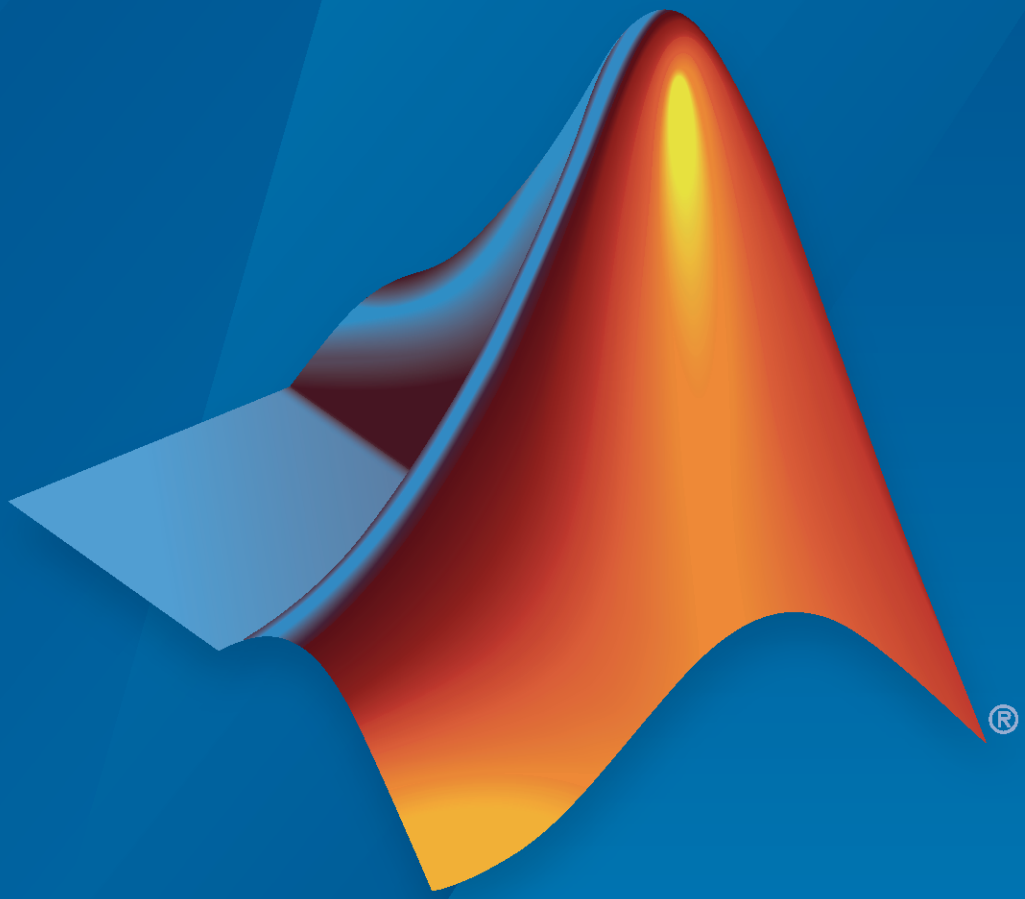
SoC Blockset™

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

SoC Blockset™ Reference

Revision History

March 2019	Online Only	New for Version 1.0 (Release 2019a)
September 2019	Online only	Revised for Version 1.1 (Release 2019b)
March 2020	Online only	Revised for Version 1.2 (Release 2020a)
September 2020	Online only	Revised for Version 1.3 (Release 2020b)
March 2021	Online only	Revised for Version 1.4 (Release 2021a)
September 2021	Online only	Revised for Version 1.5 (Release 2021b)
March 2022	Online only	Revised for Version 1.6 (Release 2022a)
September 2022	Online only	Revised for Version 1.7 (Release 2022b)
March 2023	Online only	Revised for Version 1.8 (Release 2023a)

