105 pins on the F28379D Launchpad to the J1.10 (SDA) and J1.9 (SCL) pins respectively on the BOOSTXL-SENSORS, and complete the other required connections like VDD and GND. For more details, refer to the the **Schematics** section of the **BOOSTXL-SENSORS BoosterPack Plug-in Module User's Guide**.

2. Connect the F28379D LaunchPad to the host computer.

2. In the **Configuration Parameters** window of `c28x_i2c_bmi160_sensor` model, click **Hardware Implementation** and navigate to **Target hardware resources > External mode**, and set the **Serial port in MATLAB Preferences** parameter to the corresponding COM port to which the Launchpad is connected. The COM port is available at **Device Manager > Ports** (COM & LTP) in Windows.

3. Select appropriate GPIO pins for SDA and SCL in **Hardware Implementation > I2C_A** pane, to communicate with BOOSTXL-SENSORS based on the actual hardware connections from the F28379D Launchpad.



4. In the Hardware tab of Simulink model, click **Monitor & Tune**. You can observe from the Diagnostic Viewer that the code is generated for the model and the host connects to the target after loading the generated executable.

5. Rotate the board about its axis. You can observe that the value displayed in the Display block connected to **Angular Rate** output of the block is changing.

6. Change the orientation of the board. You can observe that the value displayed in the Display block connected to **Acceleration** output of the block is changing.

Complete Hardware Connections and Read Data from BME280 Sensor

After you complete the configurations settings for the `c28x_i2c_bme280_sensor` model, perform the same steps 1 to 4 as described in the previous section to specify the connections and run the model in External mode.

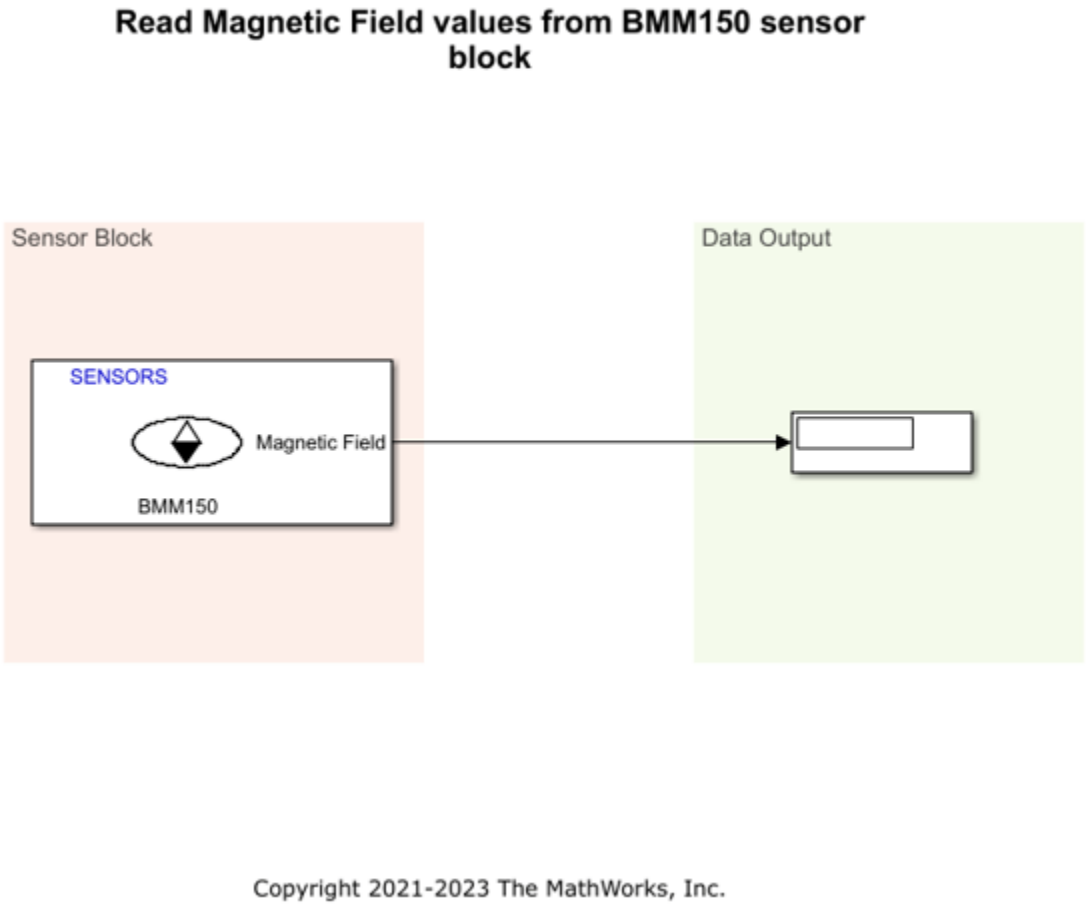
Then, observe the value displayed in the Display block connected to **Pressure**, **Temperature**, and **Humidity** output ports of the block. These values correspond to the current environmental conditions.

Model Configuration for Reading Data from BMM150 Sensor Connected as Breakout Board

The model provided in this example for reading data from BMM150 sensor uses the corresponding block, BMM150 provided with the blockset.

To open the model, run this command at the MATLAB prompt:

```
open_system('c28x_i2c_bmm150_sensor');
```



In the BMM150 block in the above model, the **I2C module** parameter is set to `I2C_A`. Therefore, to change the clock frequency, if required, change the settings by following the same steps as described for BMI160.

You can also use the different options under **Preset value** parameter inside the BMM150 block to specify the optimum operating condition.

The board that you use can be a digital compass sensor based on BMM150 that uses an I2C interface. Refer to the board's specifications for I2C connection to F28379D LaunchPad, and specify the value for **I2C address** parameter accordingly.

Complete Hardware Connections and Read Data from BMM150 Sensor

After you complete the configurations settings for the `c28x_i2c_bmm150_sensor` model, perform these steps:

1. Connect the I2C-based board with the BMM150 sensor, to the F28379D LaunchPad, and complete the other required connections.

2. Connect the F28379D LaunchPad to the host computer.

2. In the **Configuration Parameters** window of `c28x_i2c_bmm150_sensor` model, click **Hardware Implementation** and navigate to **Target hardware resources > External mode**, and set the **Serial port in MATLAB Preferences** parameter to the corresponding COM port to which the Launchpad is connected. The COM port is available at **Device Manager > Ports** (COM & LTP) in Windows.

3. Select appropriate GPIO pins for SDA and SCL in **Hardware Implementation > I2C_A** pane, to communicate with I2C-based interface based on the actual hardware connections from the F28379D Launchpad.

4. In the Hardware tab of Simulink model, click **Monitor & Tune**. You can observe from the Diagnostic Viewer that the code is generated for the model and the host connects to the target after loading the generated executable.

5. Observe the current value in the Display block connected to **Magnetic Field** output of the block. Change the position of the board. You can observe that the value displayed in the Display block is changing.

Encode and Decode Serial Data Using C2000-based Hardware

This example shows how to use C2000™ Microcontroller Blockset to encode and decode serial data with TI's™ C2000-based hardware.

Introduction

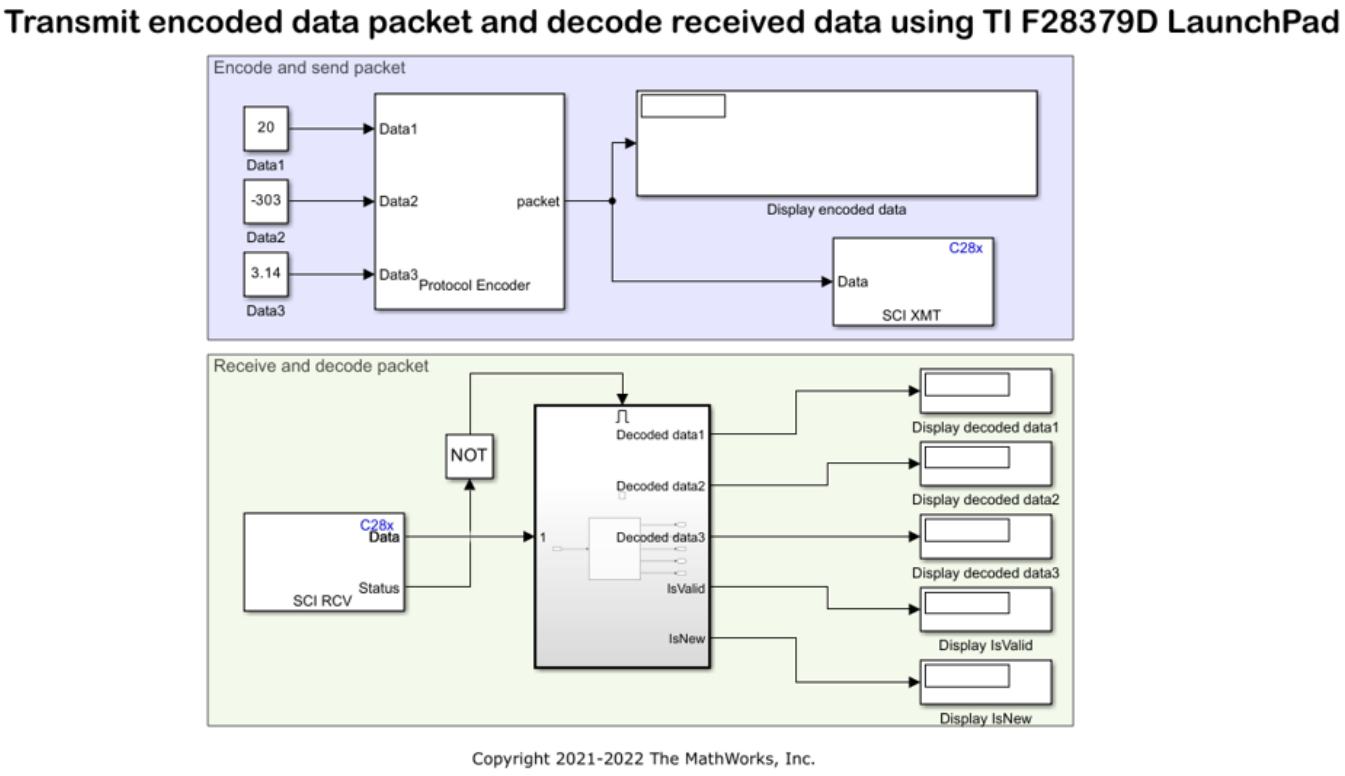
In this example, the Simulink model, performs these actions:

- At the transmission end, multiple fields are encoded into packet using Protocol Encoder block and the resulting uint8 byte stream is transmitted using C28x SCI Transmit block.
- At the Receiving end, the byte stream is received using C28x SCI Receive block and decoded into individual fields using Protocol Decoder block. The status output of SCI Receive block, which indicates a new data is available, is used to trigger the enabled subsystem containing Protocol Decoder block.

In this model, the Tx pin of SCI Module B sends serial data to the Rx pin of SCI Module B of the TI Delfino F28379D LaunchPad.

This model is configured to run in XCP-based External mode. For more information on External mode, see “Signal Monitoring and Parameter Tuning over XCP on Serial”.

```
open_system('c2000_encode_decode_packet');
```



The model provided in this example is preconfigured for the TI Delfino F28379D LaunchPad. You can run this model on any of the TI boards available under the **Hardware board** parameter in the

Simulink model. For more information on how to change the **Hardware board** parameter, see the **Step 2: Configure the Model for Connected Hardware** section of this example.

Required Hardware

To run this example, you must have the following hardware:

- Texas Instruments™ Delfino F28379D LaunchPad
- Connecting wires
- USB cable

Step 1: Connect TX and RX Pins on F28379D Launchpad

1. Connect your TI Delfino F28379D LaunchPad to your computer using the USB cable.

2. Connect the Tx pin of SCI B module (GPIO18 pin) to the Rx pin of SCI B module (GPIO19 pin). This connection is a loopback connection.

Step 2: Configure the Model for Connected Hardware

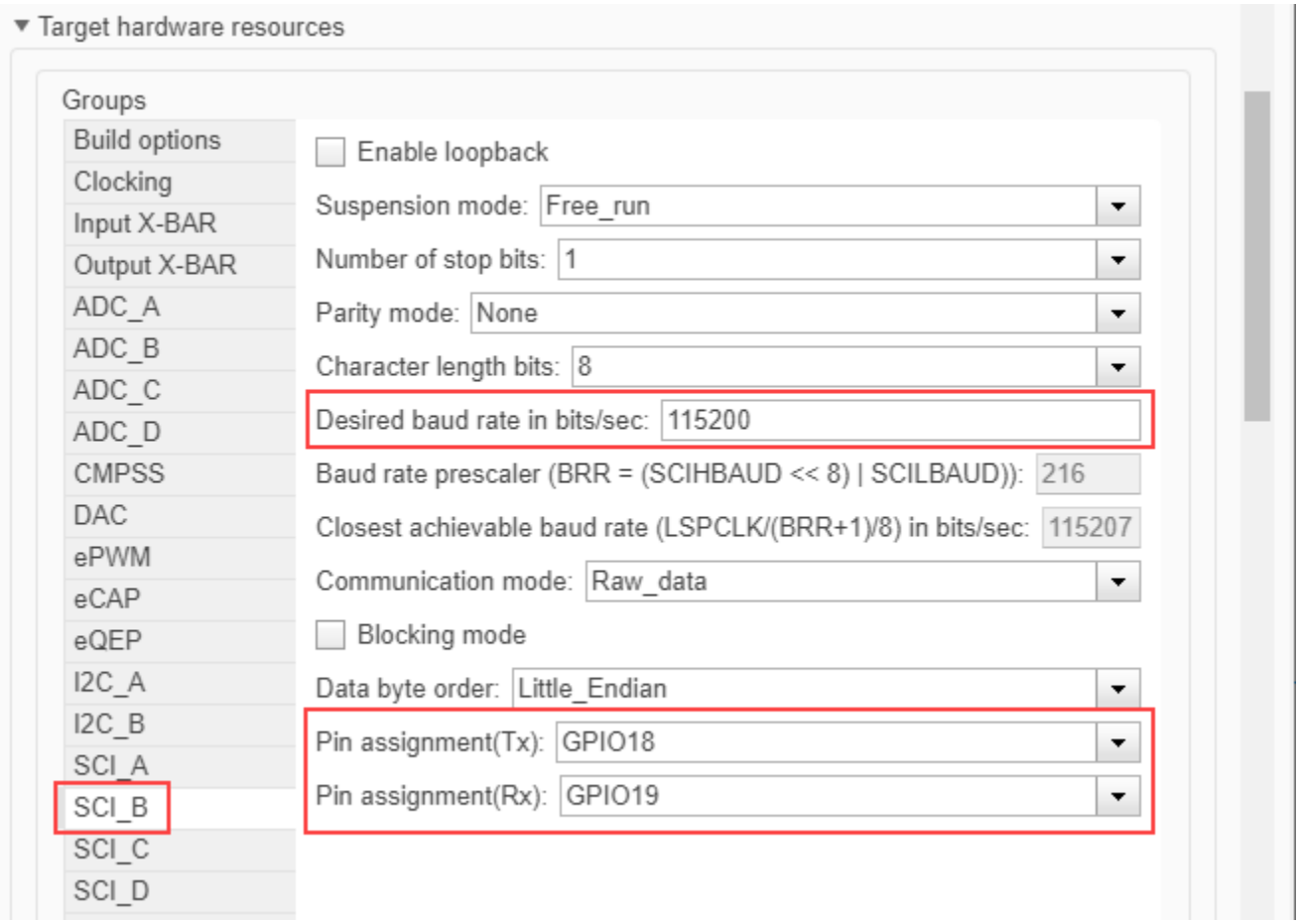
1. Open the Simulink model model. This model is configured to run on XCP-based External mode.

2. To configure the model, click **Hardware Settings** in the **Hardware** tab of the Simulink toolbar.

3. In the Configurations Parameters dialog box, select **Hardware Implementation**.

4. From the **Hardware board** list, select the TI's C2000-based processor that you are using.

5. Under **Target Hardware Resources**, click **SCI_B** tab and configure the properties including baud rate and pin assignments.



If you are using any other GPIO pins for communication, change the **Pin assignment** parameter value selection accordingly.

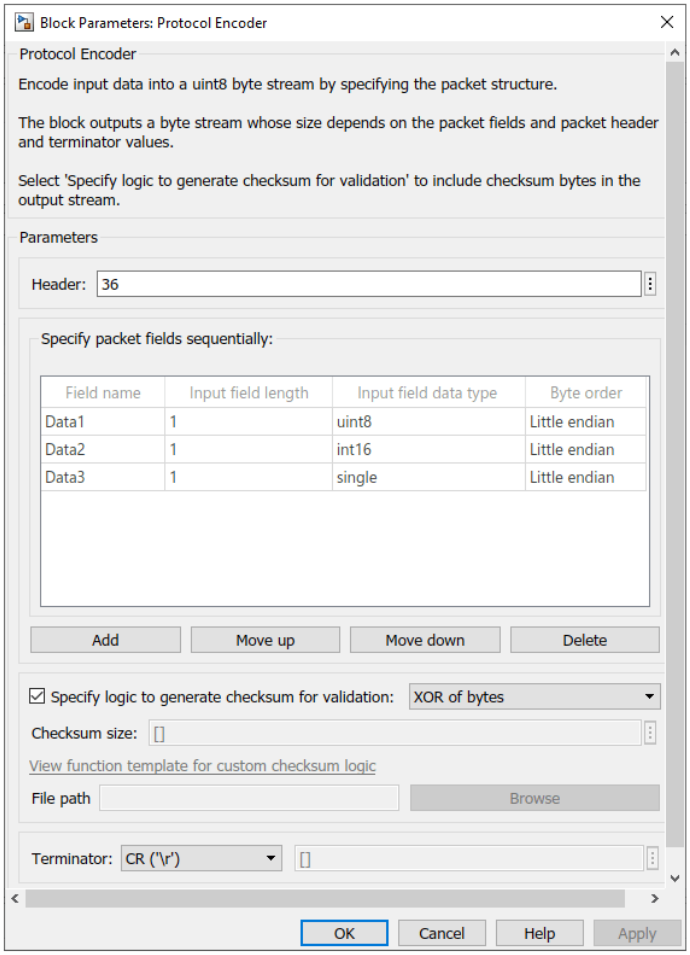
6. Click **Apply**. Click **OK** to close the dialog box.

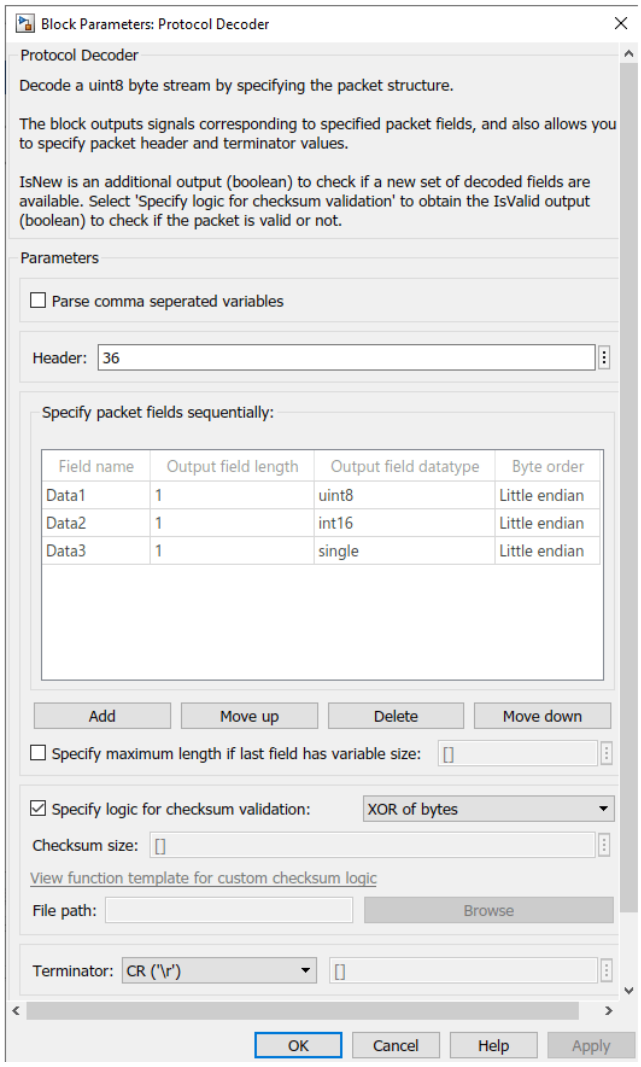
Step 3: Configure Blocks in the Simulink Model

The packet structure used in this example is:

Header (36)	Data1 Data type: uint8	Data2 Data type: int16	Data1 Data type: single	Checksum (XOR based)	Terminator (CR)

Double-click the blocks and verify the parameter values specified in the Block Parameters dialog box.





For other blocks, the parameters are:

Block	Parameter Name	Value
Constant	Interpret vector parameters as 1-D	selected
	Sample time	inf
SCI Transmit	SCI module	B
SCI Receive	SCI module	B
	Data type	uint8
	Data length	10
	Sample time	0.1
Display Status	Format	short

The value for **Data length** parameter of SCI Receive block is set to *10*, which is the sum of these values:

```
Header size (1 byte) + Data1 (1 byte) + Data2 (2 bytes) + Data3 (4 bytes) + Checksum (1 byte) + T
```

Step 4: Run the Model in XCP-based External Mode

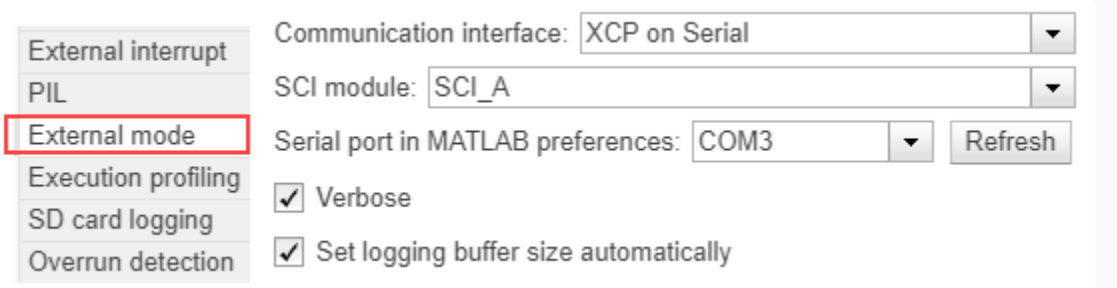
You can simulate the model in XCP-based External mode, which deploys the model as a C code on the hardware. The code obtains real-time data from the hardware.

Set Up the Model for XCP-based External Mode

1. To configure the model, click **Hardware Settings** in the **Hardware** tab of the Simulink toolbar.

2. In the Configurations Parameters dialog box, select **Hardware Implementation**.

3. Under **Target Hardware Resources**, click **External mode** tab and select **XCP on Serial** and **SCI_A** for *Communication interface* and *SCI module* parameters respectively. Select the corresponding value for the COM port of the host computer to which the C2000-based processor is connected.



4. Click **Apply**. Click **OK** to close the dialog box.

The Stop Time (under **Simulation** tab) is already set to `inf`.

Run the Model on XCP-based External Mode

1. To tune parameters and monitor signals in this model while the application runs on the target hardware, on the **Hardware** tab, click **Monitor & Tune**.

The lower left corner of the model window displays status while Simulink prepares, downloads, and runs the model on the hardware.

At each time step, data specified in the `Constant` blocks are encoded into uint8 byte stream (packet) and transmitted by the TX1 pin to the RX1 pin of your C2000-based processor. You can see the uint8 byte stream generated as per the packet structure in the `Display encoded data` block.

The RX1 pin receives the uint8 byte stream which is decoded using `Protocol Decoder` block and displays it on the `Display Decoded data` blocks. Observe the output in the `Display Decoded data` blocks which will be same as the value given in the `Constant` blocks at the transmission end.

2. Try changing the values in the `Constant` blocks connected as input to the Protocol Encoder block, and observe if the same values are getting decoded by the Protocol Decoder block.

3. To stop running the model, click **Stop**.

Other Things to Try

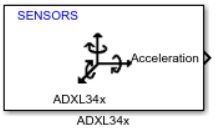
- Try specifying a different packet structure using the block properties and observe the encoded data and decoded data in the `Display` blocks.

More About

- Protocol Encoder
- Protocol Decoder

ADXL34x Accelerometer

Measure linear acceleration along axes of ADXL34x family of accelerometers



Libraries:
C2000 Microcontroller Blockset / Sensors

Description

The ADXL34x Accelerometer block reads data from the ADXL34x family of accelerometers (ADXL343, ADXL344, ADXL345, and ADXL346) with C2000 board.

You can use this block to measure linear acceleration along the *X*, *Y* and *Z*-axes. The block also provides the option to enable the data ready interrupt.

The block outputs acceleration as a 1-by-3 array.

Ports

Output

Acceleration — Linear acceleration measured by ADXL34x sensor
vector

This port outputs the linear acceleration (in m/s^2) along the *x*-, *y*- and *z*-axes as a 1-by-3 array.

Data Types: `double`

Note It is observed that there are many faulty cloned ADXL345 sensors available in the market, with Z axis raw output becoming unresponsive. Buy the sensor from a genuine distributor to avoid this issue.

Parameters

I2C module — Module for communication
`I2C_A` (default)

The I2C module to be used for communication to the ADXL34x sensor. The number of I2C modules supported varies across different C2000 processors. You can find the supported I2C modules corresponding to the processor (which you selected for the Hardware Board parameter in the Simulink model) by opening the Configuration Parameters dialog box and checking the I2C specific tabs under **Target hardware resources**.

I2C address — I2C address of ADXL34x sensor
`0x53` (default) | `0x1D`

The I2C address of the ADXL34x sensor from which the block reads the values.

The ADXL34x sensor can have two I2C addresses depending on the logic level on pin ALT of the sensor.

Pin Name	Pin State	I2C Address
ALT	Low	0x53
	High	0x1D

Data type — Output data type for values from ADXL34x sensor
`single` (default) | `double`

Select the data type of the sensor from which the block reads the values.

Specify the output data type for the values read from ADXL34x sensor. The default data type for C2000 board is `single`. Use this parameter to change the values to `double`, if required.

Sample time — Time interval to read data
`-1` (default) | positive integer

Specify how often this block reads the data from the ADXL34x sensor. When you set this parameter to `-1`, Simulink determines the best sample time for the block based on the block context within the model.

Advanced settings

Accelerometer range — Full scale for measuring linear acceleration
`±4g` (default) | `±2g` | `±8g` | `±16g`

Select the range of acceleration that the accelerometer can measure.

Accelerometer output data rate — Rate at which accelerometer data is sampled
`12.5 Hz` (default) | `0.1 Hz` | `0.2 Hz` | `0.39 Hz` | `0.78 Hz` | `1.56 Hz` | `3.13 Hz` | `6.25 Hz` | `25 Hz` | `50 Hz` | `100 Hz` | `200 Hz` | `400 Hz` | `800 Hz` | `1600 Hz`

Select the output data rate at which accelerometer data is sampled.

Enable data ready interrupt — Enable interrupt when data is ready
`off` (default) | `on`

If this option is selected, an interrupt is generated on pin INT1 or INT2 of the sensor when data is ready, allowing you to trigger other subsystems to perform any action.

Interrupt generate pin — Pin to generate data ready interrupt
`INT1` (default) | `INT2`

Select the interrupt generate pin to read accelerometer data.

Dependencies

To enable this parameter, select the **Enable data ready interrupt** parameter.

Version History

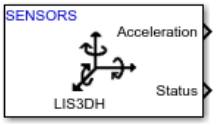
Introduced in R2022a

See Also

Topics
“Workaround to reset interrupt pin of sensors supporting latched interrupts”

LIS3DH Accelerometer Sensor

Measure linear acceleration, voltage, and temperature from LIS3DH sensor



Libraries:
C2000 Microcontroller Blockset / Sensors

Description

The LIS3DH Accelerometer Sensor block measures linear acceleration, voltages on external input pins (ADC1, ADC2 and ADC3) of the sensor, and temperature using the LIS3DH Accelerometer sensor interfaced with the C2000 board.

The block also provides the option to enable the high pass filter, FIFO and FIFO interrupt. An interrupt is generated if **Generate data ready interrupt** is selected. After selecting **Generate data ready interrupt**, if FIFO is disabled data ready interrupt is generated and if FIFO is enabled FIFO interrupt is generated.

The block outputs status of acceleration measurement, specified as a value `0`, `1`, or `2`. The value `0` indicates that the data read is new, `1` indicates that the data read is not new, and `2` indicates that the data is overwritten.

The block outputs single / double click axis as a `1`-by-`3` vector. The click axis provides information of the axis, on which click is detected. The axis value displays `1` when a click is detected otherwise it displays `0`.

The block supports inertial wake-up, free-fall, `6D` position, `6D` movement, `4D` position and `4D` movement recognition. The block outputs inertial wake-up axis as a `1`-by-`3` vector. This provides information of the axis, on which inertial wake-up is detected. The `6D` position axis and `6D` movement axis are of size `1`-by-`6`. The `4D` position and `4D` movement axis are of size `1`-by-`4`. The status contains the information about the axis, on which the detection is recognized. The status value displays `1` when the configured detection is recognized on the selected axis, otherwise it displays `0`.

Ports

Output

Acceleration — Acceleration along each axis as measured by accelerometer
vector

The block outputs acceleration as a `1`-by-`3` vector when FIFO is disabled. When FIFO is enabled, the block outputs `n`-by-`3` vector, where `n` is the number of samples on page 2-0 . Each value represents the measurement of acceleration in `m/s^2` along the X, Y, and Z axes.

Dependencies

This output port appears only if you select the **Acceleration (m/s^2)** parameter.

Data Types: `single` | `double` | `int16`

ADC1 Voltage — Voltage on external pin ADC1
vector

The block outputs voltage applied on external pin ADC1 of the LIS3DH sensor.

Dependencies

This output port appears only if you select the **Voltage (external input at ADC1)** parameter.

Data Types: `single` | `double` | `int16`

ADC2 Voltage — Voltage on external pin ADC2
vector

The block outputs voltage applied on external pin ADC2 of the LIS3DH sensor.

Dependencies

This output port appears only if you select the **Voltage (external input at ADC2)** parameter.

Data Types: `single` | `double` | `int16`

ADC3 Voltage — Voltage on external pin ADC3
vector

The block outputs voltage applied on external pin ADC3 of the LIS3DH sensor.

Dependencies

This output port appears only if you select `Voltage (external input)` for **ADC3 input** parameter.

Data Types: `single` | `double` | `int16`

Temperature — Temperature measured by LIS3DH sensor
scalar

Temperature (in ℃) measured by LIS3DH sensor connected to C2000 board.

Dependencies

This output port appears only if you select `Temperature` for **ADC3 input** parameter.

Data Types: `single` | `double` | `int16`

Accelration Status — Status of acceleration measurement
`2` | `0` | `1`

Status of acceleration measurement, specified as a value `0`, `1`, or `2`. The value `0` indicates that the data read is new, `1` indicates that the data read is not new, and `2` indicates that the data is overwritten.

If **Enable FIFO** is selected, then `0` indicates that the specified samples are collected, `1` indicates that specified number of samples are not collected, and `2` indicates that the data is overwritten.

Dependencies

This output port appears only if you select the **Acceleration data ready Status** parameter.

Data Types: `uint8`

Samples pending — Number of pending samples
scalar

The block outputs the number of pending samples to be read as a value ranging from `0-32`.

Dependencies

This output port appears only if you select the **Enable FIFO** parameter.

Data Types: `uint8`

Click status [X,Y,Z] — Click detection status
`0` | `1`

The block outputs the Click status of axes as a 1-by-3 vector (x,y,z) indicating on which axis the click has been detected. The axis value `1` indicates that the click is detected on that axis and `0` indicates that the click is not detected.

Dependencies

This output port appears only if you select the **click status** parameter.

Data Types: `uint8`

Inertial wake-up status [X,Y,Z] — Status of inertial wake-up interrupt
`0` | `1`

The block outputs inertial wake-up status as a 1-by-3 vector. This provides information of the axis, on which inertial wake-up is detected. The value `1` indicates that the inertial wake-up event is detected on the selected axis and value `0` indicates the event is not detected.

Dependencies

This parameter appears only if you select `Inertial wake-up` from **Detect** parameter and select **Detection status** parameter.

Data Types: `uint8`

Free-fall status — Status of free-fall
`0` | `1`

The block outputs free-fall status. The value `1` indicates that the free-fall is detected on the selected axis and `0` indicates the interrupt is not detected.

Dependencies

This parameter appears only if you select `Free-fall` from **Detect** parameter and select **Detection status** parameter.

Data Types: `uint8`

6D position status [X, -X, Y, -Y, Z, -Z] — Status of 6D position
`0` | `1`

The block outputs 6D position status as a 1-by-6 vector. The value `1` indicates that the 6D position interrupt is detected on the selected axis and `0` indicates the interrupt is not detected.

Dependencies

This parameter appears only if you select `6D position` from **Detect** parameter and select **Detection status** parameter.

Data Types: `uint8`

6D movement status [X, -X, Y, -Y, Z, -Z] — Status of 6D movement
`0` | `1`

The block outputs 6D movement status as a 1-by-6 vector. The value `1` indicates that the 6D movement interrupt is detected on the selected axis and `0` indicates the interrupt is not detected.

Dependencies

This parameter appears only if you select `6D movement` from **Detect** parameter and select **Detection status** parameter.

Data Types: `uint8`

4D position status [X, -X, Y, -Y] — Status of 4D position
`0` | `1`

The block outputs 4D position status as a 1-by-4 vector. The value `1` indicates that the 4D position interrupt is detected on the selected axis and `0` indicates the interrupt is not detected.

Dependencies

This parameter appears only if you select `4D position` from **Detect** parameter and select **Detection status** parameter.

Data Types: `uint8`

4D movement status [X, -X, Y, -Y] — Status of 4D movement
`0` | `1`

The block outputs 4D movement status as a 1-by-4 vector. The value `1` indicates that the 4D movement interrupt is detected on the selected axis and `0` indicates the interrupt is not detected.

Dependencies

This parameter appears only if you select `4D movement` from **Detect** parameter and select **Detection status** parameter.

Data Types: `uint8`

Parameters

Main

I2C Module — Specific module used for I2C communication
`I2C_A` (default)

Specify the module on the board that you are using for I2C communication.

I2C address — I2C addresses to communicate with sensor peripherals
`0x18` (default) | `0x19`

The I2C addresses to communicate with the accelerometer peripheral on the LIS3DH sensor are decided by the state of the SA0 pin on the hardware board. This table provides the I2C addresses corresponding to the pin and their state.

Pin Name	Pin State	I2C Address
SA0	0	0x18
	1	0x19

Acceleration (m/s^2) — Set output port for reading acceleration
`on` (default) | `off`

Select this parameter to set **Acceleration** as one of the output ports.

ADC3 input — Select output port for reading temperature or voltage
`None` (default) | `Temperature` | `Voltage (external input)`

Select `Temperature` to set **Temperature** as one of the output ports or select `Voltage (external input)` to set **ADC3 Voltage** as one of the output port.

Voltage (external input at ADC1) — Select output port for reading voltage
`off` (default) | `on`

Select this parameter to set **ADC1 Voltage** as one of the output ports.

Voltage (external input at ADC2) — Select output port for reading voltage
`off` (default) | `on`

Select this parameter to set **ADC2 Voltage** as one of the output ports.

Acceleration data ready status — Set output port for obtaining acceleration status
`on` (default) | `off`

Select this parameter to set **Acceleration status** as one of the output ports.

Accelerometer resolution — Data register bit resolution
`12bit` (default) | `10bit` | `8bit`

Select the data register bit resolution for the sensor.

Accelerometer output data rate (Hz) — Rate at which accelerometer data is sampled
`400 Hz` (default) | `1 Hz` | `10 Hz` | `25 Hz` | `50 Hz` | `100 Hz` | `200 Hz` | `1344 Hz`

Select the output data rate at which accelerometer data is sampled.

Accelerometer range — Full scale for measuring linear acceleration
`±2g` (default) | `±4g` | `±8g` | `±16g`

Select the full scale for measuring linear acceleration (the range of acceleration that the sensor needs to measure).

Enable FIFO — Enable number of samples to read
`off` (default) | `on`

Select this option to enable the option **Number of samples to read from FIFO (1-32)**, which allows you to set the number of samples to be read.

Number of samples to read from FIFO (1-32) — Number of samples to read
2 (default)

Enter a value for number of samples (1-32) to be read from FIFO.

Dependencies

This parameter appears only if you select the **Enable FIFO** parameter.

Generate FIFO overrun interrupt on INT1 pin — Generate interrupt when FIFO buffer is full
`off` (default) | `on`

Select this option to receive an interrupt if the FIFO buffer gets filled and the first sample is overwritten.

Dependencies

This parameter appears only if you select the **Enable FIFO** parameter.

Enable high pass filter — Enable high pass filter for accelerometer data
`off` (default) | `on`

Enable the high pass filter for reading accelerometer values.

HPF cutoff frequency (Hz) — Bandwidth of high pass filter
`100 Hz` (default) | `12 Hz` | `25 Hz` | `50 Hz`

Select the required bandwidth of the high pass filter for reading accelerometer values. The HPF cutoff frequency depends on the Accelerometer output data rate (Hz). The HPF cutoff frequencies for the selected Accelerometer output data rate (Hz) are shown in this table. The values `00`, `01`, `10`, and `11` are the HPF mode configurations.

HPF cutoff frequency

HPF cutoff frequency[2:1]	Acceleration output data rate - 1 Hz	Acceleration output data rate - 10 Hz	Acceleration output data rate - 25 Hz	Acceleration output data rate - 50 Hz	Acceleration output data rate - 100 Hz	Acceleration output data rate - 200 Hz	Acceleration output data rate - 400 Hz	Acceleration output data rate - 1600 Hz	Acceleration output data rate - 5376 Hz
00	0.02	0.2	0.5	1	2	4	8	32	100
01	0.008	0.08	0.2	0.5	1	2	4	16	50
10	0.004	0.04	0.1	0.2	0.5	1	2	8	25
11	0.002	0.02	0.05	0.1	0.2	0.5	1	4	12

Dependencies

This parameter appears only if you select the **Enable high pass filter** parameter.

Generate data ready interrupt on INT1 pin — Generate interrupt when data is ready
`off` (default) | `on`

If this option is selected, an interrupt is generated on pin INT1 of the sensor when data is ready, allowing you to trigger other subsystems to perform any action.

Data type — Output data type for values from lis3dh sensor
`single` (default) | `double` | `int16`

Specify the output data type for the values read from lis3dh sensor. The default data type for C2000 board is `single`. Use this parameter to change the values to `double` or `int16`, if required.

Sample time — Time interval to read data
`-1` (default) | positive integer

Specify how often this block reads the data from the LIS3DH sensor. When you set this parameter to -1, Simulink determines the best sample time for the block based on the block context within the model.

Click detection

Enable click detection — Option for enabling click detection
`off` (default) | `on`

Select this parameter to enable click detection.

Type — Select type of click to be detected
`Single click` (default) | `Double click`

Select the type of click to be detected. The available options are `Single click` and `Double click`.

Dependencies

This parameter appears only if you select **Enable click detection** parameter.

Threshold (m/s^2) — Threshold for detecting single or double click
`1.5` (default)

Specify the threshold for detecting single click or double click. When the value crosses the specified threshold value, click is detected.

Dependencies

This parameter appears only if you select **Enable click detection** parameter.

Time limit (s) — Time limit duration
`0.3` (default) | `positive integer`

Specify the time limit value in seconds for detecting single click or double click. The click must occur during the specified time limit for detecting single click or double click.

Dependencies

This parameter appears only if you select **Enable click detection** parameter.

Time latency (s) — Time latency duration
`0.3` (default) | `positive integer`

Specify the time latency value in seconds for detecting double click. The second click must occur after the specified time latency period for detecting a double click.

Dependencies

This parameter appears only if you select **Enable click detection** parameter and select `Double click` option for **Type** parameter.

Time window (s) — Time window duration
`0.3` (default) | `positive integer`

Specify the time window value in seconds for detecting double click. The second click must occur before the time window period expires for detecting double click.

Dependencies

This parameter appears only if you select **Enable click detection** parameter and select `Double click` option for **Type** parameter.

X axis — Click detection in X axis
`on` (default) | `off`

Select this option to enable click detection along X axis. The value `1` indicates that the click is detected and `0` indicates that the click is not detected

Dependencies

This parameter appears only if you select **Enable click detection** parameter.

Y axis — Click detection in Y axis
`on` (default) | `off`

Select this option to enable click detection along Y axis. The value `1` indicates that the click is detected and `0` indicates that the click is not detected

Dependencies

This parameter appears only if you select **Enable click detection** parameter.

Z axis — Click detection in Z axis
`off` (default) | `on`

Select this option to enable click detection along Z axis. The value `1` indicates that the click is detected and `0` indicates that the click is not detected

Dependencies

This parameter appears only if you select **Enable click detection** parameter.

Generate click interrupt — Generate interrupt when click is detected
`off` (default) | `on`

Select the option to generate an interrupt if a click is detected.

Dependencies

This parameter appears only if you select **Enable click detection** parameter.

Interrupt pin — Pin to generate click interrupt
`INT1` (default) | `INT2`

If this option is selected, an interrupt is generated on pin INT1 or IN2 of the sensor when data is ready, allowing you to trigger other subsystems to perform any action.

Dependencies

This parameter appears only if you select **Generate click interrupt** parameter.

Click status — Set output port for obtaining click detection status
`off` (default) | `on`

Select this parameter to set **Click status [X,Y,Z]** as one of the output ports.

Dependencies

This parameter appears only if you select **Enable click detection** parameter.

Configurable detections

Configurable detections — Set number of configurable detections
`0` (default) | `1` | `2`

Select the required option for the number of configurable detections.

Configurable detections 1

Detect — Select interrupt
`Inertial wake-up` (default) | `Free-fall` | `6D position` | `6D movement` | `4D position` | `4D movement`

Select the required feature of the sensor. (Inertial wake-up, Free-fall, 6D position, 6D movement, 4D position, 4D movement).

Dependencies

This parameter appears only if you select 1 or 2 from **Configurable detections** parameter.

X axis — Interrupt detection in X axis
`on` (default) | `off`

Select this option to enable the configurable event detection status on X axis. The value `1` indicates that the event is detected and `0` indicates that the event is not detected.

Dependencies

This parameter appears only if you select `Inertial wake-up` or `6D position` or `6D movement` or `4D position` or `4D movement` from **Detect** parameter.

Y axis — Interrupt detection in Y axis
`on` (default) | `off`

Select this option to enable the configurable event detection status on Y axis. The value `1` indicates that the event is detected and `0` indicates that the event is not detected.

Dependencies

This parameter appears only if you select `Inertial wake-up` or `6D position` or `6D movement` or `4D position` or `4D movement` from **Detect** parameter.

Z axis — Interrupt detection in Z axis
`on` (default) | `off`

Select this option to enable the configurable event detection status on Z axis. The value `1` indicates that the event is detected and `0` indicates that the event is not detected.

Dependencies

This parameter appears only if you select `Inertial wake-up` or `6D position` or `6D movement` from **Detect** parameter.

-X axis — Interrupt detection in -X axis
`off` (default) | `on`

Select this option to enable the configurable event detection status on -X axis. The value `1` indicates that the event is detected and `0` indicates that the event is not detected.

Dependencies

This parameter appears only if you select `6D position` or `6D movement` or `4D position` or `4D movement` from **Detect** parameter.

-Y axis — Interrupt detection in -Y axis
`off` (default) | `on`

Select this option to enable the configurable event detection status on -Y axis. The value `1` indicates that the event is detected and `0` indicates that the event is not detected.

Dependencies

This parameter appears only if you select `6D position` or `6D movement` or `4D position` or `4D movement` from **Detect** parameter.

-Z axis — Interrupt detection in -Z axis
`off` (default) | `on`

Select this option to enable the configurable event detection status on -Z axis. The value `1` indicates that the event is detected and `0` indicates that the event is not detected.

Dependencies

This parameter appears only if you select `6D position` or `6D movement` from **Detect** parameter.

Threshold (m/s^2 — Acceleration threshold for detecting the selected configurable detection
`1.5` (default)

Specify the threshold for detecting the selected configurable detection. When the acceleration value crosses the specified threshold value, the event is detected.

Dependencies

This parameter appears only if you select `1` or `2` from **Configurable detections** parameter.

Duration (s) — Time limit duration
`0.3` (default) | `positive integer`

Specify the time limit for detecting the configurable event. To detect an event, the Threshold criteria must be satisfied for a minimum time equal to Duration.

Dependencies

This parameter appears only if you select 1 or 2 from **Configurable detections** parameter.

Generate interrupt — Generate interrupt
`off` (default) | `on`

If this option is selected, an interrupt is generated for the selected configurable detection.

Dependencies

This parameter appears only if you select 1 or 2 from **Configurable detections** parameter.

Interrupt pin — Pin to generate interrupt
`INT1` (default) | `INT2`

If this option is selected, an interrupt is generated on pin INT1 or IN2 of the sensor when the configurable event is detected, allowing you to trigger other subsystems to perform any action.

Dependencies

This parameter appears only if you select the **Generate interrupt** parameter.

Note Parameters for **Configuration detections 2** are similar to **Configuration detections 1**. For information to configure these parameters, see “Configurable detections 1” on page 2-0 .

Version History

Introduced in R2022b

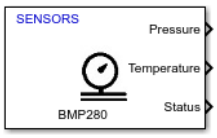
See Also

Topics

“Workaround to reset interrupt pin of sensors supporting latched interrupts”

BMP280 Pressure Sensor

Measure barometric air pressure and ambient temperature from BMP280 sensor



Libraries:
C2000 Microcontroller Blockset / Sensors

Description

The BMP280 Pressure Sensor block measures barometric air pressure and ambient temperature using the BMP280 Pressure sensor interfaced with the C2000 board.

Ports

Output

Pressure — Barometric air pressure
scalar

The block outputs barometric air pressure in Pascal (Pa).

Dependencies

This output port appears only if you select the **Pressure (Pa)** parameter.

Data Types: `double`

Temperature — Ambient temperature
scalar

The block outputs the ambient temperature in ℃.

Dependencies

This output port appears only if you select the **Temperature (°C)** parameter.

Data Types: `double`

Status — Status of values read for pressure and temperature
`0` | `1`

The block outputs the status of barometric air pressure and ambient temperature measurements. `0` indicates that the data read is new, `1` indicates that the data read is not new.

Dependencies

This output port appears only if you select the **Status** parameter.

Data Types: `uint8`

Parameters

I2C Module — Specific module used for I2C communication
`I2C_A` (default)

Specify the module on the board that you are using for I2C communication.

I2C address — I2C address to communicate with the BMP280 sensor
`0x76` (default) | `0x77`

The I2C addresses to communicate with the peripheral on the BMP280 sensor are decided by the state of the SDO pin on the hardware board. This table provides the I2C addresses corresponding to the pin and their state.

Pin Name	Pin State	I2C address
SDO	0	0x76
	1	0x77

Pressure (Pa) — Set output port for reading pressure
`on` (default) | `off`

Select this parameter to set **Pressure** as one of the output ports.

Temperature (°C) — Set output port for reading temperature
`on` (default) | `off`

Select this parameter to set **Temperature** as one of the output ports.

Status — Set output port for obtaining status of selected outputs
`on` (default) | `off`

Select this parameter to set **Status** as one of the output ports.

Data type — Output data type for values from BMP280 sensor
`single` (default) | `double` | `uint32`

Select the data type of the sensor from which the block reads the values.

Specify the output data type for the values read from BMP280 sensor. The default data type for C2000 hardware is `single`. Use this parameter to change the values to `double`, or `uint32` if required.

Sample time — Time interval to read data
`-1` (default) | positive integer

Specify how often this block reads the data from the BMP280 sensor. When you set this parameter to -1, Simulink determines the best sample time for the block based on the block context within the model.

IIR filter settings

Filter coeffiecient — Filter coefficient value
`0` (default) | `2` | `4` | `8` | `16`

Select the filter coefficient value of the sensor, which helps in reducing the bandwidth of output signals.

The default filter coefficient value for C2000 hardware is `0`. Use this parameter to change the values, if required.

Sensitivity

Pressure sensitivity factor (Pa) — Pressure sensitivity factor
`2.62` (default) | `1.31` | `0.66` | `0.33` | `0.16`

Select the pressure sensitivity factor, which the sensor uses to read values during changing air pressure.

The default data type for C2000 hardware is `2.62`. Use this parameter to change the values, if required.

When filter is enabled, the sensor uses this formula to get the latest measurement calculation.

d_t = (d_{adc} / f) + d_{t-1} (1 - 1 / f), where d_t is the current data, d_{t-1} is the previous data, d_{adc} is the prefiltered data, and `f` is the filter coefficient.

Temperature sensitivity factor (°C) — Temperature sensitivity factor
`0.005` (default) | `0.0025`

Select the temperature sensitivity factor, which the sensor uses to read values during changing temperature conditions.

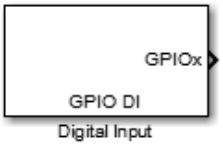
The default data type for C2000 hardware is `0.005`. Use this parameter to change the values, if required.

Version History

Introduced in R2022b

C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F2838x-M4/F28004x/F28002x/F28003x GPIO Digital Input

Configure general-purpose input/output pins as digital input



Libraries:
C2000 Microcontroller Blockset / C2802x
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C281x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / C2834x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x
C2000 Microcontroller Blockset / F2838x / M4
C2000 Microcontroller Blockset / F28M35x / C28x
C2000 Microcontroller Blockset / F28M35x / M3
C2000 Microcontroller Blockset / F28M36x / C28x
C2000 Microcontroller Blockset / F28M36x / M3

Description

This block configures the general-purpose I/O (GPIO) MUX registers that control the operation of GPIO shared pins for digital input. Each I/O port has one MUX register that selects peripheral operation or digital I/O operation (the default). When a pin is configured for digital input, it becomes unavailable for digital output or peripheral operation. You can configure the **Input qualification type** for individual digital input pins. To configure, go to **Configuration Parameters > Hardware Implementation > Target Hardware Resources** and select the appropriate GPIO group.

Each processor has a different number of available GPIO pins.

Note To avoid losing new settings, click **Apply** before changing the **GPIO Group** parameter.

Ports

Output

Pin # — GPIO pin status
scalar | vector

The port outputs the status of the digital pin you select in the **GPIO Group** parameter.

Parameters

GPIO Group — GPIO pins you want to configure
`GPIO0-GPIO7` (default) | `GPIO8-GPIO15` | `GPIO16-GPIO23` | …

Select the group of GPIO pins you want to view or configure. For a table of GPIO pins and peripherals, refer to the Texas Instruments documentation for your specific target.

Sample time — Frequency at which block reads input pin values
`-1` (default) | `0.1`

Specify how often the block receives message, in seconds. When you specify this parameter as -1, Simulink determines the best sample time for the block based on the block context within the model.

For more information, refer to the section on “Specify Sample Time”.