Contents

1 Blocks

2 Configuration Parameters

Hardware Implementation Pane **2-2**
Hardware Implementation Pane Overview **2-2**
Hardware board settings **2-3**
Design Mapping **2-3**
Task profiling in simulation **2-3**
Task profiling on processor **2-3**
Operating system/scheduler **2-4**
Simulation Settings **2-4**
Board Parameters **2-4**
Processor **2-4**
Board Options **2-4**
Clocking **2-5**
External Mode **2-5**
FPGA design (top-level) **2-5**
FPGA design (PS mem controllers) **2-6**
FPGA design (PL mem controllers) **2-6**
FPGA design (mem channels) **2-7**
FPGA design (debug) **2-7**

Hardware Board Settings **2-8**
Processing Unit **2-8**

Design Mapping **2-9**
View/Edit Hardware Mapping **2-9**

Task Profiling in Simulation **2-10**
Show in SDI **2-10**
Save to file **2-10**
Overwrite file **2-10**

Task Profiling on Processor **2-11**
Show in SDI **2-11**
Save to file **2-11**
Overwrite file **2-11**
Instrumentation **2-11**
Profiling duration **2-11**

Operating System/Scheduler **2-13**
Kernel latency **2-13**

Simulation Settings **2-14**
Set random number generator seed **2-14**
Seed Value **2-14**
Cache input data at task start **2-14**

Processor **2-15**
Number of cores **2-15**

Board Parameters **2-16**
Device Address **2-16**
Username **2-16**
Password **2-16**

Clocking **2-17**
CPU Clock (MHz) **2-17**

Build Action **2-18**
Settings **2-18**

External Mode **2-19**
Communication Interface **2-19**
Run external mode in a background thread **2-19**
Port **2-19**
Verbose **2-20**

FPGA design (top-level) **2-21**
View/Edit Memory Map **2-21**
Include 'MATLAB as AXI Master' IP for host-based interaction **2-21**
Include processing system **2-21**
Interrupt latency (s) **2-21**
Register configuration clock frequency (MHz) **2-21**
IP core clock frequency (MHz) **2-21**

FPGA design (mem controllers) **2-23**
Controller clock frequency (MHz) **2-23**
Controller data width (bits) **2-23**
Bandwidth derating (%) **2-23**
First write transfer latency (clocks) **2-23**
Last write transfer latency (clocks) **2-24**
First read transfer latency (clocks) **2-24**
Last read transfer latency (clocks) **2-24**

FPGA design (mem channels) **2-25**
Interconnect clock frequency (MHz) **2-25**
Interconnect data width (bits) **2-25**
Interconnect FIFO depth (num bursts) **2-25**
Interconnect almost-full depth **2-25**

FPGA design (debug) **2-26**
Memory channel diagnostic level **2-26**
Include AXI interconnect monitor **2-26**
Trace capture depth **2-26**

3 Functions

4 Objects

5 Tools

6 Blocks

7 Coder Target Context Sensitive Help

Model Configuration Parameters **7-2**
Hardware Implementation Pane Overview **7-2**
Design Mapping **7-3**
Target hardware resources **7-4**

Design Mapping **7-6**
View/Edit Hardware Mapping **7-6**

Device **7-7**
Series **7-7**
Step **7-7**
Package type **7-7**
Package class **7-7**
Pinout **7-7**
Enable CPU# **7-8**
Enable program cache # **7-8**
Enable data cache # **7-8**

Build Options **7-9**
Build action **7-9**
Disable parallel build **7-9**
Enable force rebuild of static library **7-9**
Boot From **7-10**
Use custom linker command file **7-11**
Linker file **7-11**

Clocking **7-12**
External oscillator (MHz) **7-12**
CPU Clock (MHz) **7-12**

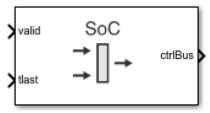
Connectivity **7-13**
Connectivity interface **7-13**
Port **7-13**
Baudrate **7-13**

8 Tools

Blocks

SoC Bus Creator

Convert control signals to bus



Libraries:
SoC Blockset / Hardware Logic Connectivity

Description

The SoC Bus Creator block combines a set of signals into a bus. The block accepts control signals and outputs a bus.

You can configure this block to support multiple protocol interface types. Parameter and port configurations for this block vary based on your desired protocol interface type and mode of operation, as outlined in this table.

Protocol Interface Type	Mode of Operation	Parameter Configuration	Enabled Input Ports
Data stream	Read data stream	Set **Control protocol** to `Data stream` and **Control type** to `Ready`.	**ready**
	Write data stream	Set **Control protocol** to `Data stream` and **Control type** to `Valid`.	**valid**
			tlast
Pixel stream	Read video stream	Set **Control protocol** to `Pixel stream` and **Control type** to `Ready`.	**ready**
	Write video stream	Set **Control protocol** to `Pixel stream` and **Control type** to `Valid`.	**hStart**
			hEnd
			vStart
			vEnd
			valid
	Read video stream with frame sync	Set **Control protocol** to `Pixel stream` and **Control type** to `Ready frame with sync`.	**ready**
			fsync
Random access read	Read data	Set **Control protocol** to `Random access read` and **Control type** to `Ready`.	**rd_addr**
			rd_len
			rd_avalid
			rd_dready

Protocol Interface Type	Mode of Operation	Parameter Configuration	Enabled Input Ports
Random access write	Write data	Set **Control protocol** to `Random access write` and **Control type** to `Valid`.	**wr_addr**
			wr_len
			wr_valid

Ports

Input

valid — Valid control signal
boolean scalar

Valid control signal, specified as a scalar. You can use this port for data stream and pixel stream protocols only.

Dependencies

To enable this port, set the **Control protocol** parameter to either `Data stream` or `Pixel stream` and the **Control type** parameter to `Valid`.

Data Types: `Boolean`

tlast — Indication of end of data packet
boolean scalar

Indication of end of the data packet, specified as a Boolean scalar.

Dependencies

To enable this port, set the **Control protocol** parameter to `Data stream` and the **Control type** parameter to `Valid`.

Data Types: `Boolean`

ready — Ready control signal
boolean scalar

Ready control signal, specified as a Boolean scalar. This port is available for `Data stream` and `Pixel stream` control protocols.

Dependencies

To enable this port, set the **Control protocol** parameter to either `Data stream` or `Pixel stream` and the **Control type** parameter to `Ready` or `Ready with frame sync`.

Data Types: `Boolean`

hStart — First pixel in horizontal line of frame
boolean scalar

First pixel in a horizontal line of a frame, specified as a Boolean scalar.

Dependencies

To enable this port, set the **Control protocol** parameter to `Pixel stream` and the **Control type** parameter to `Valid`.

Data Types: `Boolean`

hEnd — Last pixel in horizontal line of frame
boolean scalar

Last pixel in a horizontal line of a frame, specified as a Boolean scalar.

Dependencies

To enable this port, set the **Control protocol** parameter to `Pixel stream` and the **Control type** parameter to `Valid`.

Data Types: `Boolean`

vStart — First pixel in first (top) line of frame
boolean scalar

First pixel in the first (top) line of a frame, specified as a Boolean scalar.

Dependencies

To enable this port, set the **Control protocol** parameter to `Pixel stream` and the **Control type** parameter to `Valid`.

Data Types: `Boolean`

vEnd — Last pixel in last (bottom) line of frame
boolean scalar

Last pixel in the last (bottom) line of a frame, specified as a Boolean scalar.

Dependencies

To enable this port, set the **Control protocol** parameter to `Pixel stream` and the **Control type** parameter to `Valid`.

Data Types: `Boolean`

fsync — Frame synchronization
boolean scalar

Frame synchronization, specified as a Boolean scalar.

Dependencies

To enable this port, set the **Control protocol** parameter to `Pixel stream` and the **Control type** parameter to `Ready with frame sync`.