Data type — Data type of GPIO input
`auto` (default) | `unit8` | `double` | `single` | `int8` | `int16` | `uint16` | `int32` | `uint32` | `boolean`

Specify the data type of the input. The input is read as 16-bit integer, and then cast to the selected data type.

Version History

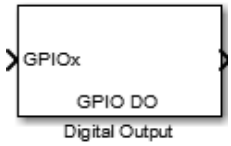
Introduced in R2016a

See Also

C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F2838x-M4/F28004x/F28002x GPIO Digital Output

C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F2838x-M4/F28004x/F28002x/F28003x GPIO Digital Output

Configure general-purpose input/output pins as digital input



Libraries:
C2000 Microcontroller Blockset / C2802x
C2000 Microcontroller Blockset / C2803x
C2000 Microcontroller Blockset / C2805x
C2000 Microcontroller Blockset / C2806x
C2000 Microcontroller Blockset / C280x
C2000 Microcontroller Blockset / C281x
C2000 Microcontroller Blockset / C2833x
C2000 Microcontroller Blockset / C2834x
C2000 Microcontroller Blockset / F28002x
C2000 Microcontroller Blockset / F28003x
C2000 Microcontroller Blockset / F28004x
C2000 Microcontroller Blockset / F2807x
C2000 Microcontroller Blockset / F2837xD
C2000 Microcontroller Blockset / F2837xS
C2000 Microcontroller Blockset / F2838x / C28x
C2000 Microcontroller Blockset / F2838x / M4
C2000 Microcontroller Blockset / F28M35x / C28x
C2000 Microcontroller Blockset / F28M35x / M3
C2000 Microcontroller Blockset / F28M36x / C28x
C2000 Microcontroller Blockset / F28M36x / M3

Description

This block configures the general-purpose I/O (GPIO) MUX registers that control the operation of GPIO shared pins for digital input. Each I/O port has one MUX register that selects peripheral operation or digital I/O operation (the default). When a pin is configured for digital input, it becomes unavailable for digital output or peripheral operation. You can configure the **Input qualification type** for individual digital input pins. To configure, go to **Configuration Parameters > Hardware Implementation > Target Hardware Resources** and select the appropriate GPIO group.

Each processor has a different number of available GPIO pins.

Note To avoid losing new settings, click **Apply** before changing the **GPIO Group** parameter.

Ports

Input

Input — Set the GPIO pin status
scalar | vector

The input port to set the GPIO pin status.

Parameters

GPIO Group — GPIO pins you want to configure
`GPIO0-GPIO7` (default) | `GPIO8-GPIO15` | `GPIO16-GPIO23` | …

Select the group of GPIO pins you want to view or configure. For a table of GPIO pins and peripherals, refer to the Texas Instruments documentation for your specific target.

GPIO pins for output — GPIO pins you want to configure
`GPIO0` (default) | `GPIO8` | `GPIO16` | …

To configure a GPIO pin for digital output, select the check box next to it. Refer to the block for a table of all available peripherals for each pin.

A value of True at the input of the block drives the selected GPIO pin high. A value of False at the input of the block grounds the selected GPIO pin.

Toggle GPIO# — GPIO pins you want to configure
`Toggle GPIO0` (default) | `Toggle GPIO1` | …

For each pin selected for output, you can elect to toggle the signal of that pin. In **Toggle** mode, a value of `True` at the input of the block switches the GPIO pin output level. Thus, if the GPIO pin was driven high, in **Toggle** mode, with the value of `True` at the input, the pin output level is driven low. If the GPIO pin was driven low, in **Toggle** mode, with the value of `True` at the input of the block, the same pin output level is driven high. If the input of the block is `False`, the GPIO pin output level is unaffected.

Note The outputs of this block can be vectorized.

Version History

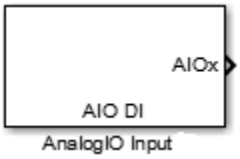
Introduced in R2016a

See Also

C280x/C2802x/C2803x/C2805x/C2806x/C2833x/C2834x/F28M3x/F2807x/F2837xD/F2837xS/F2838x/F2838x-M4/F28004x/F28002x GPIO Digital Input

C2802x/C2803x/C2806x/F28M3x AnalogIO Input

Configure pin, sample time, and data type for analog input



Description

Use this block to sample the voltage on Analog IO pins and output the results.

Parameters

Parameters (Input pins)

Select the input pins to sample.

Sample time

Specify the time interval between samples. To inherit sample time from the upstream block, set this parameter to -1.

Data type

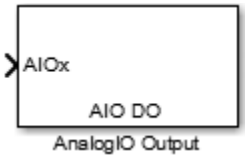
Select the data type of the digital output data. If you select `auto`, the block automatically selects the data type for your model. You can also manually select a data type. You can choose from the options `double`, `single`, `int8`, `uint8`, `int16`, `uint16`, `int32`, and `uint32`.

See Also

C2802x/C2803x/C2806x/F28M3x AnalogIO Output

C2802x/C2803x/C2806x/F28M3x AnalogIO Output

Configure Analog IO to output analog signals on specific pins



Description

Use this block to drive the output voltage of Analog IO pins.

In regular mode, a value of True at the input of the block pulls the Analog IO pin high while a value of False grounds the pin.

In toggle mode, a value of True at the input of the block switches the actual output level of the Analog IO pin while a value of False does not alter the output level of the Analog IO pin.

Parameters

Parameters (Output Pins)

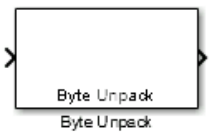
Select the analog output pins. Selecting **Toggle** inverts the output voltage levels of the pins if the input of the block is True.

See Also

C2802x/C2803x/C2806x/F28M3x AnalogIO Input

Byte Unpack

Unpack 8-, 16-, or 32-bit input vector to multiple output vectors



Libraries:
C2000 Microcontroller Blockset / Target Communication

Description

The Byte Unpack block converts a vector of `uint8`, `uint16`, or `uint32` data type to one or more signals of user-selectable data types. This block is the inverse of the Byte Pack block. The input of this block connects to an output port of a receive block, such as SPI Receive, SCI Receive, or UDP Receive. The Receive block then transmits signals across various communication networks, such as SPI, SCI, UDP, or I2C.

Input/Output Ports

Input

Port_1 — Packed data
scalar | vector | matrix

Receives a vector of packed data.

Data Types: `uint8` | `uint16` | `uint32`

Output

Port_1 — First of *N* output ports
scalar | vector | matrix

The block can have from 1 to *N* output ports, as specified by elements of the cell array in the parameter **Output port data types (cell array)**.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Parameters

Output port dimensions (cell array) — Dimensions of each output port (unpacked)
`{[1]}` (default) | `{[N], [M], ...}`

Output port dimensions specified as a cell array of vectors.

Specify the same dimensions that you set for the corresponding Byte Pack block in the model.

Output port data types (cell array) — Data types for unpacked output signals
`double` (default) | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `boolean`

Data types of the output ports (unpacked) specified for different output signals as a cell array. The number of elements in the cell array determines the number of output ports shown by this block instance.

Specify the same data types that you set in the **Input port data types (cell array)** parameter for the corresponding Byte Pack block in the model.

Byte alignment — Alignment of output signal data types before unpacking
1 (default) | 2 | 4 | 8

Each element in the input signals list starts at a multiple of the byte alignment value, specified from the start of the vector. If the byte alignment value is larger than the size of the data type in bytes, the output values are padded with zeros to fill the space allotted.

For example, if the byte alignment value is 4, a `uint32` receives no padding, a `uint16` receives 2 bytes of padding, and a `uint8` receives 3 bytes of padding.

Choose the same byte alignment value that you set in the **Byte alignment** parameter for the corresponding Byte Pack block in the model.

Example

Suppose that you are unpacking a vector of data type `uint8` or `uint16`, and the unpacked signals have these attributes.

Dimension	Size	Type
Vector	3	`int8`
Vector	2	`int16`
Scalar	1	`uint8`
Scalar	1	`uint32`

To unpack the signals:

1. Set **Output port dimensions (cell array)** to:

   ```
   {'3', '2', '1', '1'}
   ```

2. Set **Output port data types (cell array)** to:

   ```
   {'int8', 'int16', 'uint8', 'uint32'}
   ```

   The block creates four output ports that match the order of the signal data types specified in the cell array.

3. Set the required byte alignment value. The byte alignment value specifies the number of bytes after which a new byte starts from the previous boundary.

   The size of the output is based on the packed vector size, the byte alignment value, and the smallest memory cell size of the processor. Depending on the byte alignment value, output values padded with zeros are discarded before the next signal is unpacked. The smallest addressable memory cell indicates the number of bits occupied by `char` or `uint8` data type for a processor, and determines the structure of packets.

4. Connect incoming signals to the input port of the Byte Unpack block.

For processors with a smallest addressable memory cell of 8 bits per char, consider the packed input vector data type `uint8`.

Packed Vector Type - uint8

Packet	23	04	FD	DA	00	F4	FF	70	88	13	00	00
Alignment	1	1	1	1	1	1	1	1	1	1	1	1

Packet	23	04	FD	00	DA	00	F4	FF	70	00	88	13	00	00
Alignment	2		2		2		2		2		2		2	

Packet	23	04	FD	00	DA	00	F4	FF	70	00	00	00	88	13	00	00
Alignment	4				4				4				4			

Packet	23	04	FD	00	00	00	00	00	DA	00	F4	FF	00	00	00	00	70	00	00	00	00	00	00	00	88	13	00	00	00	00	00	00
Alignment	8								8								8								8							

Red zeros represent padded empty memory cells.

For a packed input vector of data type `uint8` and byte alignment value 2, the `int8` data value (23 04 FD) occupies three memory cells, with each cell occupying 8 bits. The next input signal of `int16` data value (00DA FFF4) occupies the next four cells (fifth through eighth), and the fourth cell is empty (padded). The Byte Unpack block considers the alignment and padding of cells while unpacking.

The packed input vector of data type `uint16` is:

Packed Vector Type - uint16

Packet	0423		DAFD		F400		70FF		1388		0000	
Alignment	1	1	1	1	1	1	1	1	1	1	1	1

Packet	0423	00FD	00DA	FFF4	0070	1388	0000
Alignment	2	2	2	2	2	2	2

Packet	0423	00FD	00DA	FFF4	0070	0000	1388	0000
Alignment	4		4		4		4	

Packet	0423	00FD	0000	0000	00DA	FFF4	0000	0000	0070	0000	0000	0000	1388	0000	0000	0000
Alignment	8				8				8				8			

The unpacked output signals are:

Unpacked Signals				
Dimension	**Size**	**Data Type**	**Dec Value**	**Hex Value**
Vector	3	`int8`	35	23
			4	04
			–3	FD
Vector	2	`int16`	218	00DA

Unpacked Signals				
Dimension	**Size**	**Data Type**	**Dec Value**	**Hex Value**
			–12	FFF4
Scalar	1	`uint8`	112	70
Scalar	1	`uint32`	5000	00001388

For processors such as Texas Instruments C2000, with a smallest addressable memory cell of 16 bits per char, consider a packed input vector data type `uint8`. The output packet occupies 16 bits although the data value that the packet represents is 8 bits. The byte alignment values are calculated with respect to the 16-bit addressable memory.

Packed Vector Type - uint8

Packet	0023	0004	00FD	00DA	0000	00F4	00FF	0070	0088	0013	0000	0000
Alignment	1	1	1	1	1	1	1	1	1	1	1	1

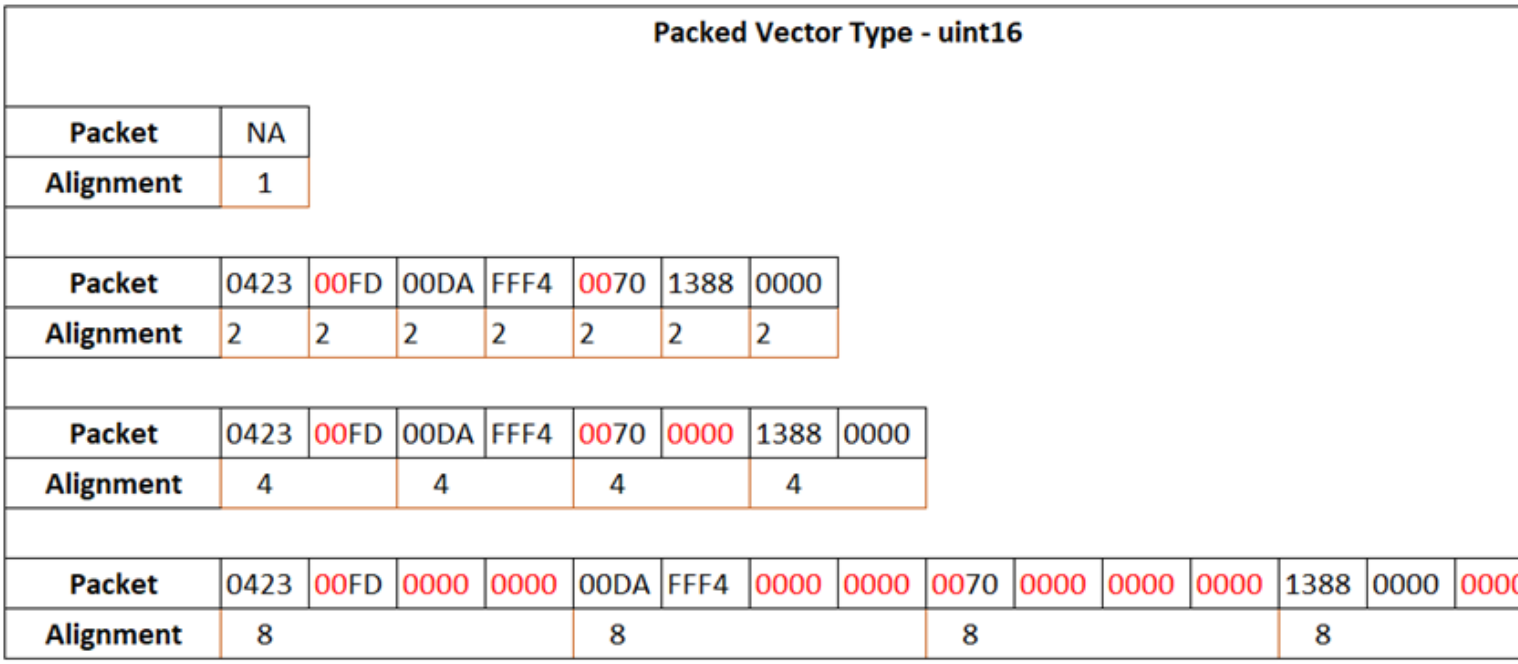
Packet	0023	0004	00FD	0000	00DA	0000	00F4	00FF	0070	0000	0088	0013	0000	0000
Alignment	2		2		2		2		2		2		2	

Packet	0023	0004	00FD	0000	00DA	0000	00F4	00FF	0070	0000	0000	0000	0088	0013	0000	0000
Alignment	4				4				4				4			

Packet	0023	0004	00FD	0000	0000	0000	0000	0000	00DA	0000	00F4	00FF	0000	0000	0000	0000	0070	0000	0000	0000	0000	0000	0000	0000	0088	0013	0000	0000	0000	0000	0000	0000
Alignment	8								8								8								8							

For a packed input vector of data type `uint8` and byte alignment value 2, the `int8` data value (0023 0004 00FD) occupies three memory cells, with each cell occupying 16 bits. The next signal of data type `int16` (00DA 0000 00F4 00FF) occupies the next four cells (fifth through eighth), and the fourth cell is empty (padded). The Byte Unpack block considers the alignment and padding of cells while unpacking.

For the packed input vector of data type `uint16`, the output packet occupies 16 bits, and the data value the packet represents is also 16 bits. For a packet size of 16 and larger, the byte alignment is calculated with respect to the number of bytes the data values have to be packed. Therefore, in this case, 1-byte alignment is not allowed.



For a packed input of data type `uint16` and byte alignment value 2, the three `int8` data values (0423 FD) occupy the first two memory cells. The fourth byte in the second memory cell is empty and padded with zero. The `int16` data value (00DA FFF4) occupies the next two memory cells (third and fourth). The Byte Unpack block considers the alignment and padding of cells while unpacking.

The table lists the unpacked output signals. The `int8` and `uint8` data values occupy 16 bits, as indicated by the hex value.

Unpacked Signals				
Dimension	**Size**	**Data Type**	**Dec Value**	**Hex Value**
Vector	3	`int8`	35	0023
			4	0004
			–3	FFFD
Vector	2	`int16`	218	00DA
			–12	FFF4
Scalar	1	`uint8`	112	0070
Scalar	1	`uint32`	5000	00001388

Version History

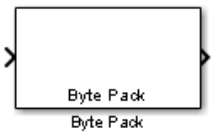
Introduced in R2016b

See Also

Byte Pack

Byte Pack

Convert input signals to 8-, 16-, or 32-bit vector



Libraries:
C2000 Microcontroller Blockset / Target Communication

Description

The Byte Pack block converts one or more signals of user-selectable data types to a single `uint8`, `uint16`, or `uint32` vector output. Using the parameters of this block, you specify the input data types and the alignment of the data in the output vector. The output of this block connects to an input port of a send block, such as SPI Transmit, SCI Transmit, or UDP Send. The send block then transmits signals across various communication networks, such as SPI, SCI, UDP, or I2C.

Note The Byte pack block requires the input signal dimensions to be calculated accurately so that the block can set the output port signal dimensions appropriately. In certain modeling scenarios, you may have to manually specify the dimensions if Simulink cannot calculate the dimensions accurately. For example,

- When you are providing an input port directly to the Byte Pack block, you need the dimensions accurately for the input port.
- When the input to the Byte Pack is coming through a feedback loop involving a delay block then the dimensions will not be automatically calculated by Simulink. In this scenario, you can use signal specification block with the correct dimension before the signal is passed as input to the Byte Pack.

Input/Output Ports

Input

Port_1 — First of *N* input ports
scalar | vector | matrix

The number of input ports and their types specified as a cell array in the **Input port data types (cell array)** parameter. The block can have from 1 to *N* input ports. *N* is the number of incoming data types specified in the cell array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Output

Port_1 — Vector containing packed data
vector

Transmits a vector of packed data.

Data Types: `uint8` | `uint16` | `uint32`

Parameters

Output port (packed) data type — Data type of packed output signal
`uint8` (default) | `uint16` | `uint32`

The data type of the packed output signal at the output port.

Input port data types (cell array) — Data types of unpacked input signals
`{'double'}` (default) | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `boolean`

Data types of input signals (unpacked), specified as a cell array. The block creates input ports in the order of the incoming data types specified in the cell array. For example, the first data type in the cell array corresponds to the top port and the last data type corresponds to the bottom port.

For example, if the data types are `single`, `uint8`, and `uint8`, the block creates three input ports. The order of the input port data types is same as the data types specified in the cell array.

Byte alignment — Alignment of input signal data types after packing
1 (default) | 2 | 4 | 8

Each element in the input signal list starts at a multiple of the byte alignment value, specified from the start of the vector. If the byte alignment value is larger than the size of the data type in bytes, the input values are padded with zeros to fill the space allotted.

For example, if the byte alignment value is 4, a `uint32` receives no padding, a `uint16` receives 2 bytes of padding, and a `uint8` receives 3 bytes of padding.

Tip If the model accesses the data items frequently, consider selecting a byte alignment value equal to the largest data type that you want to access. If the model transfers the data items frequently as a group, consider selecting a byte alignment value of 1, which packs the data into the smallest space possible.

Example

Suppose that you are packing four signals into a vector of data type `uint8` or `uint16`, and the signals have these attributes.

Dimension	Size	Type
Vector	3	`int8`
Vector	2	`int16`
Scalar	1	`uint8`
Scalar	1	`uint32`

To pack the signals:

1. Set **Output port (packed) data type**. This example compares `uint8` and `uint16`.
2. Set **Input port data types (cell array)** to:

   ```
   {'int8', 'int16', 'uint8', 'uint32'}
   ```

The block creates four input ports that match the order of the incoming signal data types specified in the cell array.

3 Set the required byte alignment value. The byte alignment value specifies the number of bytes after which a new byte starts from the previous boundary.

The size of the output is based on the packed vector size, the byte alignment value, and the smallest memory cell size of the processor. Depending on the byte alignment value, input values are padded with zeros before the next signal is packed. The smallest addressable memory cell indicates the number of bits occupied by the `char` or `uint8` data type for a processor and determines the structure of packets.

4 Connect incoming signals to the input port of the Byte Pack block.

For processors with a smallest addressable memory cell of 8 bits per char, consider these values for input signals.

Unpacked Signals				
Dimension	**Size**	**Data Type**	**Dec Value**	**Hex Value**
Vector	3	`int8`	35	23
			4	04
			–3	FD
Vector	2	`int16`	218	00DA
			–12	FFF4
Scalar	1	`uint8`	112	70
Scalar	1	`uint32`	5000	00001388

The packed output vector data type `uint8` is:

Packed Vector Type - uint8

Packet	23	04	FD	DA	00	F4	FF	70	88	13	00	00
Alignment	1	1	1	1	1	1	1	1	1	1	1	1

Packet	23	04	FD	00	DA	00	F4	FF	70	00	88	13	00	00
Alignment	2		2		2		2		2		2		2	

Packet	23	04	FD	00	DA	00	F4	FF	70	00	00	00	88	13	00	00
Alignment	4				4				4				4			

Packet	23	04	FD	00	00	00	00	00	DA	00	F4	FF	00	00	00	00	70	00	00	00	00	00	00	00	88	13	00	00	00	00	00	00
Alignment	8								8								8								8							

Red zeros represent padded empty memory cells.

For a packed output vector of data type `uint8` and byte alignment value 2, the `int8` data value (23 04 FD) occupies the first three memory cells, with each cell occupying 8 bits. Because three is not a multiple of the byte alignment value 2, the next input signal of `int16` data value (00DA FFF4) is allocated the next four cells (fifth through eighth), leaving the fourth cell empty. The block fills the empty cell with zero. The rest of the input signals are packed in a similar way.

After packing all input signals, the Byte Pack block calculates the total packets allocated and outputs a `uint8` vector of size 4 + 4 + 2 + 4 = 14. Here, the `int8` signal occupies the first 4 cells, the `int16` signal occupies the second 4 cells, the `uint16` signal occupies the third 2 cells, and the `uint32` signal occupies the fourth 4 cells.

The packed output vector of data type `uint16` is:

Packed Vector Type - uint16

Packet	0423	DAFD	F400	70FF	1388	0000
Alignment	1 1	1 1	1 1	1 1	1 1	1 1

Packet	0423	00FD	00DA	FFF4	0070	1388	0000
Alignment	2	2	2	2	2	2	2

Packet	0423	00FD	00DA	FFF4	0070	0000	1388	0000
Alignment	4		4		4		4	

Packet	0423	00FD	0000	0000	00DA	FFF4	0000	0000	0070	0000	0000	0000	1388	0000	0000	0000
Alignment	8				8				8				8			

For processors such as Texas Instruments C2000, with the smallest addressable memory cell of 16 bits per char, consider these values for input signals. The `int8` and `uint8` data values occupy 16 bits, as indicated by the hex value.

Unpacked Signals				
Dimension	**Size**	**Data Type**	**Dec Value**	**Hex Value**
Vector	3	`int8`	35	0023
			4	0004
			–3	FFFD
Vector	2	`int16`	218	00DA
			–12	FFF4
Scalar	1	`uint8`	112	0070
Scalar	1	`uint32`	5000	00001388

For the packed output vector of data type `uint8`, the output packet occupies 16 bits, although the data value the packet represents is 8 bits. The byte alignment values are calculated with respect to the16-bit addressable memory.

Packed Vector Type - uint8

Packet	0023	0004	00FD	00DA	0000	00F4	00FF	0070	0088	0013	0000	0000
Alignment	1	1	1	1	1	1	1	1	1	1	1	1

Packet	0023	0004	00FD	0000	00DA	0000	00F4	00FF	0070	0000	0088	0013	0000	0000
Alignment	2		2		2		2		2		2		2	

Packet	0023	0004	00FD	0000	00DA	0000	00F4	00FF	0070	0000	0000	0000	0088	0013	0000	0000
Alignment	4				4				4				4			

Packet	0023	0004	00FD	0000	0000	0000	0000	0000	00DA	0000	00F4	00FF	0000	0000	0000	0000	0070	0000	0000	0000	0000	0000	0000	0000	0088	0013	0000	0000	0000	0000	0000	0000
Alignment	8								8								8								8							

For a packed output vector of data type `uint8` and byte alignment value 2, the `int8` data value (0023 0004 00FD) occupies the first three memory cells, with each cell occupying 16 bits. Because three is not a multiple of byte alignment value 2, the next signal of data type `int16` (00DA 0000 00F4 00FF) is allocated the next four cells (fifth through eighth), leaving the fourth cell empty. The block fills the empty cell with zero. The rest of the input signals are packed in a similar way. After packing all input signals, the Byte Pack block calculates total packets allocated and outputs a `uint8` vector of size 4 + 4 + 2 + 4 = 14.

For the packed output vector of data type `uint16`, the output packet occupies 16 bits, and the data value the packet represents is also 16 bits. For a packet size of 16 and larger, the byte alignment is calculated with respect to the number of bytes the data values are packed into. Therefore, in this case, 1-byte alignment is not allowed.

Packed Vector Type - uint16

Packet	NA
Alignment	1

Packet	0423	00FD	00DA	FFF4	0070	1388	0000
Alignment	2	2	2	2	2	2	2

Packet	0423	00FD	00DA	FFF4	0070	0000	1388	0000
Alignment	4		4		4		4	

Packet	0423	00FD	0000	0000	00DA	FFF4	0000	0000	0070	0000	0000	0000	1388	0000	0000
Alignment	8				8				8				8		

For a packed output of data type `uint16` and byte alignment value 2, the three `int8` data values (0423 FD) are packed together as two words in the first two memory cells. The fourth byte in the second memory cell is empty and filled with zero. The `int16` data value (00DA FFF4) is allocated the next two memory cells (third and fourth). The rest of the input signals are packed in a similar way. After packing all signals, the Byte Pack block calculates total packets allocated and outputs a `uint16` vector of size 2 + 2 + 1 + 2 = 7.

Version History

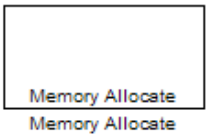
Introduced in R2016b

See Also

Byte Unpack

Memory Allocate

Allocate memory for new variable



Libraries:
C2000 Microcontroller Blockset / Memory Operations

Description

The Memory Allocate block, on C2xxx processors, directs the TI compiler to allocate a memory location for a new variable. Block parameters specify the variable name, the alignment of the variable in memory, the data type of the variable, and other features that fully define the memory required.

The block does not verify whether the parameter settings for the variable are valid, such as checking the variable name, data type, or section. You must check that the parameters settings are valid.

You do not connect the Memory Allocation block to other blocks in a model.

Parameters

Memory

Allocate memory for storing the variable. Specify the data type and size.

Variable name — Name for variable
myVariable (default) | string | character vector

Specify the name of the variable for which to allocate memory. The variable is allocated in the generated code.

Specify variable alignment — Variable alignment flag
off (default) | on

Select this parameter, if required by your target processor, to direct the compiler to align the new variable to a byte alignment boundary.

Parameter Dependencies

If you select this parameter, use parameter **Memory alignment boundary** to set the byte alignment boundary.

Memory alignment boundary — Memory alignment for variable
4 (default) | 1 | 2 | 8

Specify the alignment boundary for the variable data type in bytes. Alignment can occur on 1-, 2-, 4-, or 8-byte boundaries. If the variable contains multiple values, such as a vector or an array, the block aligns elements according to rules applied by the compiler.

Parameter Dependencies

To enable this parameter, select **Specify variable alignment**.

Data type — Data type for variable
uint32 (default) | double | single | int8 | uint8 | int16 | uint16 | int32 | int64 | uint64 | boolean

Specify the data type for the variable.

Specify data type qualifier — Data type qualifier flag
off (default) | on

Select this parameter to specify a data type qualifier to apply to the variable.

Parameter Dependencies

If you select this parameter, use parameter **Data type qualifier** to set the data type qualifier to apply to the variable.

Data type qualifier — Data type qualifier for variable
volatile (default) | sting | character vectory

Specify the data type qualifier to apply to the variable in generated code as a string or character vector. Common qualifiers are `volatile`, `const`, `static`, and `register`. The block does not check whether the value that you enter is a valid qualifier.

Data dimension — Number of elements of variable data type
64 (default) | positive integer

Specify the number of elements of the specified data type for the variable as a positive integer.

Initialize memory — Memory initialization flag
off (default) | on

Select this parameter to specify an initial value for the variable.

Parameter Dependencies

If you select this parameter, use parameter **Initial value** to set the initial value.

Initial value — Initial value for variable
0 (default) | scalar | vector | matrix

Specify the initial value for the variable. At run time, the block sets the memory location to this value.

Parameter Dependencies

To enable this parameter, select **Initialize memory**.

Section

Specify the memory section in which to allocate the variable.

Specify memory section — Memory section flag
off (default) | on

Select this parameter to specify a memory section to use for allocating space in memory for the variable.

Parameter Dependencies

If you select this parameter, use parameters **Memory section**, **Bind memory section**, **Section start address** to specify memory section details.

Memory section — Memory section for variable
mySEC1 (default) | string | character vector

Specify the name of the memory section to use for allocating memory for the variable as a string or character vector. Specify a standard memory section or a custom memory section that you declare elsewhere in your code.

Verify that the memory section has enough space to store the variable.

Parameter Dependencies

- To enable this parameter, select **Specify memory section**.
- To bind the specified memory section to a specific start address in memory, select **Bind memory section** and specify the address by entering a value for **Section start address**.

Bind memory section — Bind memory section to start address flag
off (default) | on

Select this parameter to bind a newly created memory section for the variable to a specific start address.

The new memory section specified for **Memory section** is defined when you select this parameter.

Parameter Dependencies

- Select this parameter to enable parameter **Section start address**.
- Do not select this parameter if you are associating the variable with an existing memory section.

Section start address — Start address of memory section for variable
hex2dec('8000') (default) | memory address in decimal or hexadecimal form

Specify the start address to which to bind the memory section for the variable in decimal form or in hexadecimal form with a conversion to decimal as shown by the default value `hex2dec('8000')`. The block does not verify the address. Verify that the address that you specify exists and that it can contain the specified memory section.

Parameter Dependencies

- Enable this parameter by selecting parameter **Bind memory section**.
- Do not specify a value for this parameter if you are associating the variable with an existing memory section.

Version History

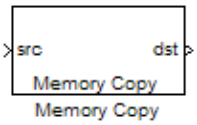
Introduced in R2011a

See Also

Memory Copy

Memory Copy

Copy data from and to memory section



Libraries:
C2000 Microcontroller Blockset / Memory Operations

Description

Generated code for the Memory Copy block copies data from and to processor memory as configured by block parameters. When you use this block to copy an individual data element from a source to a destination, the block copies the element from the source, using the source data type, and then casts the data element to the specified destination data type.

Include as many instances of the Memory Copy block in a model as required to manipulate memory on a target processor. Each instance of the block works with one variable, address, or set of addresses provided to the block as input.

Specify the source and destination for a memory copy by using block parameters. You can use block parameters to control other aspects of a memory copy, such as:

- Initialization for memory locations
- Memory stride and offset during run time
- Write operations to memory during program initialization, during program termination, and at every sample time
- Insertion of custom ANSI® C source code before and after each memory copy read and write operation (for example, to lock and unlock registers before and after accessing them)

The Memory Copy block performs operations at three periods during program execution:

- Initialization
- Real-time operations
- Termination

You can use block parameters to control when and how the block initializes memory, copies data or variables to and from memory, and terminates copy operations. The parameters enable you to turn on and off memory copy operations in the three periods independently.

Use the Memory Copy block and the Memory Allocate block to manipulate and allocate memory for custom device drivers, such as PCI bus drivers or codec-style drivers.

During simulation, the Memory Copy block does not perform an operation. The block output is not defined.

Ports

Input

src — Input data for copy operation
scalar | vector

The source data for the memory copy operation, specified as a scalar or vector.

Port Dependencies

To use this port as the source for the memory copy operation, set parameter **Copy from** to `Input port`.

Data Types: `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

&src — Address of input data for copy operation
scalar | vector

The memory address of source data for the copy operation, specified as a scalar or vector.

Port Dependencies

To use this port as the source for the memory copy operation, set parameter **Copy from** to `Specified address` and **Specify address source** to `Input port`. The Copy Memory block converts input port `src` to `&src`.

Data Types: `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

src ofs — Offset for data read during copy operation
scalar | vector

Offset to use for data read during the copy operation, specified as a scalar or vector.

Port Dependencies

To create this port, select parameter **Use offset when reading** and set **Specify offset source** to `Input port`.

Data Types: `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

&dst — Address of output data for copy operation
scalar | vector

The memory address to use as the data destination for the copy operation, specified as a scalar or vector.

Port Dependencies

To use this port as the destination for the memory copy operation, set parameter **Copy to** to `Specified address` and **Specify address source** to `Input port`. The Copy Memory block converts output port `dst` to input port `&dst`.

Data Types: `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

dst ofs — Offset for data written during copy operation
scalar | vector

Offset to use for data write during the copy operation, specified as a scalar or vector.

Port Dependencies

To create this port, select parameter **Use offset when writing** and set **Specify offset source** to `Input port`.

Data Types: `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Output

dst — Output data for copy operation
scalar | vector

The data copied, specified as a scalar or vector.

Port Dependencies

To use this port as the destination for the memory copy operation, set parameter **Copy to** to `Output port`.

Data Types: `single` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64`

Parameters

Source

Specify the source sequential memory location for the copy operation. Specify the data type, size, and other attributes of the source variable.

Copy from — Input source for copy operation
Input port (default) | Specified address | Specified source code symbol

Specify the input source for the data read part of the copy operation. Choose from the sources listed in this table.

Source of Data Read	Parameter Value to Specify
`src` input port	`Input port`
Memory address	`Specified address`
Symbol (variable) in source code lookup table	`Specified source code symbol`

Parameter Dependencies

- If you select `Specified address`, use **Specify address source** to specify the source of the memory address and **Address** to specify the address.
- If you select `Specified source code symbol`, use **Source code symbol** to specify the symbol (variable) in the source code symbol table to copy.
- If you select `Specified address` or `Specified source code symbol`, change **Data type** to a value other than `Inherit from source` (the default). If you do not make this change, you receive an error message indicating that the data type cannot be inherited because the input port does not exist.

Specify address source — Source of memory address for input data
Specify via dialog (default) | Input port

Specify the source of the memory address of the input source for the copy operation. To specify a memory address for the source variable, select `Specify via dialog`. That selection enables an **Address** parameter that you use to specify the memory address.

To specify that the block get the address from the input port, select `Input port`. When you select `Input port`, the block input port label changes to `&src`.

Parameter Dependencies

- To enable this parameter, set **Copy from** to `Specified address`.
- If you select `Specify via dialog`, this parameter enables the **Address** parameter, which you use to specify the address of the source variable.
- If you select `Specify via dialog`, set **Data type** to a value other than `Inherit from source` (the default). If you do not make this change, you receive an error message indicating that the data type cannot be inherited because the input port does not exist.
- If you select `Inport port` , specify a data type for the **Data type** parameter.

Address — Memory address of source data
hex2dec('00001000') (default) | memory address in decimal or hexadecimal form with a conversion to decimal

Specify the memory address of the source data in decimal form or in hexadecimal form with a conversion to decimal as shown by the default value `hex2dec('00001000')`.

This example converts `0x1000` to decimal form.

```
4096 = hex2dec('1000');
```

For this example, you can specify the address as `4096` or `hex2dec('1000')`.

Parameter Dependencies

To enable this parameter, set **Copy from** to `Specified address` and **Specify address source** to `Specify via dialog`.

Source code symbol — Symbol in source code symbol table
myVariableSrc (default) | string | character vector

Specify the symbol (variable) in the source code symbol table to copy. The symbol that you specify must exist in the symbol table for your program. The block does not verify whether the symbol exists in the symbol table and whether you specify the symbol with valid syntax. Enter text that specifies the symbol exactly as it appears in your code.

Parameter Dependencies

- To enable this parameter, set **Copy from** to `Specified source code symbol`.
- Set **Data type** to a value other than `Inherit from source` (the default). If you do not make this change, you receive an error message indicating that the data type cannot be inherited because the input port does not exist.

Data type — Data type of data being copied
uint8 (default) | double | single | int8 | int16 | uint16 | int32 | uint32 | int64 | uint64 | boolean | Inherit from input port

Specify the data type of the source data being copied. To inherit the data type from the `src` input port, select `Inherit from input port`.

Data length — Number of data elements to copy
1 (default) | positive integer

Specify the number of elements to copy from the source location. Each element has the data type specified by the **Data type** parameter.

Use offset when reading — Use offset when reading input
off (default) | on

Specify whether the block uses an offset when reading input. The offset value is in elements of the specified data type. Specify the source of the offset by using the **Specify offset source** parameter.

Parameter Dependencies

- If you select this parameter, use **Specify offset source** to specify the source of the offset.
- Use **Offset** to specify the offset value.

Specify offset source — Source of input offset
Specify via dialog (default) | Input port

Specify the source of the input offset for the copy operation. To specify an offset value, select `Specify via dialog`. That selection enables an **Offset** parameter that you use to specify the offset value.

To specify for the block to get the offset from an input port, select `Input port`. When you select `Input port`, the block creates an input port labeled `src ofs` and reads the offset value from that port. The `src ofs` port enables your program to change the offset dynamically during program execution.

Parameter Dependencies

To enable this parameter, select **Use offset when reading**.

Offset — Number of values to skip before copying first value to destination
0 (default) | positive integer

Before copying the first value to the destination, specify the number of values to skip.

Parameter Dependencies

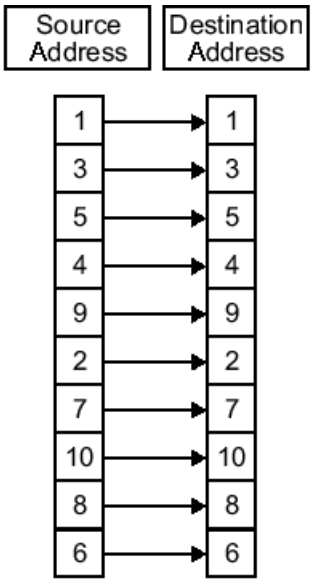
To enable this parameter, select **Use offset when reading** and set **Specify offset source** to `Specify via dialog`.

Stride — Spacing for reading input
1 (default) | positive integer

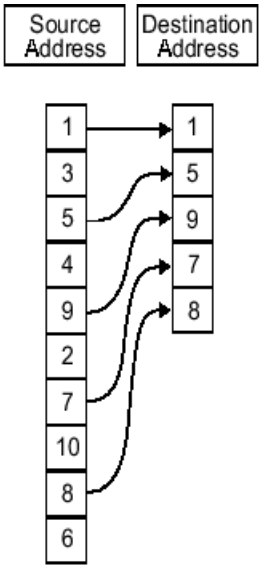
Specify the spacing for reading the input. By default, the stride value is one, meaning that the generated code reads the input data sequentially. When you add a stride value that is not equal to one, when reading input data, the generated code skips spaces in the source address equal to the stride.

These figures show the stride concept. In the first figure, data is copied without a stride. The second figure shows the results of a stride value of two. You can specify a stride value for the block output

with parameter **Stride** on the **Destination** tab. You can also compare stride with the offset to see the differences.



Input Stride = 1
Output Stride = 1
Number of Elements Copied = 10



Input Stride = 2
Output Stride = 1
Number of Elements Copied = 5

Destination

Specify the destination memory location for the copy operation. Specify the attributes of the destination.

Copy to — Type of output destination for copy operation
Output port (default) | Specified address | Specified destination code symbol

Specify the type of output destination for the copy operation. Select one of the values listed in this table.

Parameter Value	Destination of Data Write
`Output port`	Block `dst` output port
`Specified address`	Memory location specified by parameters **Specify address destination** and **Address**
`Specified source code symbol`	Symbol (variable) specified by parameter **Source code symbol**

Parameter Dependencies

- If you select `Specified address`, use **Specify address destination** to specify the destination memory location.
- If you select `Specified source code symbol`, use **Destination code symbol** to specify the symbol (variable) in the source code symbol table to which to copy the variable.

Specify address source — Source of memory address for output destination
Specify via dialog (default) | Input port |

Specify the source of the destination memory address of the variable for the copy operation. To specify a memory address for the variable, select `Specify via dialog`. That selection enables an **Address** parameter that you use to specify the memory address. To specify that the block get the address from an input port, select `Input port`. When you select `Input port`, the block creates an input port labeled `&dst`. Changing the address dynamically means that you can use the block to copy different variables by providing the variable address from an upstream block in the model.

Parameter Dependencies

- To enable this parameter, set **Copy to** to `Specified address`.
- If you select `Specify via dialog`, this parameter enables the **Address** parameter, which you use to specify the address of the destination variable.

Address — Memory address of destination variable
hex2dec('00002000') (default) | memory address in decimal or hexadecimal form with a conversion to decimal

Specify the memory address of the destination variable in decimal form or in hexadecimal form with a conversion to decimal as shown by the default value `hex2dec('00001000')`.

This example converts `0x2000` to decimal form.

```
8192 = hex2dec('2000');
```

For this example, you can specify the address as `8192` or `hex2dec('2000')`.

Parameter Dependencies

To enable this parameter, set **Copy to** to `Specified address`and **Specify address source** to `Specify via dialog`.

Source code symbol — Symbol in source code symbol table
myVariableDst (default) | string | character vector

Specify the symbol (variable) in the source code symbol table to which to copy the variable. The symbol that you specify, must exist in the symbol table for your program. The block does not verify whether the symbol exists in the symbol table and whether you specify the symbol with valid syntax. Enter text that specifies the symbol exactly as it appears in your code.

Parameter Dependencies

To enable this parameter, set **Copy to** to `Specified source code symbol`.

Data type — Data type of variable
uint32 (default) | double | single | int8 | uint8 | int16 | uint16 | uint32 | int64 | uint64 | boolean | Inherit from source

Specify the data type of the source variable. To inherit the data type from the source variable, select `Inherit from source`.

Use offset when writing — Use offset when writing output
off (default) | on

Specify whether the block uses an offset when writing output. The offset value is in elements of the specified data type. Specify the source of the offset by using the **Specify offset source** parameter.

Parameter Dependencies

If you select this parameter, use **Specify offset source** to specify the source of the offset. Use **Offset** to specify the offset value.

Specify offset source — Source of offset for output destination
Specify constant value (default) | Specify source code symbol

Specify the source of the offset for the output destination for the copy operation. To specify an offset value for the destination variable, select `Specify via dialog`. That selection enables an **Offset** parameter that you use to specify the offset value.

To specify that the block get the offset from the input port, select `Input port`. When you select `Input port`, the block adds an input port labeled `dst ofs` and reads the offset value from that port. The `dst ofs` port enables your program to change the offset dynamically during execution.

Parameter Dependencies

To enable this parameter, select **Specify offset source**.

Offset — Number of values to skip before writing first value to destination
0 (default) | positive integer

Before writing the first value to the destination, specify the number of values to skip.

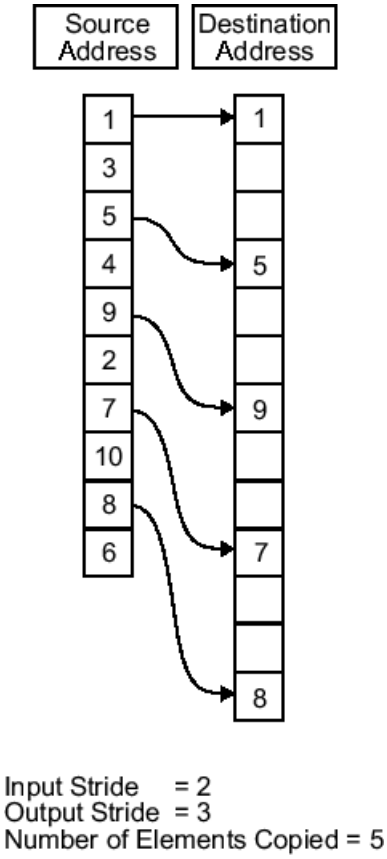
Parameter Dependencies

To enable this parameter, select **Use offset when writing** and set **Specify offset source** to `Specify via dialog`.

Stride — Spacing for writing output
1 (default) | positive integer

Specify the spacing for writing the output. By default, the stride value is one meaning that the generated code writes the input data sequentially to the destination in consecutive locations. When you add a stride value that is not equal to one, when writing input data, the generated code skips spaces in the destination address equal to the stride.

This figure shows a stride value of three applied to writing the input to an output location. You can specify a stride value for the input with parameter **Stride** on the **Source** pane. As shown in the figure, you can use an input stride and an output stride at the same time to enable manipulating memory more fully.



Sample time — Rate of memory copy
inf (default) | scalar

Specify the rate at which the memory copy operation occurs in seconds. To use a constant sample time, specify `Inf`. To inherit the sample time from the input port or, when the block does not have an input port, from the Simulink model, specify `-1`.

Options

Configure parameters that control the copy process.

Set memory value at initialization — Set memory address during initialization
off (default) | on

Specify whether to initialize the memory address to a specific value during program initialization.

Parameter Dependencies

If you select this parameter, use a combination of parameters to configure the initialization value.

What to Configure	Parameter
Source of the initialization value	**Specify initialization value source**
Initialization value as a constant	**Initialization value (constant)**
Initialization value as a variable	**Initialization value (source code symbol)**
Initialization value as a mask to manipulate register contents at the bit level	**Apply initialization value as mask**
Apply a mask value	**Bitwise operator**

Specify initialization value source — Source of initialization value
Specify constant value (default) | Specify source code symbol

Specify the source of the initial value. To configure the source for initializing memory as a specific value, select `Specify constant value`. To configure the source as a variable (a symbol), select `Specify source code symbol`.

Parameter Dependencies

- To enable this parameter, select **Set memory value at initialization**.
- Use **Initialization value (constant)** or **Initialization value (source code symbol)** to specify the initial value.

Initialization value (constant) — Constant initialization value
1 (default) | scalar

Specify a constant value.

Parameter Dependencies

To enable this parameter, select **Set memory value at initialization** and set **Set initialization value source** to `Specify constant value`.

Initialization value (source code symbol) — Symbol in source code symbol table
myInitValueVariable (default) | string | character vector

Specify the symbol (variable) in the source code symbol table to use for the initialization value. The symbol that you specify must exist in the symbol table for your program. The block does not verify whether the symbol exists in the symbol table and whether you specify the symbol with valid syntax. Enter text that specifies the symbol exactly as it appears in your code.

Parameter Dependencies

To enable this parameter, select **Set memory value at initialization** and set **Set initialization value source** to `Specify source code symbol`.

Apply initialization value as mask — Apply initialization value as mask
off (default) | on

Specify whether to use the initialization value as a mask to manipulate register content at the bit level. Your initialization value is treated as a string of bits for the mask.

To define how to apply the mask value, specify a value for the **Bitwise operator** parameter.

To use your initialization value as a mask, the output from the copy must be a specific address. The output:

- Can be a symbol
- Cannot be an output port

Parameter Dependencies

If you select this parameter, use **Bitwise operator** to define how to apply the mask value.

Bitwise operator — Type of bitwise operation
bitwise AND (default) | bitwise OR | bitwise exclusive OR | left shift | right shift

Specify the type of bitwise operation to use as a mask to manipulate the memory value. Applying a mask to the copy process means that you can select individual bits in the result. For example, by applying a mask, you can read the value of the fifth bit.

Select one of the bitwise operations in this table.

Bitwise Operation	Description
`bitwise AND`	Apply the mask value as a bitwise AND to the value in the register.
`bitwise OR`	Apply the mask value as a bitwise OR to the value in the register.
`bitwise exclusive OR`	Apply the mask value as a bitwise exclusive OR to the value in the register.
`left shift`	Shift the bits in the register left by the number of bits represented by the initialization value. For example, if your initialization value is 3, the block shifts the register value to the left 3 bits. In this case, the value must be a positive integer.
`right shift`	Shift the bits in the register to the right by the number of bits represented by the initialization value. For example, if your initialization value is 6, the block shifts the register value to the right 6 bits. In this case, the value must be a positive integer.

Parameter Dependencies

To enable this parameter, select **Apply initialization value as mask**.

Set memory value at termination — Copy memory during program termination
off (default) | on

Specify that your program copy memory during program termination. Copying a value in memory during termination occurs in addition to a copy during program initialization.

Parameter Dependencies

If you select this parameter, you can use **Set memory value only at initialization/termination** to limit copy operations to occur during program initialization and termination only.

Termination value — Termination value
1 (default) | scalar | vector | matrix

Specify a value to write to memory during program termination.

Parameter Dependencies

To enable this parameter, select **Set memory value at termination**.

Set memory value only at initialization/termination — Copy memory value during program initialization and termination only
off (default) | on

Specify whether to perform copies during program initialization and termination only. When this parameter is cleared, the block performs copies during initialization, real-time operations, and termination. If you select this parameter, the block performs copies during initialization and termination only.

Insert custom code before memory write — Insert custom code before memory write
off (default) | on

Specify whether the code generator inserts custom ANSI C code immediately before the program writes to the specified memory location. You can use this parameter and **Insert custom code after memory write** to lock and unlock registers before and after accessing them. For example, some processors have registers that you might need to unlock and lock with EALLOW and EDIS macros before and after your program accesses them.

Parameter Dependencies

If you select this parameter, use **Custom code** to specify the custom ANSI C code to insert into the generated code immediately before the memory write operation.

Insert custom code after memory write — Custom code after memory write flag
off (default) | on

Specify whether the code generator inserts custom ANSI C code immediately after the program writes to the specified memory location. You can the **Insert custom code before memory write** and this parameter to lock and unlock registers before and after accessing them. For example, some processors have registers that you might need to unlock and lock with EALLOW and EDIS macros before and after your program accesses them.

Parameter Dependencies

If you select this parameter, use **Custom code** to specify the custom ANSI C code to insert into the generated code immediately after the memory write operation.

Custom code —
/* Custom Code Before Write*/ or /* Custom Code After Write*/ (default) | string | character vector

Specify custom ANSI C code to insert into the generated code immediately before or immediately after the memory write operation. Code that you specify appears in the generated code exactly as you enter it.

Parameter Dependencies

To enable this parameter, select **Insert custom code before memory write** or **Insert custom code after memory write**.

Version History

Introduced in R2011a

See Also

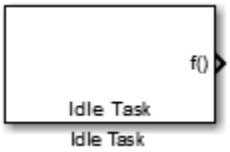
Memory Allocate

Idle Task

Create free-running task

<softwaremeta type="block" version="5.0-variant tmwbook5.0" xml:base="../../../shareddoc/prod_softwaremeta/c2b/block_idletask_softwaremeta.xml">
<librarypath> C2000 Microcontroller Blockset / Scheduling </librarypath>
<extendedcapabilities></extendedcapabilities>
</softwaremeta>

Description



The Idle Task block, and the subsystem connected to it, specify one or more functions to execute as background tasks. The tasks executed through the Idle Task block are of the lowest priority, lower than that of the base rate task.