Data Types: `Boolean`

rd_addr — Reader address
scalar

Reader address, specified as a scalar. It is the starting address for the read transaction that is sampled at the first cycle of the transaction.

Dependencies

To enable this port, set the **Control protocol** parameter to `Random access read`.

Data Types: `uint32`

rd_len — Reader data length
scalar

Reader data length, specified as a scalar. It means the number of data values that you want to read, sampled at the first cycle of the transaction.

Dependencies

To enable this port, set the **Control protocol** parameter to `Random access read`.

Data Types: `uint32`

rd_avalid — Reader valid status
boolean scalar

Reader valid status, specified as a Boolean scalar. It indicates whether the read request is valid.

Dependencies

To enable this port, set the **Control protocol** parameter to `Random access read`.

Data Types: `Boolean`

rd_dready — Reader ready status
boolean scalar

Reader ready status, specified as a Boolean scalar. It indicates when the hardware logic can start accepting data.

Dependencies

To enable this port, set the **Control protocol** parameter to `Random access read`.

Data Types: `Boolean`

wr_addr — Writer address
scalar

Specify the starting address to which the hardware writes.

Dependencies

To enable this port, set the **Control protocol** parameter to `Random access write`.

Data Types: `uint32`

wr_len — Writer data length
scalar

Specify the number of data elements in the write transaction.

Dependencies

To enable this port, set the **Control protocol** parameter to `Random access write`.

Data Types: `uint32`

wr_valid — Writer valid data
boolean scalar

Writer valid data, specified as a scalar. It indicates the data signal sampled at the output is valid.

Dependencies

To enable this port, set the **Control protocol** parameter to `Random access write`.

Data Types: `Boolean`

Output

ctrlBus — Output control bus
bus

Output control bus, returned as a bus.

The data type of the output control bus depends on the values of the **Control protocol** and **Control type** parameters.

Parameter Configuration	Output Data Type
Set **Control protocol** to `Data stream` and **Control type** to `Ready`.	`StreamS2MBusObj`
Set **Control protocol** to `Data stream` and **Control type** to `Valid`.	`StreamM2SBusObj`
Set **Control protocol** to `Pixel stream` and **Control type** to `Ready`.	`StreamVideoS2MBusObj`
Set **Control protocol** to `Pixel stream` and **Control type** to `Valid`.	`pixelcontrol`
Set **Control protocol** to `Pixel stream` and **Control type** to `Ready frame with sync`.	`StreamvideoFsyncS2MBusObj`
Set **Control protocol** to `Random access read` and **Control type** to `Ready`.	`ReadControlM2SBusObj`
Set **Control protocol** to `Random access write` and **Control type** to `Valid`.	`WriteControlM2SBusObj`

Data Types: `StreamS2MBusObj` | `StreamM2SBusObj` | `StreamVideoS2MBusObj` | `pixelcontrol` | `StreamvideoFsyncS2MBusObj` | `ReadControlM2SBusObj` | `WriteControlM2SBusObj`

Parameters

Control protocol — Protocol interface selection
`Data stream` (default) | `Pixel stream` | `Random access read` | `Random access write`

Specify the protocol interface as one of these values:

- `Data stream` — Use this protocol if you require AXI4 data stream.
- `Pixel stream` — Use this protocol if you require AXI4 video stream.

- `Random access read` — Use this protocol if you require AXI4 read.
- `Random access write` — Use this protocol if you require AXI4 write.

The input ports of the block vary based on the type of **Control protocol** and **Control type** that you select. For more details, see “Description” on page 1-2.

Control type — Control type selection
`Valid` (default) | `Ready` | `Ready with frame sync`

Specify the type of control.

To enable the `Ready with frame sync` option, set the **Control protocol** parameter to `Pixel stream`.

The input ports of the block vary based on the type of **Control protocol** and **Control type** that you select. For more details, see “Description” on page 1-2.