This block is not supported on targets running an operating system or RTOS.

Vectorized Output

The block output comprises a set of vectors—the task numbers vector and the preemption flag or flags vector. A preemption-flag vector must be the same length as the number of tasks vector unless the preemption flag vector has only one element. The value of the preemption flag determines whether a given interrupt (and task) is preemptable. Preemption overrides prioritization. A lower-priority nonpreemptable task can preempt a higher-priority preemptable task.

When the preemption flag vector has one element, that element value applies to the functions in the downstream subsystem as defined by the task numbers in the task number vector. If the preemption flag vector has the same number of elements as the task number vector, each task defined in the task number vector has a preemption status defined by the value of the corresponding element in the preemption flag vector.

Parameters

Task numbers

Identifies the created tasks by number. Enter as many tasks as you need by entering a vector of integers. The default values are `[1,2]` to indicate that the downstream subsystem has two functions.

The values you enter determine the execution order of the functions in the downstream subsystem, while the number of values you enter corresponds to the number of functions in the downstream subsystem.

Enter a vector containing the same number of elements as the number of functions in the downstream subsystem. This vector can contain up to 16 elements, and the values must be from 0 to 15 inclusive.

The value of the first element in the vector determines the order in which the first function in the subsystem is executed, the value of the second element determines the order in which the second function in the subsystem is executed, and so on.

For example, entering [2,3,1] in this field indicates that there are three functions to be executed, and that the third function is executed first, the first function is executed second, and the second function is executed third. After the functions are executed, the Idle Task block cycles back and repeats the execution of the functions in the same order.

Preemption flags

Higher-priority interrupts can preempt interrupts that have lower priority. To allow you to control preemption, use the preemption flags to specify whether an interrupt can be preempted.

Entering `1` indicates that the interrupt can be preempted. Entering `0` indicates the interrupt cannot be preempted. When **Task numbers** contains more than one task, you can assign different preemption flags to each task by entering a vector of flag values, corresponding to the order of the tasks in **Task numbers**. If **Task numbers** contains more than one task, and you enter only one flag value here, that status applies to the tasks.

In the default settings `[0 1]`, the task with priority `1` in **Task numbers** is not preemptable, and the priority `2` task can be preempted.

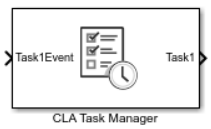
Enable simulation input

When you select this option, Simulink software adds an input port to the Idle Task block. This port is used in simulation only. Connect one or more simulated interrupt sources to the simulation input.

Note Select this check box to test asynchronous interrupt processing behavior in Simulink software.

CLA Task Manager

Create and manage task executions in Control Law Accelerator (CLA) model



Libraries:
C2000 Microcontroller Blockset / Scheduling

Description

The CLA Task Manager block simulates the execution of software tasks as they would be expected to behave on a CLA processor. Using this block, you can add and remove event-driven tasks from your model. Tasks can be represented as function-call subsystems contained inside a single Model block. The CLA Task Manager block executes individual tasks based on their parameters. Task parameters include duration, trigger, and priority

Note The CLA Task Manager block cannot be used in a referenced model. For more information on referenced models, see the Model block.

The CLA Task Manager block provides three methods to specify the duration of a task in simulation:

- A probability model of task duration defined in the block mask.
- A data file recording of either a previous task simulation or from a task on an SoC device.
- Input ports, which you can connect to more dynamic models of task duration.

Limitations

- A model containing a CLA Task Manager blocks does not support simulation stepping. For more information on simulation stepping, see “Debug Simulations in the Simulink Editor”.

Ports

Output

Task1 — Function-call from Task1
scalar

A function-call signal that can trigger event-driven tasks. The CLA Model block represents these tasks as function-call subsystems.

To show the function-call port from an event-driven subsystem contained in a Function-Call Subsystem block on the Model block, include an Inport in the processor Model block connected to the function-call trigger port of the subsystem. In the Inport, select **Block Parameters > Signal Attributes > Output function call**.

Note The `Task1` port must be connected to a function-call port on a Model block.

Dependencies

To create or remove a control signal port for a task, add or remove the task from the CLA Task Manager block by clicking the **Add** or **Delete** buttons in the block dialog mask.

Input

***Task1*Event** — Message event notification
scalar

A message port that triggers the associated event-driven task. The `Task1Event` port receives the message from a Software Trigger CPU<->CLA, PWM Interface, or ADC Interface block. For more information on messages, see “Messages”.

Dependencies

To show a *Task1*`Event` port, set the **Type** parameter of *Task1* to `Event-driven`.

Data Types: `rteEvent`

***Task1*Dur** — Task duration
positive scalar

A positive-value signal that specifies the execution duration of a task at the present time. For more information on specifying task duration, see “Task Duration” (SoC Blockset).

Dependencies

To enable this port, set the **Specify task duration via** parameter to `Input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Parameters

Enable task simulation — Option to enable simulation of task duration
`on` (default) | `off`

Enable or disable the simulation of task duration. If you clear this parameter, tasks simulate using a function-call generator. Event-driven tasks inherit their period from the fundamental sampling time of the model. Timer-driven tasks inherit their period from the block dialog mask.

List of tasks — List of tasks
`Task1` (default)

List of the tasks generated by the CLA Task Manager block. Each task has a set of parameters listed in the **Main** and **Simulation** tabs of the block dialog mask.

Add — Option to add task
button

Add a task to the CLA Task Manager block. During deployment, the generated code encapsulates each task as an interrupt. The **Main** parameters for each task define the properties of its interrupt. During simulation, the task uses a combination of the **Main** and **Simulation** parameters for that task.

Delete — Option to delete existing task
button

Remove a task from the Task Manager.

Dependencies

To enable this parameter, specify at least two tasks.

Main

Name — Name of task
`Task1` (default) | character vector

Unique name of the task. The task name must contain only alphanumeric characters and underscores.

Priority — Priority of task in scheduler
`10` (default) | integer in the range [1, 99]

Specify the scheduler's priority for the event-driven task as an integer in the range [1, 99]. Higher-priority tasks can preempt lower-priority tasks. The hardware attributes limit the task priority range. For more information on task priority, see “Task Priority and Preemption” (SoC Blockset).

Dependencies

To enable this parameter, set `Type` to `Event-driven`.

Simulation

Specify task duration via — Source of task execution time
`Dialog` (default) | `Input port` | `Record task execution statistics`

Specify the source of timing information for task execution as one of these options.

- `Dialog` — Use a normally-distributed probabilistic model with **Mean**, **Deviation**, **Min**, and **Max** defined in the block dialog mask.
- `Input port` — The block input port dynamically defines the execution duration.
- `Record task execution statistics` — Use a normally-distributed probabilistic model with mean and deviation defined in the file specified by **File name**.

For more information on configuring task duration, see “Task Duration” (SoC Blockset).

Task duration settings

Add — Option to add distribution
button

Adds a distribution to the set of normal distributions that generates an execution duration. For more information on configuring task duration, see “Task Duration” (SoC Blockset).

Note Only a maximum five distributions can be assigned to a single task.

Delete — Option to remove distribution
button

Remove a distribution from the set of normal distributions.

Percent — Likelihood of distribution
`100` (default) | positive scalar

Specify the likelihood of each normal distribution. The **Percent** weighted sum of normal distributions determines the task duration likelihood. For more information on configuring task duration, see “Task Duration” (SoC Blockset).

Note The sum of **Percent** for all the distributions in a single task must equal 100.

Mean — Mean task duration in simulation
`1e-06` (default) | positive scalar

Specify the mean task duration during simulation of the task. The simulated task duration uses a normal distribution with specified **Mean** and **SD** parameter values as a first-order approximation of the task behavior. For more information on configuring task duration, see “Task Duration” (SoC Blockset).

SD — Standard deviation of task duration in simulation
`0` (default) | positive scalar

Specify the standard deviation task duration during simulation of the task. The simulated task duration uses a normal distribution with specified **Mean** and **SD** as a first-order approximation of the task behavior. For more information on configuring task duration, see “Task Duration” (SoC Blockset).

Min — Lower limit of task duration
`1e-06` (default) | positive scalar

Lower limit of a task duration distribution. For more information on configuring task duration, see “Task Duration” (SoC Blockset).

Max — Upper limit of task duration
`1e-06` (default) | positive scalar

Upper limit of a task duration distribution. For more information on configuring task duration, see “Task Duration” (SoC Blockset).

Version History

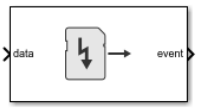
Introduced in R2022a

See Also

Task Manager | SW Trigger CPU<->CLA

Software Trigger CPU<->CLA

Trigger software events between processor (CPU) and control law accelerator (CLA)



Libraries:
C2000 Microcontroller Blockset / Scheduling

Description

The Software Trigger CPU<->CLA block simulates the triggering of software events between processor (CPU) and CLA in a supported Texas Instruments hardware board. When you use this block as a task on a CPU, the block generates an event to execute tasks on the connected CLA unit. When you use this block as a task on the CLA, the block generates an event to execute tasks on the connected CPU. Use this block to manage the communication and synchronization between the CPU and CLA.

Ports

Input

data — Event trigger signal
scalar

Specify an event signal to start the software interrupt in the CPU or CLA.

Data Types: `Boolean`

Output

event — Task event signal
scalar

This port sends a message at whenever the trigger condition occurs. Specify the trigger condition in the **Trigger condition** parameter. This output connects to the input of the Task Manager block to execute the associated event-driven task.

Note The Outport block, at the top-level of the reference model that contains the Software Trigger CPU<->CLA block, that connects to this **event** port must be configured as a non-virtual bus.

Data Types: `rteEvent`

Parameters

Task number — Task identification number
1 (default) | integer from 1 to 8

Specify the number of the CLA task to trigger.

Trigger condition — Condition to trigger an event
`Rising edge` (default) | `Input high`

Generate an event on the **event** port either on a rising edge of the **data** port or whenever the input is high. To generate an event when the input signal changes from zero to one, set this parameter to `Rising edge`. To generate an event once per execution of the task, set this parameter to `Input high`.

Trigger type — Trigger type
`Trig` (default) | `TrigAndWait(codegen)`

Specify the trigger type of on the CPU. To generate an event and continue running the CPU, set this parameter to `Trig`. To generate an event and instruct the CPU to wait for the interrupt to complete, set this parameter to `TrigAndWait(codegen)`. The `TrigAndWait(codegen)` option does not run in simulation.

Dependencies

To enable this parameter, the block must be inside a reference model with the **Processing Unit** set to one of the CPUs.

Version History

Introduced in R2022a

See Also

Task Manager | CLA Task Manager | PWM Interface | “Modeling Control Law Accelerator (CLA) Using Model Reference”

Topics

“Control Law Accelerator in DC-DC Power Conversion” (SoC Blockset)

CAN FD Pack

Pack individual signals into message for CAN FD bus

Libraries:
Vehicle Network Toolbox / CAN FD Communication
Embedded Coder Support Package for Texas Instruments C2000 Processors / Target Communication
Simulink Real-Time / CAN / CAN-FD MSG blocks

Description

The CAN FD Pack block loads signal data into a message at specified intervals during the simulation.

To use this block, you also need a license for Simulink software.

The CAN FD Pack block supports:

- The use of Simulink Accelerator™ mode. Using this feature, you can speed up the execution of Simulink models. For more information, see “Design Your Model for Effective Acceleration”.

Tip

- To work with J1939 messages, use the blocks in the J1939 Communication block library instead of this block.

Ports

Input

Data — CAN FD message signal input
single | double | int8 | int16 | int32 | int64 | uint32 | uint64 | boolean

The CAN FD Pack block has one input port by default. The number of block inputs is dynamic and depends on the number of signals that you specify for the block. For example, if your message has four signals, the block can have four input ports.

Code generation to deploy models to targets. Code generation is not supported if your signal information consists of signed or unsigned integers greater than 32 bits long.

Output

Msg — CAN FD message output
`CAN_FD_MESSAGE_BUS`

This block has one output port, Msg. The CAN FD Pack block takes the specified input signals and packs them into a CAN FD message, output as a Simulink `CAN_FD_MESSAGE_BUS` signal. For more information on Simulink bus objects, see “Composite Interfaces”.

Parameters

Data input as — Select your data signal
`raw data` (default) | `manually specified signals` | `CANdb specified signals`

- `raw data`: Input data as a uint8 vector array. If you select this option, you only specify the message fields. all other signal parameter fields are unavailable. This option opens only one input port on your block.

  The conversion formula is:

  ```
  raw_value = (physical_value - Offset) / Factor
  ```

  where `physical_value` is the original value of the signal and `raw_value` is the packed signal value.
- `manually specified signals`: Allows you to specify data signal definitions. If you select this option, use the **Signals** table to create your signals. The number of block inputs depends on the number of signals you specify.
- `CANdb specified signals`: Allows you to specify a CAN database file that contains message and signal definitions. If you select this option, select a CANdb file. The number of block inputs depends on the number of signals specified in the CANdb file for the selected message.

Programmatic Use
Block Parameter: `DataFormat`
Type: `string` | `character vector`
Values: `'raw data'` | `'manually specified signals'` | `'CANdb specified signals'`
Default: `'raw data'`

CANdb file — CAN database file
character vector

This option is available if you specify that your data is input through a CANdb file in the **Data is input as** list. Click **Browse** to find the CANdb file on your system. The message list specified in the CANdb file populates the **Message** section of the dialog box. The CANdb file also populates the **Signals** table for the selected message. File names that contain non-alphanumeric characters such as equal signs, ampersands, and so on are not valid CAN database file names. You can use periods in your database name. Before you use the CAN database files, rename them with non-alphanumeric characters.

Programmatic Use
Block Parameter: `CANdbFile`
Type: `string` | `character vector`

Message list — CAN message list
array of character vectors

This option is available if you specify that your data is input through a CANdb file in the **Data is input as** field and you select a CANdb file in the **CANdb file** field. Select the message to display signal details in the **Signals** table.

Programmatic Use
Block Parameter: `MsgList`
Type: `string` | `character vector`

Name — CAN FD message name
`CAN Msg` (default) | character vector

Specify a name for your CAN FD message. The default is `CAN Msg`. This option is available if you choose to input raw data or manually specify signals. This option is not available if you choose to use signals from a CANdb file.

Programmatic Use
Block Parameter: `MsgName`
Type: `string` | `character vector`

Protocol mode — CAN FD message protocol
`CAN FD` (default) | `CAN`

Specify the message protocol mode.

Programmatic Use
Block Parameter: `ProtocolMode`
Type: `string` | `character vector`
Values: `'CAN FD'` | `'CAN'`
Default: `'CAN FD'`

Identifier type — CAN identifier type
`Standard (11-bit identifier)` (default) | `Extended (29-bit identifier)`

Specify whether your CAN message identifier is a `Standard` or an `Extended` type. The default is `Standard`. A standard identifier is an 11-bit identifier and an extended identifier is a 29-bit identifier. This option is available if you choose to input raw data or manually specify signals. For `CANdb specified signals`, the **Identifier type** inherits the type from the database.

Programmatic Use
Block Parameter: `MsgIDType`
Type: `string` | `character vector`
Values: `'Standard (11-bit identifier)'` | `'Extended (29-bit identifier)'`
Default: `'Standard (11-bit identifier)'`

Identifier — Message identifier
`0` (default) | `0 .. 536870911`

Specify your message ID. This number must be a positive integer from 0 through 2047 for a standard identifier and from 0 through 536870911 for an extended identifier. You can also specify hexadecimal values by using the `hex2dec` function. This option is available if you choose to input raw data or manually specify signals.

Programmatic Use
Block Parameter: `MsgIdentifier`
Type: `string` | `character vector`
Values: `'0'` to `'536870911'`

Length (bytes) — CAN FD message length
`8` (default) | `0` to `64`

Specify the length of your message. For CAN messages the value can be 0 to 8 bytes; for CAN FD the value can be 0 to 8, 12, 16, 20, 24, 32, 48, or 64 bytes. If you are using `CANdb specified signals` for your data input, the CANdb file defines the length of your message. This option is available if you choose to input raw data or manually specify signals.

Programmatic Use
Block Parameter: `MsgLength`
Type: `string` | `character vector`
Values: `'0'` to `'8'`, `'12'`, `'16'`, `'20'`, `'24'`, `'32'`, `'48'`, `'64'`
Default: `'8'`

Remote frame — CAN message as remote frame
`off` (default) | `on`

(Disabled for CAN FD protocol mode.) Specify the CAN message as a remote frame.

Programmatic Use
Block Parameter: `Remote`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Bit Rate Switch (BRS) — Enable bit rate switch
`off` (default) | `on`

(Disabled for CAN protocol mode.) Enable bit rate switch.

Programmatic Use
Block Parameter: `BRSSwitch`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Add signal — Add CAN FD signal

Add a signal to the signal table.

Programmatic Use

None

Delete signal — Remove CAN FD signal

Remove the selected signal from the signal table.

Programmatic Use

None

Signals — Signals table
table

This table appears if you choose to specify signals manually or define signals by using a CANdb file.

If you are using a CANdb file, the data in the file populates this table and you cannot edit the fields. To edit signal information, switch to manually specified signals.

If you have selected to specify signals manually, create your signals in this table. Each signal that you create has these values:

Name

Specify a descriptive name for your signal. The Simulink block in your model displays this name. The default is `Signal [row number]`.

Start bit

Specify the start bit of the data. The start bit is the least significant bit counted from the start of the message data. For CAN the start bit must be an integer from 0 through 63, for CAN FD 0 through 511, within the number of bits in the message. (Note that message length is specified in bytes.)

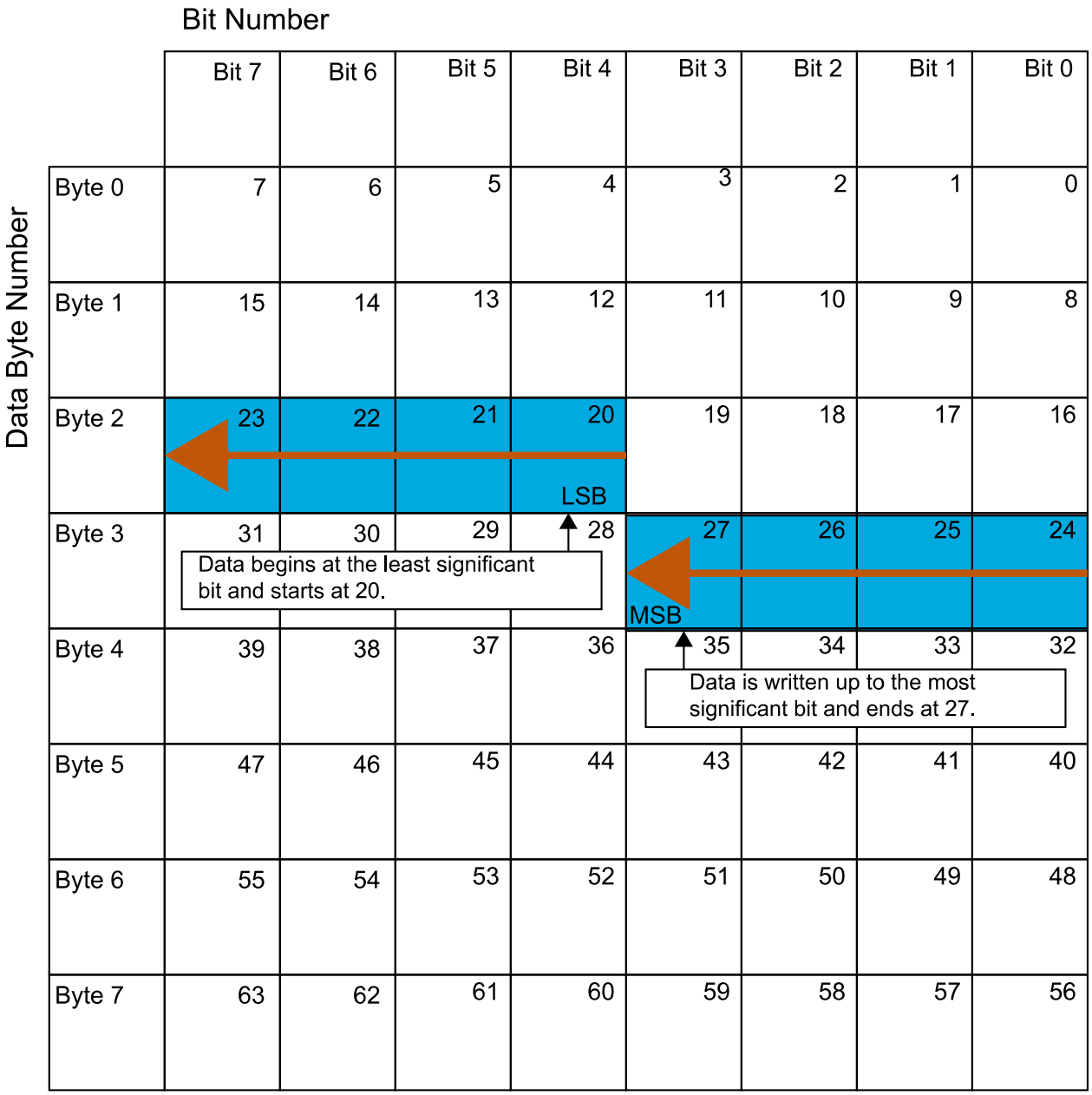
Length (bits)

Specify the number of bits the signal occupies in the message. The length must be an integer from 1 through 64. The sum of all the signal lengths in a message is limited to the number of bits in the message length; that is, all signals must cumulatively fit within the length of the message. (Note that message length is specified in bytes and signal length in bits.)

Byte order

Select either of these options:

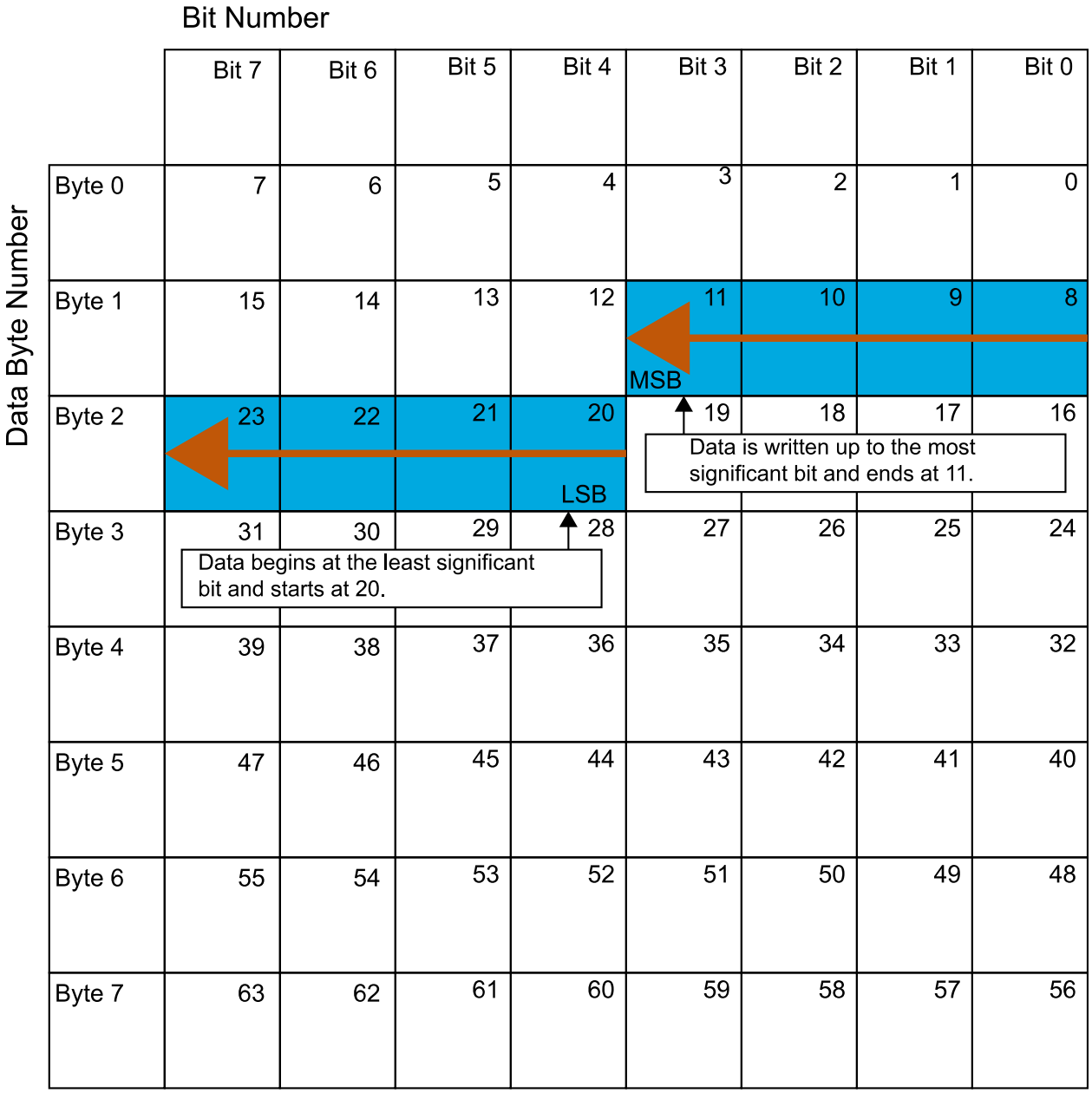
- LE: Where the byte order is in little-endian format (Intel®). In this format you count bits from the least significant bit, to the most significant bit. For example, if you pack one byte of data in little-endian format, with the start bit at 20, the data bit table resembles this figure.



Little-Endian Byte Order Counted from the Least-Significant Bit to the Highest

Address

- BE: Where byte order is in big-endian format (Motorola®). In this format you count bits from the least-significant bit to the most-significant bit. For example, if you pack one byte of data in big-endian format, with the start bit at 20, the data bit table resembles this figure.



Big-Endian Byte Order Counted from the Least Significant Bit to the Lowest Address

Data type

Specify how the signal interprets the data in the allocated bits. Choose from:

- `signed` (default)
- `unsigned`
- `single`
- `double`

  Note: If you have a `double` signal that does not align exactly to the message byte boundaries, to generate code with Embedded Coder you must check **Support long long** under **Device Details** in the **Hardware Implementation** pane of the Configuration Parameters dialog.

Multiplex type

Specify how the block packs the signals into the message at each time step:

- `Standard`: The signal is packed at each time step.
- `Multiplexor`: The `Multiplexor` signal, or the mode signal is packed. You can specify only one `Multiplexor` signal per message.
- `Multiplexed`: The signal is packed if the value of the `Multiplexor` signal (mode signal) at run time matches the configured **Multiplex value** of this signal.

For example, a message has four signals with these types and values.

Signal Name	Multiplex Type	Multiplex Value
Signal-A	Standard	Not applicable
Signal-B	Multiplexed	1
Signal-C	Multiplexed	0
Signal-D	Multiplexor	Not applicable

In this example:

- The block packs Signal-A (Standard signal) and Signal-D (Multiplexor signal) in every time step.
- If the value of Signal-D is 1 at a particular time step, then the block packs Signal-B along with Signal-A and Signal-D in that time step.
- If the value of Signal-D is 0 at a particular time step, then the block packs Signal-C along with Signal-A and Signal-D in that time step.
- If the value of Signal-D is not 1 or 0, the block does not pack either of the Multiplexed signals in that time step.

Multiplex value

This option is available only if you have selected the **Multiplex type** to be `Multiplexed`. The value you provide here must match the `Multiplexor` signal value at run time for the block to pack the `Multiplexed` signal. The **Multiplex value** must be a positive integer or zero.

Factor

Specify the **Factor** value to apply to convert the physical value (signal value) to the raw value packed in the message. See the **Data input as** parameter conversion formula to understand how physical values are converted to raw values packed into a message.

Offset

Specify the **Offset** value to apply to convert the physical value (signal value) to the raw value packed in the message. See the **Data input as** parameter conversion formula to understand how physical values are converted to raw values packed into a message.

Min, **Max**

Define a range of signal values. The default settings are `-Inf` (negative infinity) and `Inf`, respectively. For **CANdb specified signals**, these settings are read from the CAN database. For **manually specified signals**, you can specify the minimum and maximum physical value of the signal. By default, these settings do not clip signal values that exceed them.

Programmatic Use

Block Parameter: `SignalInfo`
Type: `string` | `character vector`

Version History

Introduced in R2018a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

Topics

“Design Your Model for Effective Acceleration”
“Composite Interfaces”

CAN FD Unpack

Unpack individual signals from CAN FD messages

Libraries:
Vehicle Network Toolbox / CAN FD Communication
Embedded Coder Support Package for Texas Instruments C2000 Processors / Target Communication
Simulink Real-Time / CAN / CAN-FD MSG blocks

Description

The CAN FD Unpack block unpacks a CAN FD message into signal data by using the specified output parameters at every time step. Data is output as individual signals.

To use this block, you also need a license for Simulink software.

The CAN FD Unpack block supports:

- Simulink Accelerator mode. You can speed up the execution of Simulink models. For more information, see “Design Your Model for Effective Acceleration”.

Tip

- To process every message coming through a channel, it is recommended that you use the CAN FD Unpack block in a function trigger subsystem. See “Using Triggered Subsystems”.
- To work with J1939 messages, use the blocks in the J1939 Communication block library instead of this block.

Ports

Input

Msg — CAN FD message input
`CAN_FD_MESSAGE_BUS`

This block has one input port, Msg. The CAN FD Unpack block takes the specified input CAN messages and unpacks their signal data to separate outputs.

The block supports the following input signal data types: single, double, int8, int16, int32, int64, uint8, uint16, uint32, uint64, and boolean. The block does not support fixed-point data types.

Code generation to deploy models to targets. Code generation is not supported if your signal information consists of signed or unsigned integers greater than 32 bits long.

Output

Data — CAN message output
single | double | int8 | int16 | int32 | int64 | uint32 | uint64 | boolean

The CAN FD Unpack block has one output port by default. The number of data output ports is dynamic and depends on the number of signals you specify for the block to output. For example, if your block has four signals, it has four output ports, labeled by signal name.

For manually or CANdb specified signals, the default output signal data type is double. To specify other types, use a Signal Specification block. This allows the block to support the following output signal data types: single, double, int8, int16, int32, int64, uint8, uint16, uint32, uint64, and boolean. The block does not support fixed-point types.

Additional output ports can be added by the options in the parameters **Output ports** pane.

Parameters

Data to output as — Select your data signal
`raw data` (default) | `manually specify signals` | `CANdb specified signals`

- `raw data`: Output data as a uint8 vector array. If you select this option, you specify only the message fields. The other signal parameter fields are unavailable. This option opens only one output port on your block.

  The conversion formula is:

  ```
  physical_value = raw_value * Factor + Offset
  ```

  where `raw_value` is the unpacked signal value and `physical_value` is the scaled signal value.
- `manually specified signals`: You can specify data signals. If you select this option, use the `Signals` table to create your signals message manually. The number of output ports on your block depends on the number of signals that you specify. For example, if you specify four signals, your block has four output ports.
- `CANdb specified signals`: You can specify a CAN database file that contains data signals. If you select this option, select a CANdb file. The number of output ports on your block depends on the number of signals specified in the CANdb file. For example, if the selected message in the CANdb file has four signals, your block has four output ports.

Programmatic Use
Block Parameter: `DataFormat`
Type: `string` | `character vector`
Values: `'raw data'` | `'manually specified signals'` | `'CANdb specified signals'`
Default: `'raw data'`

CANdb file — CAN database file
character vector

This option is available if you specify that your data is input via a CANdb file in the **Data to be output as** list. Click **Browse** to find the CANdb file on your system. The messages and signal definitions specified in the CANdb file populate the **Message** section of the dialog box. The signals specified in the CANdb file populate the **Signals** table. File names that contain non-alphanumeric characters such as equal signs, ampersands, and so forth, are not valid CAN database file names. You can use periods in your database name. Rename CAN database files with non-alphanumeric characters before you use them.

Programmatic Use
Block Parameter: `CANdbFile`

Type: `string` | `character vector`

Message list — Message list
array of character vectors

This option is available if you specify in the **Data to be output as** list that your data is to be output as a CANdb file and you select a CANdb file in the **CANdb file** field. You can select the message that you want to view. The **Signals** table then displays the details of the selected message.

Programmatic Use
Block Parameter: `MsgList`
Type: `string` | `character vector`

Name — Message name
`CAN Msg` (default) | character vector

Specify a name for your message. The default is `Msg`. This option is available if you choose to output raw data or manually specify signals.

Programmatic Use
Block Parameter: `MsgName`
Type: `string` | `character vector`

Identifier type — Identifier type
`Standard (11-bit identifier)` (default) | `Extended (29-bit identifier)`

Specify whether your message identifier is a `Standard` or an `Extended` type. The default is `Standard`. A standard identifier is an 11-bit identifier and an extended identifier is a 29-bit identifier. This option is available if you choose to output raw data or manually specify signals. For CANdb-specified signals, the **Identifier type** inherits the type from the database.

Programmatic Use
Block Parameter: `MsgIDType`
Type: `string` | `character vector`
Values: `'Standard (11-bit identifier)'` | `'Extended (29-bit identifier)'`
Default: `'Standard (11-bit identifier)'`

Identifier — Message identifier
`0` (default) | `0 .. 536870911`

Specify your message ID. This number must be an integer from 0 through 2047 for a standard identifier and from 0 through 536870911 for an extended identifier. If you specify `–1`, the block unpacks the messages that match the length specified for the message. You can also specify hexadecimal values using the `hex2dec` function. This option is available if you choose to output raw data or manually specify signals.

Programmatic Use
Block Parameter: `MsgIdentifier`
Type: `string` | `character vector`
Values: `'0'` to `'536870911'`

Length (bytes) — CAN message length
`8` (default) | `0 .. 8`

Specify the length of your message. For CAN messages the value can be 0-8 bytes; for CAN FD the value can be 0-8, 12, 16, 20, 24, 32, 48, or 64 bytes. If you are using `CANdb specified signals`

for your output data, the CANdb file defines the length of your message. This option is available if you choose to output raw data or manually specify signals.

Programmatic Use
Block Parameter: `MsgLength`
Type: `string` | `character vector`
Values: `'0'` to `'8'`, `'12'`, `'16'`, `'20'`, `'24'`, `'32'`, `'48'`, `'64'`
Default: `'8'`

Add signal — Add CAN signal

Add a signal to the signal table.

Programmatic Use

None

Delete signal — Remove CAN signal

Remove the selected signal from the signal table.

Programmatic Use

None

Signals — Signals table
table

If you choose to specify signals manually or define signals by using a CANdb file, this table appears.

If you are using a CANdb file, the data in the file populates this table and you cannot edit the fields. To edit signal information, switch to specified signals.

If you have selected to specify signals manually, create your signals manually in this table. Each signal that you create has these values:

Name

Specify a descriptive name for your signal. The Simulink block in your model displays this name. The default is `Signal [row number]`.

Start bit

Specify the start bit of the data. The start bit is the least significant bit counted from the start of the message data. For CAN the start bit must be an integer from 0 through 63, for CAN FD 0 through 511, within the number of bits in the message. (Note that message length is specified in bytes.)

Length (bits)

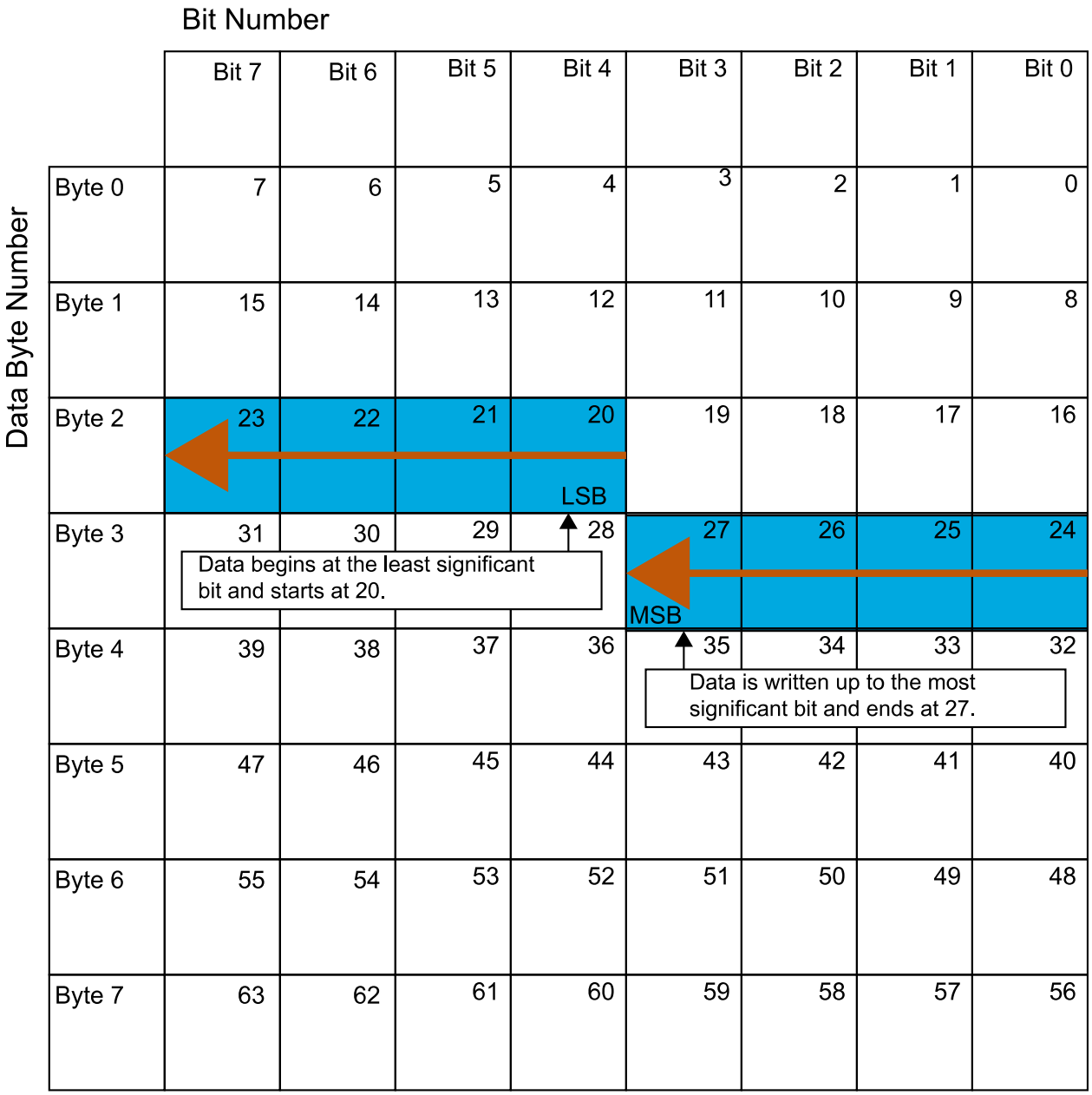
Specify the number of bits the signal occupies in the message. The length must be an integer from 1 through 64. The sum of all the signal lengths in a message is limited to the number of bits in the message length; that is, all signals must cumulatively fit within the length of the message. (Note that message length is specified in bytes and signal length in bits.)

Byte order

Select either of the following options:

- LE: Where the byte order is in little-endian format (Intel). In this format you count bits from the least-significant bit to the most-significant bit and proceeding to the next higher byte as

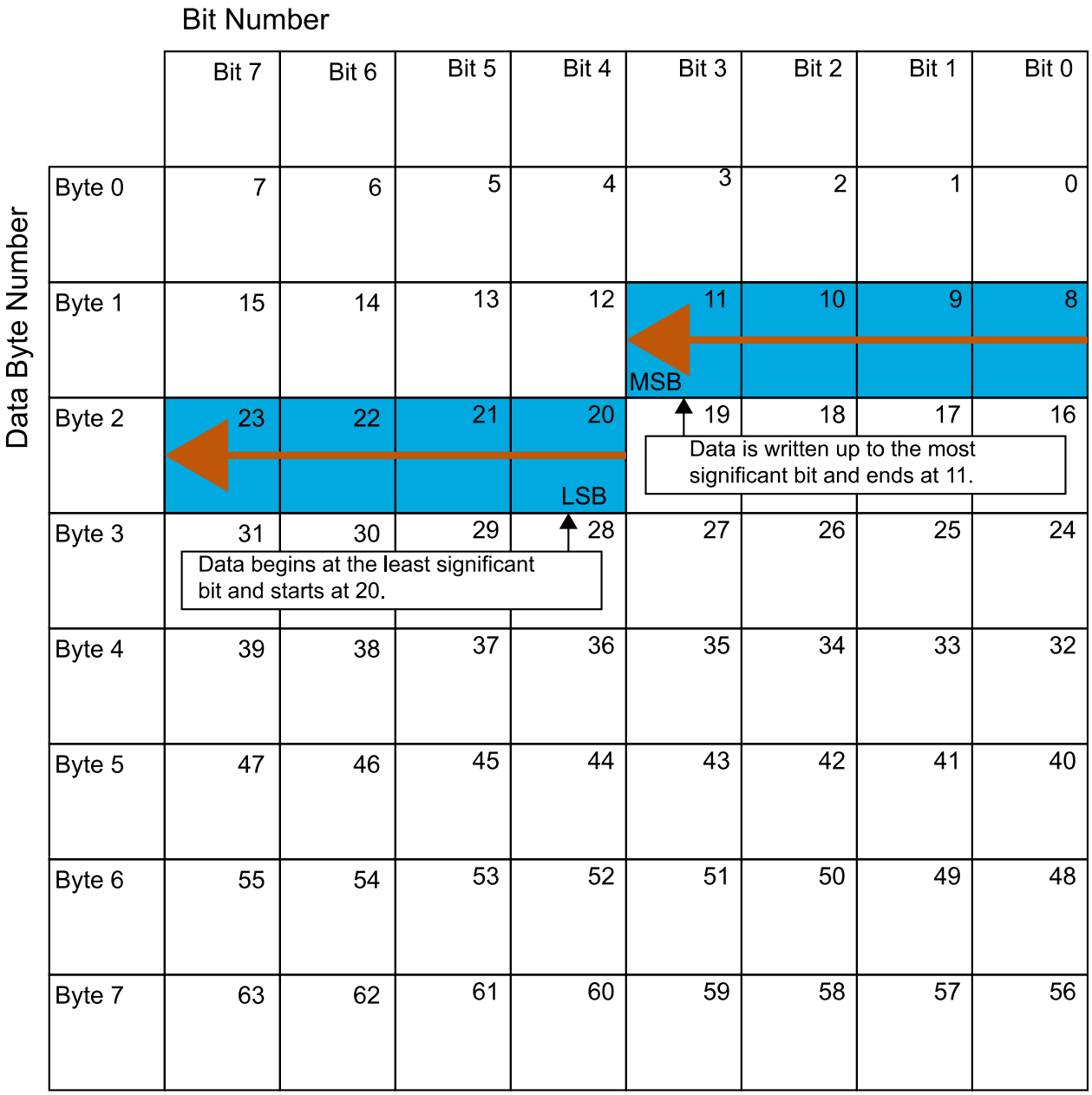
you cross a byte boundary. For example, if you pack one byte of data in little-endian format, with the start bit at 20, the data bit table resembles this figure.



Little-Endian Byte Order Counted from the Least Significant Bit to the Highest

Address

- BE: Where the byte order is in big-endian format (Motorola). In this format you count bits from the least-significant bit to the most-significant bit and proceeding to the next lower byte as you cross a byte boundary. For example, if you pack one byte of data in big-endian format, with the start bit at 20, the data bit table resembles this figure.



Big-Endian Byte Order Counted from the Least Significant Bit to the Lowest Address

Data type

Specify how the signal interprets the data in the allocated bits. Choose from:

- `signed` (default)
- `unsigned`
- `single`
- `double`

  Note: If you have a `double` signal that does not align exactly to the message byte boundaries, to generate code with Embedded Coder you must check **Support long long** under **Device Details** in the **Hardware Implementation** pane of the Configuration Parameters dialog.

Multiplex type

Specify how the block unpacks the signals from the message at each time step:

- `Standard`: The signal is unpacked at each time step.
- `Multiplexor`: The `Multiplexor` signal, or the mode signal is unpacked. You can specify only one `Multiplexor` signal per message.
- `Multiplexed`: The signal is unpacked if the value of the `Multiplexor` signal (mode signal) at run time matches the configured **Multiplex value** of this signal.

For example, a message has four signals with these values.

Signal Name	Multiplex Type	Multiplex Value
Signal-A	Standard	Not applicable
Signal-B	Multiplexed	1
Signal-C	Multiplexed	0
Signal-D	Multiplexor	Not applicable

In this example:

- The block unpacks Signal-A (Standard signal) and Signal-D (Multiplexor signal) in every time step.
- If the value of Signal-D is 1 at a particular time step, then the block unpacks Signal-B along with Signal-A and Signal-D in that time step.
- If the value of Signal-D is 0 at a particular time step, then the block unpacks Signal-C along with Signal-A and Signal-D in that time step.
- If the value of Signal-D is not 1 or 0, the block does not unpack either of the Multiplexed signals in that time step.

Multiplex value

This option is available only if you have selected the **Multiplex type** to be `Multiplexed`. The value you provide here must match the `Multiplexor` signal value at run time for the block to unpack the `Multiplexed` signal. The **Multiplex value** must be a positive integer or zero.

Factor

Specify the **Factor** value applied to convert the unpacked raw value to the physical value (signal value). For more information, see the **Data input as** parameter conversion formula.

Offset

Specify the **Offset** value applied to convert the physical value (signal value) to the unpacked raw value. For more information, see the **Data input as** parameter conversion formula.

Min, **Max**

Define a range of raw signal values. The default settings are `-Inf` (negative infinity) and `Inf`, respectively. For **CANdb specified signals**, these settings are read from the CAN database. For **manually specified signals**, you can specify the minimum and maximum physical value of the signal. By default, these settings do not clip signal values that exceed them.

Programmatic Use

Block Parameter: `SignalInfo`
Type: `string` | `character vector`

Output identifier — Add CAN ID output port
`off` (default) | `on`

Select this option to output a CAN message identifier. The data type of this port is `uint32`.

Programmatic Use

Block Parameter: `IDPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output timestamp — Add Timestamp output port
`off` (default) | `on`

Select this option to output the message timestamp. This value indicates when the message was received, measured as the number of seconds elapsed since the model simulation began. This option adds a new output port to the block. The data type of this port is `double`.

Programmatic Use

Block Parameter: `TimestampPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output error — Add Error output port
`off` (default) | `on`

Select this option to output the message error status. This option adds a new output port to the block. An output value of `1` on this port indicates that the incoming message is an error frame. If the output value is `0`, there is no error. The data type of this port is `uint8`.

Programmatic Use

Block Parameter: `ErrorPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output remote — Add Remote output port
`off` (default) | `on`

Select this option to output the message remote frame status. This option adds a new output port to the block. The data type of this port is `uint8`.

Programmatic Use
Block Parameter: `RemotePort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output length — Add Length output port
`off` (default) | `on`

Select this option to output the length of the message in bytes. This option adds a new output port to the block. The data type of this port is `uint8`.

Programmatic Use
Block Parameter: `LengthPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output status — Add Status output port
`off` (default) | `on`

Select this option to output the message received status. The status is `1` if the block receives new message and `0` if it does not. This option adds a new output port to the block. The data type of this port is `uint8`.

Programmatic Use
Block Parameter: `StatusPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output Bit Rate Switch (BRS) — Add BRS output port
`off` (default) | `on`

(Disabled for CAN protocol.) Select this option to output the message bit rate switch. This option adds a new output port to the block, which indicates if the CAN FD message bit rate switch is set. The data type of this port is `boolean`, indicating whether the bit rate for the data phase of the message is faster (`true`) or the same (`false`) as the bit rate of the arbitration phase.

For more information on BRS, see CAN FD - Some Protocol Details.

Programmatic Use
Block Parameter: `BRSPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output Error Status Indicator (ESI) — Add ESI output port
`off` (default) | `on`

(Disabled for CAN protocol.) Select this option to output the message error status. This option adds a new output port to the block. The data type of this port is `boolean`, indicating if the CAN FD message error state indicator flag is set.

For more information on ESI, see CAN FD - Some Protocol Details.

Programmatic Use
Block Parameter: `ESIPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output Data Length Code (DLC) — Add DLC output port
`off` (default) | `on`

(Disabled for CAN protocol.) Select this option to output the message data length. This option adds a new output port to the block. The data type of this port is `double`.

Programmatic Use
Block Parameter: `DLCPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Version History

Introduced in R2018a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Blocks

Topics

“Design Your Model for Effective Acceleration”
“Composite Interfaces”

CAN Pack

Pack individual signals into CAN message

Libraries:
Vehicle Network Toolbox / CAN Communication
Embedded Coder / Embedded Targets / Host Communication
Embedded Coder Support Package for Texas Instruments C2000 Processors / Target Communication
Simulink Real-Time / CAN / CAN MSG blocks

Description

The CAN Pack block loads signal data into a CAN message at specified intervals during the simulation.

To use this block, you must have a license for Simulink software.

The CAN Pack block supports:

- Simulink Accelerator rapid accelerator mode. You can speed up the execution of Simulink models.
- Model referencing. Your model can include other Simulink models as modular components.

For more information, see “Design Your Model for Effective Acceleration”.

Tip

- This block can be used to encode the signals of J1939 parameter groups up to 8 bytes. However, to work with J1939 messages, it is preferable to use the blocks in the J1939 Communication block library instead of this block.

Ports

Input

Data — CAN message signal input
single | double | int8 | int16 | int32 | int64 | uint32 | uint64 | boolean

The CAN Pack block has one input port by default. The number of block inputs is dynamic and depends on the number of signals you specify for the block. For example, if your message has four signals, the block can have four input ports.

The block supports the following input signal data types: single, double, int8, int16, int32, int64, uint8, uint16, uint32, uint64, and boolean. The block does not support fixed-point data types.

Code generation to deploy models to targets. If your signal information consists of signed or unsigned integers greater than 32 bits long, code generation is not supported.

Output

CAN Msg — CAN message output
`CAN_MESSAGE` | `CAN_MESSAGE_BUS`

This block has one output port, CAN Msg. The CAN Pack block takes the specified input signals and packs them into a CAN message. The output data type is determined by the **Output as bus** parameter setting.

Parameters

Data input as — Select your data signal
`raw data` (default) | `manually specified signals` | `CANdb specified signals`

- `raw data`: Input data as a uint8 vector array. If you select this option, you only specify the message fields. all other signal parameter fields are unavailable. This option opens only one input port on your block.

  The conversion formula is:

  ```
  raw_value = (physical_value - Offset) / Factor
  ```

  where `physical_value` is the original value of the signal and `raw_value` is the packed signal value.
- `manually specified signals`: Allows you to specify data signal definitions. If you select this option, use the **Signals** table to create your signals. The number of block inputs depends on the number of signals you specify.
- `CANdb specified signals`: Allows you to specify a CAN database file that contains message and signal definitions. If you select this option, select a CANdb file. The number of block inputs depends on the number of signals specified in the CANdb file for the selected message.