Version History

Introduced in R2019a

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

SoC Bus Selector

SoC Bus Selector

Convert bus to control signals

Libraries:
SoC Blockset / Hardware Logic Connectivity

Description

The SoC Bus Selector block converts a set of control signals from a bus. The block accepts a bus and outputs control signals.

You can configure this block to support multiple protocol interface types. Parameter and port configurations for this block vary based on your desired protocol interface type and mode of operation, as outlined in this table.

Protocol Interface Type	Mode of Operation	Parameter Configuration	Enabled Output Ports
Data stream	Read stream data	Set **Control protocol** to `Data stream` and **Control type** to `Valid`.	**valid**
			tlast
	Write stream data	Set **Control protocol** to `Data stream` and **Control type** to `Ready`.	**ready**
Pixel stream	Read video stream	Set **Control protocol** to `Pixel stream` and **Control type** to `Valid`.	**hStart**
			hEnd
			vStart
			vEnd
			valid
	Write video stream	Set **Control protocol** to `Pixel stream` and **Control type** to `Ready`.	**ready**
Random access read	Read data	Set **Control protocol** to `Random access read` and **Control type** to `Valid`.	**rd_aready**
			rd_dvalid
Random access write	Write data	Set **Control protocol** to `Random access write` and **Control type** to `Ready`.	**wr_ready**
			wr_bvalid
			wr_complete

Ports

Input

ctrlBus — Input control bus
bus

Input control bus, specified as a bus.

The data type of the input control bus depends on the values of the **Control protocol** and **Control type** parameters.

Parameter Configuration	Input Data Type
Set **Control protocol** to `Data stream` and **Control type** to `Valid`.	`StreamM2SBusObj`
Set **Control protocol** to `Data stream` and **Control type** to `Ready`.	`StreamS2MBusObj`
Set **Control protocol** to `Pixel stream` and **Control type** to `Valid`.	`pixelcontrol`
Set **Control protocol** to `Pixel stream` and **Control type** to `Ready`.	`StreamVideoS2MBusObj`
Set **Control protocol** to `Random access read` and **Control type** to `Valid`.	`ReadControlS2MBusObj`
Set **Control protocol** to `Random access write` and **Control type** to `Ready`.	`WriteControlS2MBusObj`

Data Types: `StreamM2SBusObj` | `StreamS2MBusObj` | `pixelcontrol` | `StreamVideoS2MBusObj` | `ReadControlS2MBusObj` | `WriteControlS2MBusObj`

Output

valid — Valid control signal
boolean scalar

Valid control signal, returned as a scalar. You can use this port for data stream and pixel stream protocols only.

Dependencies

To enable this port, set the **Control protocol** parameter to either `Data stream` or `Pixel stream` and the **Control type** parameter to `Valid`.

Data Types: `Boolean`

tlast — Indication of end of data packet
boolean scalar

Indication of end of the data packet, returned as a Boolean scalar.

Dependencies

To enable this port, set the **Control protocol** parameter to `Data stream` and the **Control type** parameter to `Valid`.

Data Types: `Boolean`

ready — Ready control signal
boolean scalar

Ready control signal, returned as a Boolean scalar. This port is available for `Data stream` and `Pixel stream` control protocols.

Dependencies

To enable this port, set the **Control protocol** parameter to either `Data stream` or `Pixel stream` and the **Control type** parameter to `Ready`.

Data Types: `Boolean`

hStart — First pixel in horizontal line of frame
boolean scalar

First pixel in a horizontal line of a frame, returned as a Boolean scalar.

Dependencies

To enable this port, set the **Control protocol** parameter to `Pixel stream` and the **Control type** parameter to `Valid`.

Data Types: `Boolean`

hEnd — Last pixel in horizontal line of frame
boolean scalar

Last pixel in a horizontal line of a frame, returned as a Boolean scalar.

Dependencies

To enable this port, set the **Control protocol** parameter to `Pixel stream` and the **Control type** parameter to `Valid`.

Data Types: `Boolean`

vStart — First pixel in first (top) line of frame
boolean scalar

First pixel in the first (top) line of a frame, returned as a Boolean scalar.