Programmatic Use

Block Parameter: `DataFormat`
Type: `string` | `character vector`
Values: `'raw data'` | `'manually specified signals'` | `'CANdb specified signals'`
Default: `'raw data'`

CANdb file — CAN database file
character vector

This option is available if you specify that your data is input through a CANdb file in the **Data is input as** list. Click **Browse** to find the CANdb file on your system. The message list specified in the CANdb file populates the **Message** section of the dialog box. The CANdb file also populates the **Signals** table for the selected message.

File names that contain non-alphanumeric characters such as equal signs, ampersands, and so on are not valid CAN database file names. You can use periods in your database name. Before you use the CAN database files, rename them with non-alphanumeric characters.

Programmatic Use

Block Parameter: `CANdbFile`
Type: `string` | `character vector`

Message list — CAN message list
array of character vectors

This option is available if you specify that your data is input through a CANdb file in the **Data is input as** field and you select a CANdb file in the **CANdb file** field. Select the message to display signal details in the **Signals** table.

Programmatic Use
Block Parameter: `MsgList`
Type: `string` | `character vector`

Name — CAN message name
`CAN Msg` (default) | character vector

Specify a name for your CAN message. The default is `CAN Msg`. This option is available if you choose to input raw data or manually specify signals. This option is not available if you choose to use signals from a CANdb file.

Programmatic Use
Block Parameter: `MsgName`
Type: `string` | `character vector`

Identifier type — CAN identifier type
`Standard (11-bit identifier)` (default) | `Extended (29-bit identifier)`

Specify whether your CAN message identifier is a `Standard` or an `Extended` type. The default is `Standard`. A standard identifier is an 11-bit identifier and an extended identifier is a 29-bit identifier. This option is available if you choose to input raw data or manually specify signals. For `CANdb specified signals`, the **Identifier type** inherits the type from the database.

Programmatic Use
Block Parameter: `MsgIDType`
Type: `string` | `character vector`
Values: `'Standard (11-bit identifier)'` | `'Extended (29-bit identifier)'`
Default: `'Standard (11-bit identifier)'`

CAN Identifier — CAN message ID
`0` (default) | `0` to `536870911`

Specify your CAN message ID. This number must be a positive integer from 0 through 2047 for a standard identifier and from 0 through 536870911 for an extended identifier. You can also specify hexadecimal values by using the `hex2dec` function. This option is available if you choose to input raw data or manually specify signals.

Programmatic Use
Block Parameter: `MsgIdentifier`
Type: `string` | `character vector`
Values: `'0'` to `'536870911'`

Length (bytes) — CAN message length
`8` (default) | `0` to `8`

Specify the length of your CAN message from 0 to 8 bytes. If you are using `CANdb specified signals` for your data input, the CANdb file defines the length of your message. If not, this field defaults to `8`. This option is available if you choose to input raw data or manually specify signals.

Programmatic Use
Block Parameter: `MsgLength`
Type: `string` | `character vector`
Values: `'0'` to `'8'`
Default: `'8'`

Remote frame — CAN message as remote frame
`off` (default) | `on`

Specify the CAN message as a remote frame.

Programmatic Use
Block Parameter: `Remote`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output as bus — CAN message as bus
`off` (default) | `on`

Select this option for the block to output CAN messages as a Simulink bus signal. For more information on Simulink bus objects, see “Composite Interfaces”.

Programmatic Use
Block Parameter: `BusOutput`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Add signal — Add CAN signal

Add a new signal to the signal table.

Programmatic Use

None

Delete signal — Remove CAN signal

Remove the selected signal from the signal table.

Programmatic Use

None

Signals — Signals table
table

This table appears if you choose to specify signals manually or define signals by using a CANdb file.

If you are using a CANdb file, the data in the file populates this table and you cannot edit the fields. To edit signal information, switch to manually specified signals.

If you have selected to specify signals manually, create your signals in this table. Each signal that you create has these values:

Name

Specify a descriptive name for your signal. The Simulink block in your model displays this name. The default is `Signal [row number]`.

Start bit

Specify the start bit of the data. The start bit is the least significant bit counted from the start of the message data. The start bit must be an integer from 0 through 63.

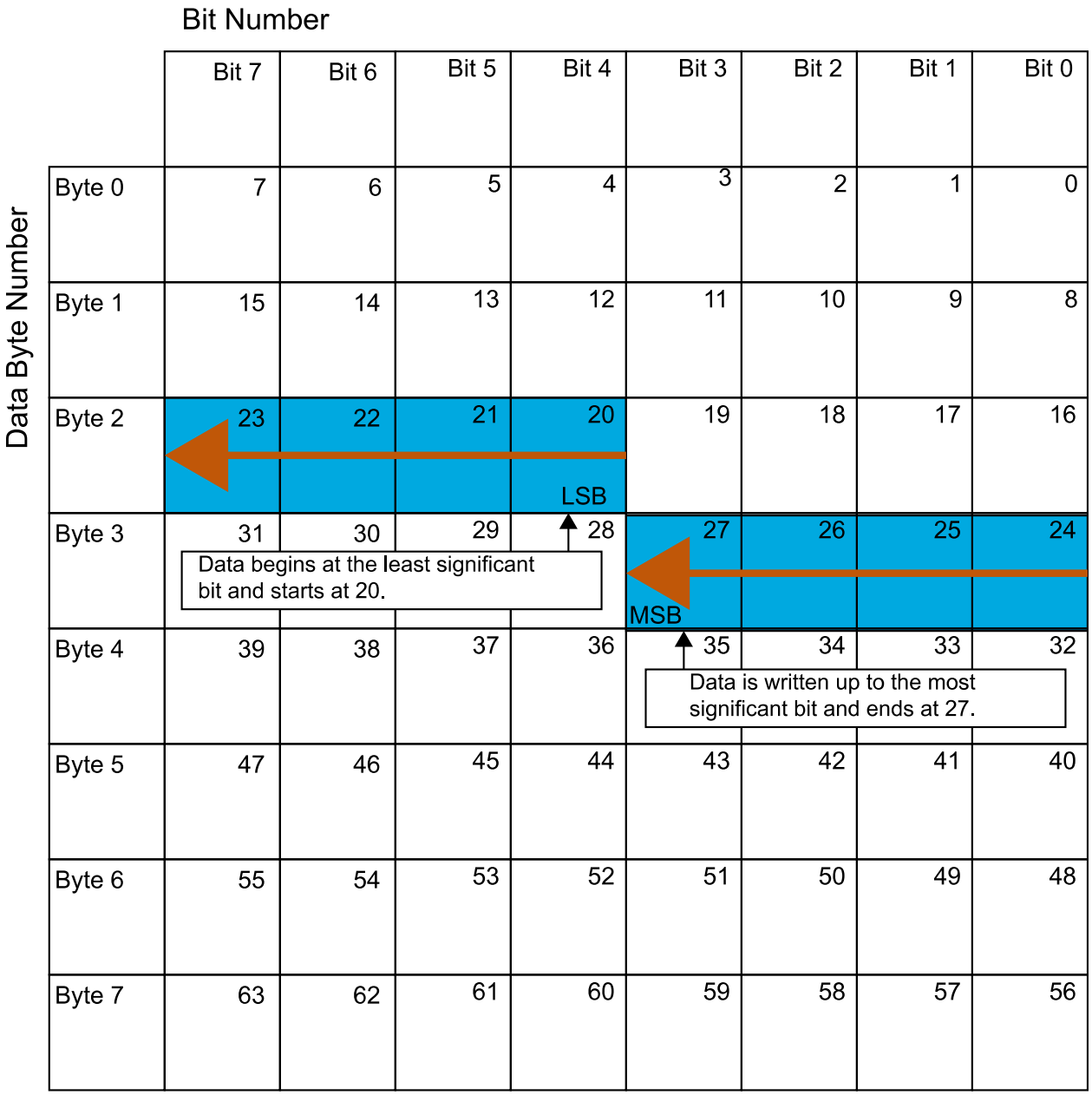
Length (bits)

Specify the number of bits the signal occupies in the message. The length must be an integer from 1 through 64.

Byte order

Select either of these options:

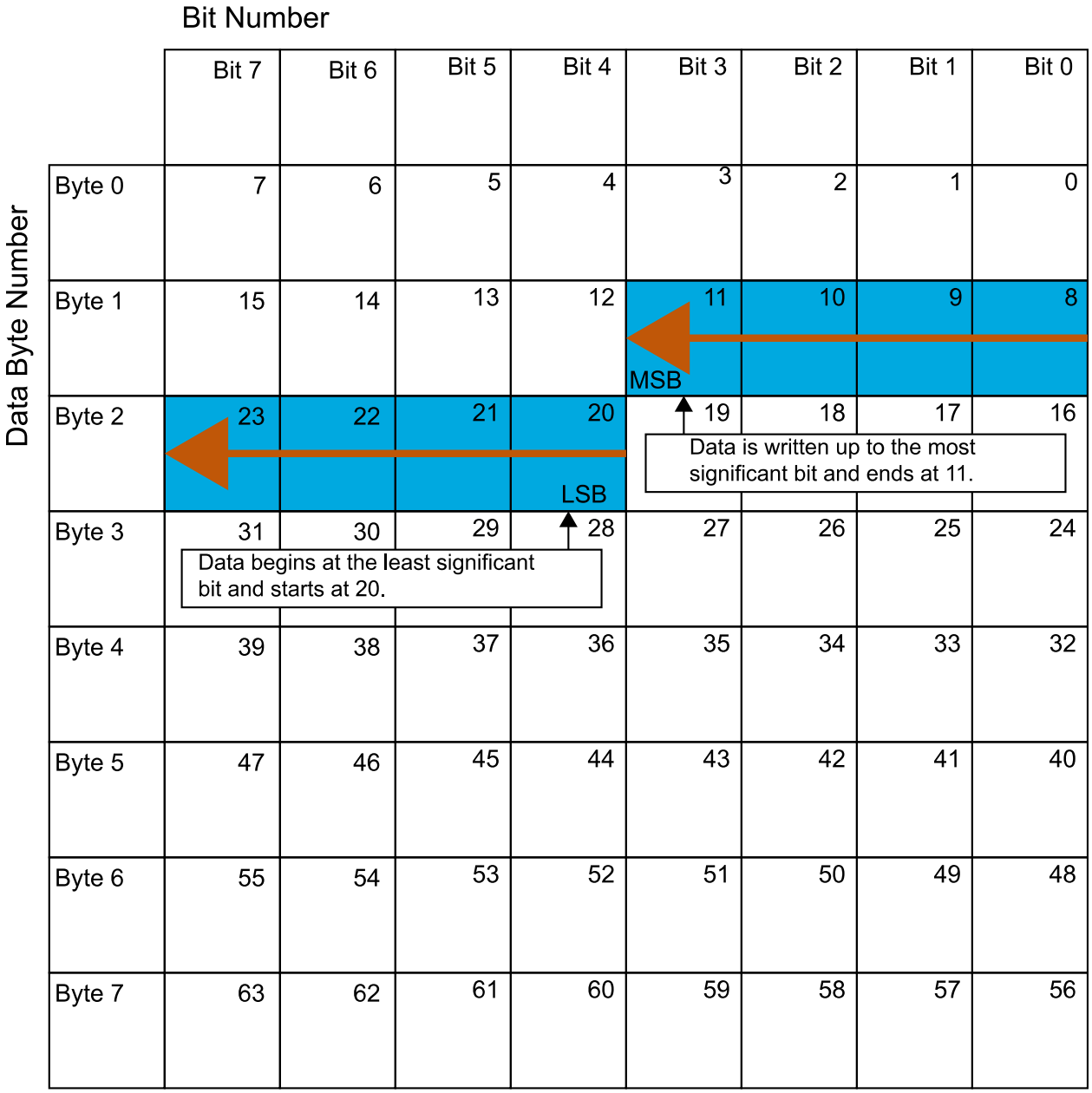
- LE: Where the byte order is in little-endian format (Intel). In this format you count bits from the least significant bit, to the most significant bit. For example, if you pack one byte of data in little-endian format, with the start bit at 20, the data bit table resembles this figure.



Little-Endian Byte Order Counted from the Least-Significant Bit to the Highest

Address

- BE: Where byte order is in big-endian format (Motorola). In this format you count bits from the least-significant bit to the most-significant bit. For example, if you pack one byte of data in big-endian format, with the start bit at 20, the data bit table resembles this figure.



Big-Endian Byte Order Counted from the Least Significant Bit to the Lowest Address

Data type

Specify how the signal interprets the data in the allocated bits. Choose from:

- `signed` (default)
- `unsigned`
- `single`
- `double`

Multiplex type

Specify how the block packs the signals into the CAN message at each time step:

- `Standard`: The signal is packed at each time step.
- `Multiplexor`: The `Multiplexor` signal, or the mode signal is packed. You can specify only one `Multiplexor` signal per message.
- `Multiplexed`: The signal is packed if the value of the `Multiplexor` signal (mode signal) at run time matches the configured **Multiplex value** of this signal.

For example, a message has these signals with the following types and values.

Signal Name	Multiplex Type	Multiplex Value
Signal-A	Standard	Not applicable
Signal-B	Multiplexed	1
Signal-C	Multiplexed	0
Signal-D	Multiplexor	Not applicable

In this example:

- The block packs Signal-A (Standard signal) and Signal-D (Multiplexor signal) in every time step.
- If the value of Signal-D is 1 at a particular time step, then the block packs Signal-B along with Signal-A and Signal-D in that time step.
- If the value of Signal-D is 0 at a particular time step, then the block packs Signal-C along with Signal-A and Signal-D in that time step.
- If the value of Signal-D is not 1 or 0, the block does not pack either of the Multiplexed signals in that time step.

Multiplex value

This option is available only if you have selected the **Multiplex type** to be `Multiplexed`. The value you provide must match the `Multiplexor` signal value at run time for the block to pack the `Multiplexed` signal. The **Multiplex value** must be a positive integer or zero.

Factor

Specify the **Factor** value to apply to convert the physical value (signal value) to the raw value packed in the message. For more information, see the **Data input as** parameter conversion formula.

Offset

Specify the **Offset** value to apply to convert the physical value (signal value) to the raw value packed in the message. For more information, see the **Data input as** parameter conversion formula.

Min, **Max**

Define a range of signal values. The default settings are `-Inf` (negative infinity) and `Inf`, respectively. For **CANdb specified signals**, these settings are read from the CAN database. For **manually specified signals**, you can specify the minimum and maximum physical value of the signal. By default, these settings do not clip signal values that exceed the settings.

Programmatic Use

Block Parameter: `SignalInfo`
Type: `string` | `character vector`

Version History

Introduced in R2009a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

See Also

Topics

“Design Your Model for Effective Acceleration”

CAN Unpack

Unpack individual signals from CAN messages

Libraries:
Vehicle Network Toolbox / CAN Communication
Embedded Coder / Embedded Targets / Host Communication
Embedded Coder Support Package for Texas Instruments C2000 Processors / Target Communication
Simulink Real-Time / CAN / CAN MSG blocks

Description

The CAN Unpack block unpacks a CAN message into signal data using the specified output parameters at every time step. Data is output as individual signals.

To use this block, you also need a license for Simulink software.

The CAN Unpack block supports:

- The use of Simulink Accelerator Rapid Accelerator mode. Using this feature, you can speed up the execution of Simulink models.
- The use of model referencing. Using this feature, your model can include other Simulink models as modular components.

For more information on these features, see “Design Your Model for Effective Acceleration”.

Tip

- To process every message coming through a channel, it is recommended that you use the CAN Unpack block in a function trigger subsystem. See “Using Triggered Subsystems”.
- This block can be used to decode the signals of J1939 parameter groups up to 8 bytes. However, to work with J1939 messages, it is preferable to use the blocks in the J1939 Communication block library instead of this block.

Ports

Input

CAN Msg — CAN message input
`CAN_MESSAGE` | `CAN_MESSAGE_BUS`

This block has one input port, **CAN Msg**. The block takes the specified input CAN messages and unpacks their signal data to separate outputs.

The block supports the following signal data types: single, double, int8, int16, int32, int64, uint8, uint16, uint32, uint64, and boolean. The block does not support fixed-point data types.

Code generation to deploy models to targets. Code generation is not supported if your signal information consists of signed or unsigned integers greater than 32 bits long.

Output

Data — CAN signal output
single | double | int8 | int16 | int32 | int64 | uint32 | uint64 | boolean

The block has one output port by default. The number of output ports is dynamic and depends on the number of signals that you specify for the block to output. For example, if your message has four signals, the block can have four output ports.

For signals specified manually or by a CANdb, the default output data type for CAN signals is double. To specify other types, use a Signal Specification block. This allows the block to support the following output signal data types: single, double, int8, int16, int32, int64, uint8, uint16, uint32, uint64, and boolean. The block does not support fixed-point types.

Additional output ports can be added by selecting the options in the parameters **Output ports** pane. For more information, see the parameters `Output identifier`, `Output timestamp`, `Output error`, `Output remote`, `Output length`, and `Output status`.

Parameters

Data to output as — Select your data signal
`raw data` (default) | `manually specify signals` | `CANdb specified signals`

- `raw data`: Output data as a uint8 vector array. If you select this option, you specify only the message fields. The other signal parameter fields are unavailable. This option opens only one output port on your block.

  The conversion formula is:

  ```
  physical_value = raw_value * Factor + Offset
  ```

  where `raw_value` is the unpacked signal value and `physical_value` is the scaled signal value.
- `manually specified signals`: You can specify data signals. If you select this option, use the `Signals` table to create your signals message manually. The number of output ports on your block depends on the number of signals that you specify. For example, if you specify four signals, your block has four output ports.
- `CANdb specified signals`: You can specify a CAN database file that contains data signals. If you select this option, select a CANdb file. The number of output ports on your block depends on the number of signals specified in the CANdb file. For example, if the selected message in the CANdb file has four signals, your block has four output ports.

Programmatic Use
Block Parameter: `DataFormat`
Type: `string` | `character vector`
Values: `'raw data'` | `'manually specified signals'` | `'CANdb specified signals'`
Default: `'raw data'`

CANdb file — CAN database file
character vector

This option is available if you specify that your data is input via a CANdb file in the **Data to be output as** list. Click **Browse** to find the CANdb file on your system. The messages and signal

definitions specified in the CANdb file populate the **Message** section of the dialog box. The signals specified in the CANdb file populate **Signals** table. File names that contain non-alphanumeric characters such as equal signs, ampersands, and so forth are not valid CAN database file names. You can use periods in your database name. Rename CAN database files with non-alphanumeric characters before you use them.

Programmatic Use
Block Parameter: `CANdbFile`
Type: `string` | `character vector`

Message list — CAN message list
array of character vectors

This option is available if you specify in the **Data to be output as** list that your data is to be output as a CANdb file and you select a CANdb file in the **CANdb file** field. You can select the message that you want to view. The **Signals** table then displays the details of the selected message.

Programmatic Use
Block Parameter: `MsgList`
Type: `string` | `character vector`

Name — CAN message name
`CAN Msg` (default) | character vector

Specify a name for your CAN message. The default is `CAN Msg`. This option is available if you choose to output raw data or manually specify signals.

Programmatic Use
Block Parameter: `MsgName`
Type: `string` | `character vector`

Identifier type — CAN identifier type
`Standard (11-bit identifier)` (default) | `Extended (29-bit identifier)`

Specify whether your CAN message identifier is a `Standard` or an `Extended` type. The default is `Standard`. A standard identifier is an 11-bit identifier and an extended identifier is a 29-bit identifier. This option is available if you choose to output raw data or manually specify signals. For CANdb-specified signals, the **Identifier type** inherits the type from the database.

Programmatic Use
Block Parameter: `MsgIDType`
Type: `string` | `character vector`
Values: `'Standard (11-bit identifier)'` | `'Extended (29-bit identifier)'`
Default: `'Standard (11-bit identifier)'`

CAN Identifier — CAN message identifier
`0` (default) | `0` to `536870911`

Specify your CAN message ID. This number must be an integer from 0 through 2047 for a standard identifier and from 0 through 536870911 for an extended identifier. If you specify `-1`, the block unpacks the messages that match the length specified for the message. You can also specify hexadecimal values by using the `hex2dec` function. This option is available if you choose to output raw data or manually specify signals.

Programmatic Use
Block Parameter: `MsgIdentifier`

Type: `string` | `character vector`
Values: `'0'` to `'536870911'`

Length (bytes) — CAN message length
`8` (default) | `0` to `8`

Specify the length of your CAN message from 0 to 8 bytes. If you are using `CANdb specified signals` for your output data, the CANdb file defines the length of your message. Otherwise, this field defaults to `8`. This option is available if you choose to output raw data or manually specify signals.

Programmatic Use

Block Parameter: `MsgLength`
Type: `string` | `character vector`
Values: `'0'` to `'8'`
Default: `'8'`

Add signal — Add CAN signal

Add a signal to the signal table.

Programmatic Use

None

Delete signal — Remove CAN signal

Remove the selected signal from the signal table.

Programmatic Use

None

Signals — Signals table
table

If you choose to specify signals manually or define signals by using a CANdb file, this table appears.

If you are using a CANdb file, the data in the file populates this table and you cannot edit the fields. To edit signal information, switch to specified signals.

If you have selected to specify signals manually, create your signals manually in this table. Each signal that you create has these values:

Name

Specify a descriptive name for your signal. The Simulink block in your model displays this name. The default is `Signal [row number]`.

Start bit

Specify the start bit of the data. The start bit is the least significant bit counted from the start of the message. The start bit must be an integer from 0 through 63.

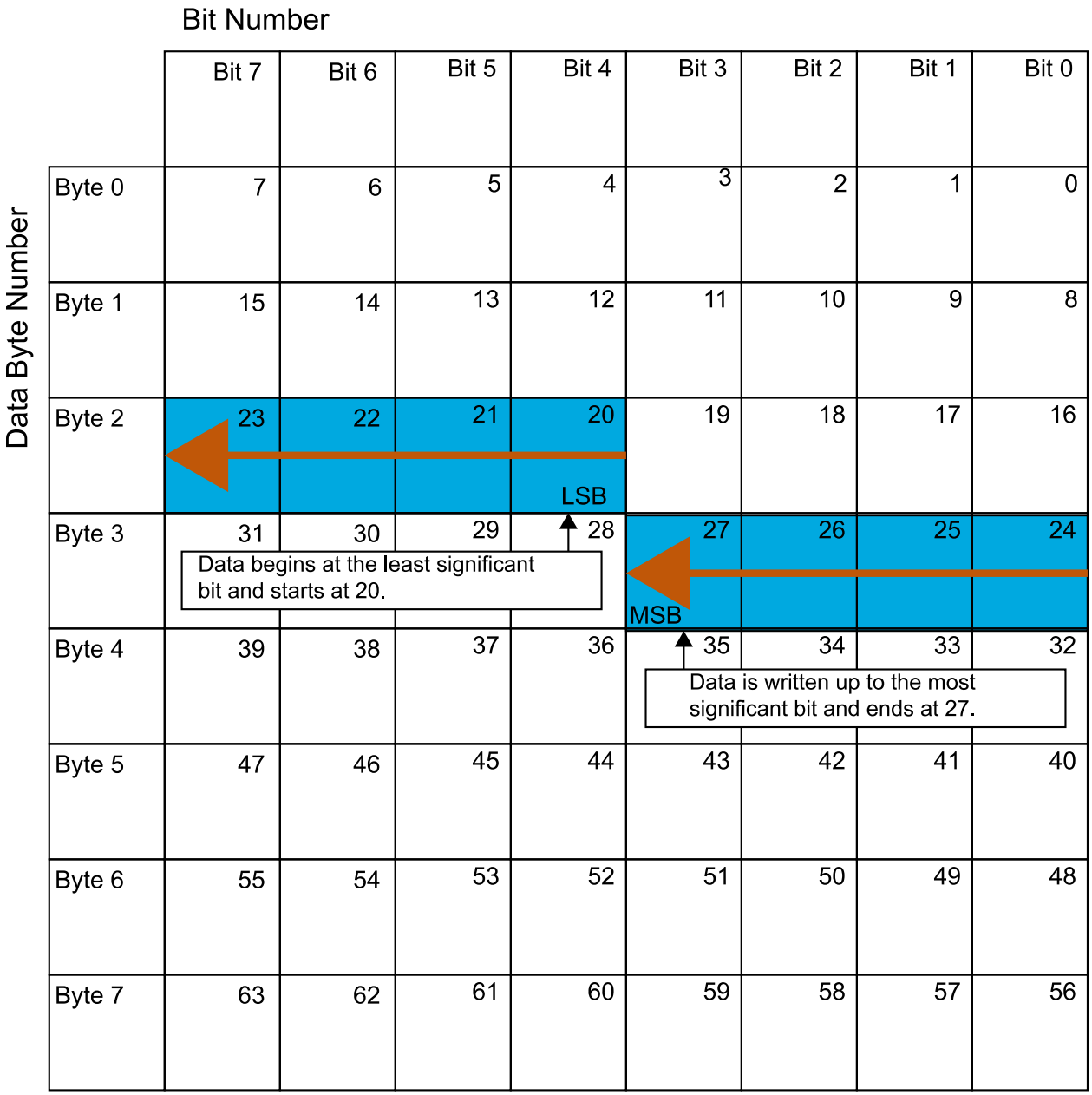
Length (bits)

Specify the number of bits the signal occupies in the message. The length must be an integer from 1 through 64.

Byte order

Select either of these options:

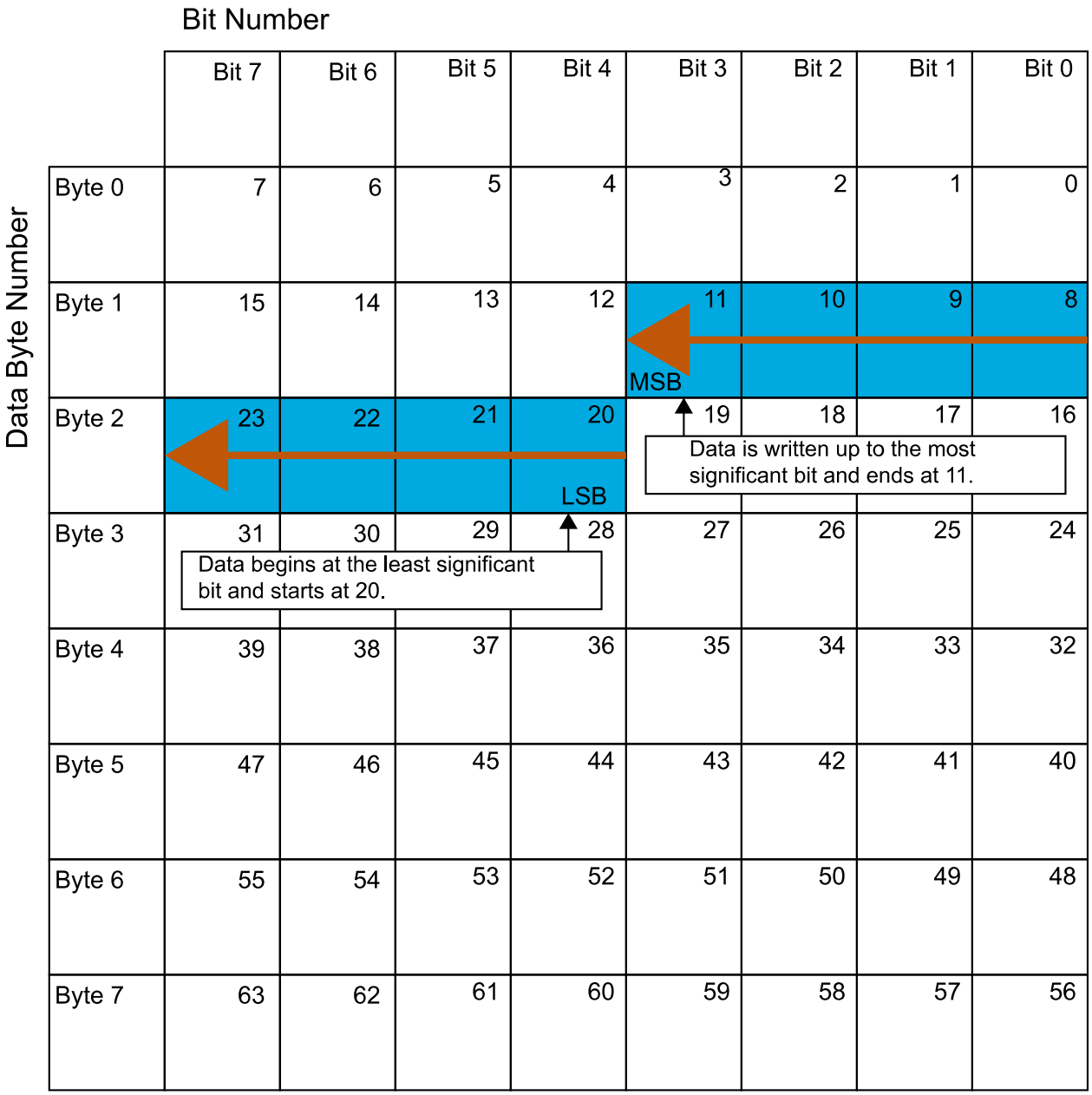
- LE: Where the byte order is in little-endian format (Intel). In this format you count bits from the least-significant bit to the most-significant bit. For example, if you pack one byte of data in little-endian format, with the start bit at 20, the data bit table resembles this figure.



Little-Endian Byte Order Counted from the Least Significant Bit to the Highest

Address

- BE: Where the byte order is in big-endian format (Motorola). In this format you count bits from the least-significant bit to the most-significant bit. For example, if you pack one byte of data in big-endian format, with the start bit at 20, the data bit table resembles this figure.



Big-Endian Byte Order Counted from the Least Significant Bit to the Lowest Address

Data type

Specify how the signal interprets the data in the allocated bits. Choose from:

- `signed` (default)
- `unsigned`
- `single`
- `double`

Multiplex type

Specify how the block unpacks the signals from the CAN message at each time step:

- `Standard`: The signal is unpacked at each time step.
- `Multiplexor`: The `Multiplexor` signal or the mode signal is unpacked. You can specify only one `Multiplexor` signal per message.
- `Multiplexed`: The signal is unpacked if the value of the `Multiplexor` signal (mode signal) at run time matches the configured **Multiplex value** of this signal.

For example, a message has four signals with these values.

Signal Name	Multiplex Type	Multiplex Value
Signal-A	Standard	Not applicable
Signal-B	Multiplexed	1
Signal-C	Multiplexed	0
Signal-D	Multiplexor	Not applicable

In this example:

- The block unpacks Signal-A (Standard signal) and Signal-D (Multiplexor signal) in every time step.
- If the value of Signal-D is 1 at a particular time step, then the block unpacks Signal-B along with Signal-A and Signal-D in that time step.
- If the value of Signal-D is 0 at a particular time step, then the block unpacks Signal-C along with Signal-A and Signal-D in that time step.
- If the value of Signal-D is not 1 or 0, the block does not unpack either of the Multiplexed signals in that time step.

Multiplex value

This option is available only if you have selected the **Multiplex type** to be `Multiplexed`. The value you provide must match the `Multiplexor` signal value at run time for the block to unpack the `Multiplexed` signal. The **Multiplex value** must be a positive integer or zero.

Factor

Specify the **Factor** value applied to convert the unpacked raw value to the physical value (signal value). For more information, see the **Data input as** parameter conversion formula.

Offset

Specify the **Offset** value applied to convert the physical value (signal value) to the unpacked raw value. For more information, see the **Data input as** parameter conversion formula.

Min, **Max**

Define a range of raw signal values. The default settings are `-Inf` (negative infinity) and `Inf`, respectively. For **CANdb specified signals**, these settings are read from the CAN database. For **manually specified signals**, you can specify the minimum and maximum physical value of the signal. By default, these settings do not clip signal values that exceed them.

Programmatic Use
Block Parameter: `SignalInfo`
Type: `string` | `character vector`

Output identifier — Add CAN ID output port
`off` (default)

Select this option to output a CAN message identifier. The data type of this port is `uint32`.

Programmatic Use
Block Parameter: `IDPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output timestamp — Add Timestamp output port
`off` (default) | `on`

Select this option to output the message timestamp. This value indicates when the message was received, measured as the number of seconds elapsed since the model simulation began. This option adds a new output port to the block. The data type of this port is `double`.

Programmatic Use
Block Parameter: `TimestampPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output error — Add Error output port
`off` (default) | `on`

Select this option to output the message error status. This option adds a new output port to the block. An output value of `1` on this port indicates that the incoming message is an error frame. If the output value is `0`, there is no error. The data type of this port is `uint8`.

Programmatic Use
Block Parameter: `ErrorPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output remote — Add Remote output port
`off` (default) | `on`

Select this option to output the message remote frame status. This option adds a new output port to the block. The data type of this port is `uint8`.

Programmatic Use
Block Parameter: `RemotePort`

Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output length — Add Length output port
`off` (default) | `on`

Select this option to output the length of the message in bytes. This option adds a new output port to the block. The data type of this port is `uint8`.

Programmatic Use
Block Parameter: `LengthPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Output status — Add Status output port
`off` (default) | `on`

Select this option to output the message received status. The status is `1` if the block receives a new message and `0` if it does not. This option adds a new output port to the block. The data type of this port is `uint8`.

Programmatic Use
Block Parameter: `StatusPort`
Type: `string` | `character vector`
Values: `'off'` | `'on'`
Default: `'off'`

Version History

Introduced in R2009a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

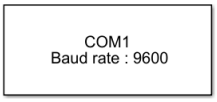
See Also

Topics

“Design Your Model for Effective Acceleration”

Serial Configuration

Configure parameters for serial port



Libraries:
Instrument Control Toolbox
Motor Control Blockset / Protection and Diagnostics

Description

The Serial Configuration block configures parameters for a serial port that you can use to send and receive data. Use this block to set the parameters of your serial port before you set up the Serial Receive and the Serial Send blocks.

Note You must configure your serial port parameters using the Serial Configuration block before you specify the Serial Receive and Serial Send block parameters.

Other Supported Features

- The Serial Configuration block supports the use of Simulink Accelerator mode, but not Rapid Accelerator. This feature speeds up the execution of Simulink models.
- The Serial Configuration block supports the use of model referencing. This feature lets your model include other Simulink models as modular components.
- The Serial Configuration block supports C/C++ code generation. This feature allows you to generate C and C++ code using Simulink Coder.

For more information on these features, see the “Simulink” documentation.

Parameters

Port — Serial communication port
available communication ports

Serial port on your machine that you want to configure. Use this configured port to send and receive data with your Serial Send and Serial Receive blocks. If you have not configured a port, the block returns an error when you run your model.

Note Each Serial Send and Serial Receive block must have a configured Serial Configuration block. If you use multiple serial ports in your simulation, you must configure each port using a separate Serial Configuration block.

Programmatic Use
Block Parameter: `Port`
Type: character vector, string

Baud rate — Communication speed
9600 (default) | positive integer

Rate at which bits are transmitted for the serial interface, in bits per second.

Programmatic Use
Block Parameter: `BaudRate`
Type: character vector, string
Values: positive integer
Default: `'9600'`

Data bits — Number of bits to represent one character of data
8 (default) | 5 | 6 | 7

Number of data bits to transmit over the serial interface.

Programmatic Use
Block Parameter: `DataBits`
Type: character vector, string
Values: `'5'` | `'6'` | `'7'` | `'8'`
Default: `'8'`

Parity — Parity bit type
`none` (default) | `even` | `odd`

Parity bit type added to data transmitted by serial port. You can use this parameter to add a parity bit (also referred to as a check bit) to your data. Adding a parity bit to a string of binary code is a method of detecting errors in data transmission by ensuring that the total number of 1-bits is even or odd.

The value of the parity bit is determined by the number of 1s in a given set of bits and is set as follows.

Parity Bit Type	Parity Bit Value	
	If number of 1s is even	If number of 1s is odd
`none`	No parity bit set	No parity bit set
`even`	0	1
`odd`	1	0

Note Starting in R2021a, the **Parity** parameter no longer supports `mark` or `space`. For more information, see .

Programmatic Use
Block Parameter: `Parity`
Type: character vector, string
Values: `'none'` | `'even'` | `'odd'`
Default: `'none'`

Stop bits — Pattern of bits that indicates the end of a character
1 (default) | positive scalar

Number of bits used to indicate the end of a byte. This parameter depends on the value you select for the **Data bits** parameter. If you select data bits `6`, `7`, or `8`, the default value is `1` and the other available choice is `2`. If you select data bit `5`, the default value is `1` and the other available choice is `1.5`.

Programmatic Use
Block Parameter: `StopBits`
Type: character vector, string
Values: positive scalar
Default: `'1'`

Byte order — Sequential order of bytes
`little-endian` (default) | `big-endian`

Sequential order in which bytes are arranged into larger numerical values. If the byte order is `little-endian`, then the instrument stores the first byte in the first memory address. If the byte order is `big-endian`, then the instrument stores the last byte in the first memory address.

Configure the byte order to the appropriate value for your instrument before performing a read or write operation. Refer to your instrument documentation for information about the order in which it stores bytes.

Programmatic Use
Block Parameter: `ByteOrder`
Type: character vector, string
Values: `'little-endian'` | `'big-endian'`
Default: `'little-endian'`

Flow control — Mode for managing data transmission rate
`none` (default) | `hardware`

Process of managing the rate of data transmission on your serial port. Select `none` to have no flow control or `hardware` to let your hardware determine the flow control.

Programmatic Use
Block Parameter: `FlowControl`
Type: character vector, string
Values: `'none'` | `'hardware'`
Default: `none`

Timeout — Allowed time to complete operations
10 (default) | positive scalar

Amount of time that the model waits for data during each simulation time step.

Programmatic Use
Block Parameter: `Timeout`
Type: character vector, string
Values: positive scalar
Default: `'10'`

Version History

Introduced in R2008a

R2021a: Parity parameter no longer supports `mark` or `space` in Serial Configuration block
Errors starting in R2021a

The `mark` and `space` options for the **Parity** parameter are no longer supported in the Serial Configuration block. Valid values for **Parity** are `none` (default), `even`, and `odd`.

If you try to open an existing model that has the **Parity** value set to `mark` or `space`, MATLAB returns a warning and changes the parameter to the default value `none`.

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

This block generates platform-specific code for the host machine's platform only (Windows®, macOS, Linux®).

See Also

Serial Receive | Serial Send

Serial Receive

Receive binary data over serial port

Libraries:
Instrument Control Toolbox
Motor Control Blockset / Protection and Diagnostics

Description

The Serial Receive block configures and opens an interface to the specified serial port. The configuration and initialization occur once at the start of the model's execution. The block acquires data from the serial port during the model's run time. You can use only one Serial Receive block at a time to receive data from a specific serial port.

Note You must configure your serial port parameters using the Serial Configuration block before you specify the Serial Receive block parameters.

This block has no input ports. It has one or two output ports based on whether you select blocking or non-blocking mode. If you select blocking mode, the block has one output port, **Data**, corresponding to the data it receives. If you do not select blocking mode, the block has two output ports, **Data** and **Status**.

This block uses a First In, First Out (FIFO) buffer to receive data from the serial port. At each time step, the **Data** port returns the requested values from the buffer. In non-blocking mode, the **Status** port indicates if the block has received new data. If the **Status** port displays `1`, new data is available and if the **Status** port displays `0`, no new data is available.

Other Supported Features

- The Serial Receive block supports the use of Simulink Accelerator mode, but not Rapid Accelerator. This feature speeds up the execution of Simulink models.
- The Serial Receive block supports the use of model referencing. This feature lets your model include other Simulink models as modular components.
- The Serial Receive block supports C/C++ code generation. This feature allows you to generate C and C++ code using Simulink Coder.

For more information on these features, see the “Simulink” documentation.

Ports

Output

Data — Data received
vector | matrix | array

Data received by the block from the serial port, returned as a vector, matrix, or array.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Status — New data available
`false` or `0` (default) | `true` or `1`

New data available status, returned as numeric or logical `1` (`true`) or `0` (`false`). If this port returns `1`, new data is available to be read.

Dependencies

To enable this port, unselect the **`Enable blocking mode`** parameter.

Data Types: `Boolean`

Parameters

Port — Serial communication port
available communication ports

Serial port on your machine that you want to receive data from. Select a port from the available ports and then configure the port using the Serial Configuration block. If you have not configured a port, the block returns an error when you run your model.

Note Each Serial Receive block must have a configured Serial Configuration block. If you use multiple serial ports in your simulation, you must configure each port using a separate Serial Configuration block.

Programmatic Use
Block Parameter: `Port`
Type: character vector, string

Data type — Output data type
`uint8` (default) | `single` | `double` | `int8` | `int16` | `uint16` | `int32` | `uint32`

Data type that the block receives from the serial port, specified as a MATLAB numeric data type.

Programmatic Use
Block Parameter: `DataType`
Type: character vector, string
Values: `'uint8'` | `'single'` | `'double'` | `'int8'` | `'int16'` | `'uint16'` | `'int32'` | `'uint32'`
Default: `'uint8'`

Header — Header
numeric array | integer from 0 to 255, inclusive

If this parameter is selected, you can specify the header that indicates the beginning of your data block. The simulation disregards data that occurs before the header. The header data is not sent to the output port. By default, this parameter is not selected and no header is specified.

The numeric array specified in this parameter is the `uint8` integer representation of the corresponding ASCII characters. The exact form of this parameter depends on the type of the ASCII character.

Type of ASCII Character	Example ASCII Character	MATLAB Command	Parameter Value
Special character	"#"	uint8('#')	[35]
Numeric	"81"	uint8('81')	[56 49]
Alphabet	"Start"	uint8('Start')	[83 116 97 114 116]

You can also specify this parameter using the hexadecimal representation of the ASCII characters.

Programmatic Use
Block Parameter: ToggleHeader
Type: character vector, string
Values: 'on' | 'off'
Default: 'off'
Block Parameter: Header
Type: character vector, string
Values: integer array

Terminator — Terminator
<none> (default) | CR ('\r') | LF ('\n') | CR/LF ('\r\n') | NULL ('\0') | Custom Terminator

If this parameter is selected, you can specify the terminator that indicates the end of your data block. The simulation considers any data that occurs after the terminator as a new data block. The terminator data is not sent to the output port. This terminator must match the terminator in the data you are reading from your serial port, if it has one.

If you select Custom Terminator, you can specify your own terminator value.

Programmatic Use
Block Parameter: ToggleTerminator
Type: character vector, string
Values: 'on' | 'off'
Default: 'off'
Block Parameter: Terminator
Type: character vector, string
Values: '<none>' | 'CR ('\r')' | 'LF ('\n')' | 'CR/LF ('\r\n')' | 'NULL ('\0')' | 'Custom Terminator'
Default: '<none>'

Custom terminator — Custom terminator
numeric array | integer from 0 to 255, inclusive

Custom terminator that indicates the end of your data block. The simulation considers any data that occurs after the terminator as a new data block. The terminator data is not sent to the output port.

The numeric array specified in this parameter is the uint8 integer representation of the corresponding ASCII characters. The exact form of this parameter depends on the type of the ASCII character.

Type of ASCII Character	Example ASCII Character	MATLAB Command	Parameter Value
Special character	`"#"`	`uint8('#')`	`[35]`
Numeric	`"81"`	`uint8('81')`	`[56 49]`
Alphabet	`"End"`	`uint8('End')`	`[69 110 100]`

You can also specify this parameter using the hexadecimal representation of the ASCII characters.

Programmatic Use
Block Parameter: `CustomTerminator`
Type: character vector, string
Values: integer array

Input Format — Format of data read
`Column major` (default) | `Row major`

Format of the data that the block receives from the serial port, specified as `Row major` or `Column major`.

Programmatic Use
Block Parameter: `InputFormat`
Type: character vector, string
Values: `'Row major'` | `'Column major'`
Default: `'Column major'`

Data size — Number of values read
`[1 1]` (default) | numeric array

Output data size, or the number of values that should be read at each simulation time step. This parameter is specified as a multidimensional numeric array. The data size does not include the header or terminator values.

Programmatic Use
Block Parameter: `DataSize`
Type: character vector, string
Values: integer array
Default: `'[1 1]'`

Enable blocking mode — Simulation waits while receiving data
`on` (default) | `off`

This parameter has the simulation wait while the block receives data. When new data becomes available, the simulation continues from the next time step. Unselect the check box if you do not want the read operation to cause the simulation to wait.

If you enable blocking mode, the simulation waits for the requested data to become available. The model waits for up to the amount of time specified by the **Timeout** parameter in the Serial Configuration block. If new data does not become available during a simulation, you can return an error by selecting the `Error` option for the **Action when data is not available** parameter.

If you do not enable blocking mode, the simulation runs continuously and the block has two output ports, **Status** and **Data**. The **Data** port contains the requested set of data at each time step. The **Status** port contains 0 or 1 based on whether it received new data at the given time step.

Programmatic Use
Block Parameter: `EnableBlockingMode`
Type: character vector, string
Values: `'on'` | `'off'`
Default: `'on'`

Action when data is not available — Action to take when data not available
`Output last received value` (default) | `Output custom value` | `Error`

Action the block should take when data is not available. Available options are:

- `Output last received value` — Block returns the value it received at the preceding time step when it does not receive data at current time step.
- `Output custom value` — Block returns any user-defined value when it does not receive current data. Define the custom value in the **Custom value** field.
- `Error` — Block returns an error when it does not receive current data. You must select **Enable blocking mode** to use this option.

Programmatic Use
Block Parameter: `ActionDataUnavailable`
Type: character vector, string
Values: `'Output last received value'` | `'Output custom value'` | `'Error'`
Default: `'Output last received value'`

Custom value — Custom output value when data unavailable
0 (default) | numeric

Custom value for the block to output when it does not receive new data. The custom value can be a scalar or value equal to the size of data that it receives (specified by **Data size** parameter). You must select `Output custom value` as the **Action when data is unavailable** to set this parameter.

Programmatic Use
Block Parameter: `CustomValue`
Type: character vector, string
Values: numeric
Default: `'0'`

Block sample time — Sampling time
0.01 (default) | positive numeric

Sampling time of the block during the simulation. This is the rate at which the block is executed during simulation.

Programmatic Use
Block Parameter: `SampleTime`
Type: character vector, string
Values: positive numeric
Default: `'0.01'`

Version History

Introduced in R2008a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

This block generates platform-specific code for the host machine's platform only (Windows, macOS, Linux).

See Also

Serial Configuration | Serial Send

Serial Send

Send binary data over serial port

Libraries:
Instrument Control Toolbox
Motor Control Blockset / Protection and Diagnostics

Description

The Serial Send block configures and opens an interface to the specified serial port. The configuration and initialization occur once at the start of the model's execution. The block sends data from the model to the serial port during the model's run time. You can use multiple Serial Send blocks at a time to send data to a specific serial port.

Note You must configure your serial port parameters using the Serial Configuration block before you specify the Serial Send block parameters.

The Serial Send block has one input port that accepts both 1-D vector and matrix data. This block has no output ports. The block inherits the data type from the signal at the input port. Valid data types are `single`, `double`, `int8`, `uint8`, `int16`, `uin16`, `int32`, and `uint32`.

Other Supported Features

- The Serial Send block supports the use of Simulink Accelerator mode, but not Rapid Accelerator. This feature speeds up the execution of Simulink models.
- The Serial Send block supports the use of model referencing. This feature lets your model include other Simulink models as modular components.
- The Serial Send block supports C/C++ code generation. This feature allows you to generate C and C++ code using Simulink Coder.

For more information on these features, see the “Simulink” documentation.

Ports

Input

Data — Data values to send
vector | matrix | array

Data values to send from the block over your serial port, specified as a vector, matrix, or array. Set the parameters for this block before you send data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Parameters

Port — Serial communication port
available communication ports

Serial ports on your machine that you want to send data to. Select a port from the available ports and then configure the port using the Serial Configuration block. If you have not configured a port, the block returns an error when you run your model.

Note Each Serial Send block must have a configured Serial Configuration block. If you use multiple serial ports in your simulation, you must configure each port using a separate Serial Configuration block.

Programmatic Use
Block Parameter: `Port`
Type: character vector, string

Header — Header
numeric array | integer from 0 to 255, inclusive

Header that indicates the beginning of your data block. The Serial Send block adds the header in front of the data before sending it over the serial port. By default, no header is specified.

The numeric array specified in this parameter is the `uint8` integer representation of the corresponding ASCII characters. The exact form of this parameter depends on the type of the ASCII character.

Type of ASCII Character	Example ASCII Character	MATLAB Command	Parameter Value
Special character	`"#"`	`uint8('#')`	`[35]`
Numeric	`"81"`	`uint8('81')`	`[56 49]`
Alphabet	`"Start"`	`uint8('Start')`	`[83 116 97 114 116]`

You can also specify this parameter using the hexadecimal representation of the ASCII characters.

Programmatic Use
Block Parameter: `Header`
Type: character vector, string
Values: integer array

Terminator — Terminator
<none> (default) | `CR ('\r')` | `LF ('\n')` | `CR/LF ('\r\n')` | `NULL ('\0')` | `Custom Terminator`

Terminator that indicates the end of your data block. The Serial Send block appends the terminator to the data before sending it over the serial port.

If you select `Custom Terminator`, you can specify your own terminator value.

Programmatic Use
Block Parameter: `Terminator`
Type: character vector, string
Values: `'<none>'` | `'CR ('\r')'` | `'LF ('\n')'` | `'CR/LF ('\r\n')'` | `'NULL ('\0')'` | `'Custom Terminator'`
Default: `'<none>'`

Custom terminator — Custom terminator
numeric array | integer from 0 to 255, inclusive

Custom terminator that indicates the end of your data block. The Serial Send block appends the terminator to the data before sending it over the serial port.

The numeric array specified in this parameter is the `uint8` integer representation of the corresponding ASCII characters. The exact form of this parameter depends on the type of the ASCII character.

Type of ASCII Character	Example ASCII Character	MATLAB Command	Parameter Value
Special character	`"#"`	`uint8('#')`	`[35]`
Numeric	`"81"`	`uint8('81')`	`[56 49]`
Alphabet	`"End"`	`uint8('End')`	`[69 110 100]`

You can also specify this parameter using the hexadecimal representation of the ASCII characters.

Programmatic Use
Block Parameter: `CustomTerminator`
Type: character vector, string
Values: integer array

Enable blocking mode — Simulation waits while sending data
`on` (default) | `off`

This parameter has the simulation wait while the block sends data. Unselect the check box if you do not want the write operation to cause the simulation to wait.

If you enable blocking mode, the simulation waits for the data to be sent. If you do not enable blocking mode, the simulation runs continuously.

Programmatic Use
Block Parameter: `EnableBlockingMode`
Type: character vector, string
Values: `'on'` | `'off'`
Default: `'on'`

Version History

Introduced in R2008a

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

This block generates platform-specific code for the host machine's platform only (Windows, macOS, Linux).

See Also

Serial Configuration | Serial Receive

Appendix