Dependencies

To enable this port, set the **Control protocol** parameter to `Pixel stream` and the **Control type** parameter to `Valid`.

Data Types: `Boolean`

vEnd — Last pixel in last (bottom) line of frame
boolean scalar

Last pixel in the last (bottom) line of a frame, returned as a Boolean scalar.

Dependencies

To enable this port, set the **Control protocol** parameter to `Pixel stream` and the **Control type** parameter to `Valid`.

Data Types: `Boolean`

rd_aready — Accept read requests
boolean scalar

Accept read requests, returned as a scalar. It indicates when to accept read requests.

Dependencies

To enable this port, set the **Control protocol** parameter to `Random access read`.

Data Types: `Boolean`

rd_dvalid — Read request valid
boolean scalar

Read request valid, returned as a Boolean scalar. It is the control signal that indicates the data returned from the read request is valid.

Dependencies

To enable this port, set the **Control protocol** parameter to `Random access read`.

Data Types: `Boolean`

wr_ready — Write ready signal
boolean scalar

Write ready signal, returned as a Boolean scalar. It corresponds to the backpressure from the slave IP core or external memory. When this value is `1` (`high`), it indicates that data can be sent. When this value is `0` (`low`), it indicates that the hardware logic must stop sending data within one clock cycle.

Dependencies

To enable this port, set the **Control protocol** parameter to `Random access write`.

Data Types: `Boolean`

wr_bvalid — Write valid signal
boolean scalar

Write valid signal, returned as a Boolean scalar. It is the response signal from the slave IP core that you can use for diagnosis purposes. This value becomes `1` (`high`) after the AXI4 interconnect accepts each burst transaction.

Dependencies

To enable this port, set the **Control protocol** parameter to `Random access write`.

Data Types: `Boolean`

wr_complete — Write transaction complete
boolean scalar

Write transaction complete, specified as a Boolean scalar. It is the control signal that when remains high for one clock cycle indicates that the write transaction has completed. This signal asserts at the last `wr_bvalid` of the burst.

Dependencies

To enable this port, set the **Control protocol** parameter to `Random access write`.

Data Types: `Boolean`

Parameters

Control protocol — Protocol interface selection
`Data stream` (default) | `Pixel stream` | `Random access read` | `Random access write`

Specify the protocol interface as one of these values:

- `Data stream` — Use this protocol if you require AXI4 data stream.
- `Pixel stream` — Use this protocol if you require AXI4 video stream.
- `Random access read` — Use this protocol if you require AXI4 read.
- `Random access write` — Use this protocol if you require AXI4 write.

The output ports of the block vary based on the type of **Control protocol** and **Control type** that you select. For more details, see “Description” on page 1-8.

Control type — Control type selection
`Valid` (default) | `Ready`

Specify the type of control.

The output ports of the block vary based on the type of **Control protocol** and **Control type** that you select. For more details, see “Description” on page 1-8.

Version History

Introduced in R2019a

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion

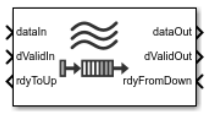
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

SoC Bus Creator

Stream FIFO

Control backpressure between hardware logic and upstream data interface



Libraries:
SoC Blockset / Hardware Logic Connectivity

Description

The Stream FIFO block controls the backpressure from the hardware logic to the upstream data interface. It also controls the flow between the upstream and downstream data interfaces of the hardware logic. Integrate this block as a configurable first-in, first-out (FIFO) block for AXI4 data stream applications. The block enables you to configure its depth and set its almost full threshold value.

Ports

Input

dataIn — Input stream data
scalar

Input stream data from the data source. Specify this value as a scalar.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

dValidIn — Indication of valid input stream data
Boolean scalar

Control signal that indicates if the input stream data from the data source is valid. When this value is `1` (true), the block accepts the values on the **dataIn** port. When this value is `0` (false), the block ignores the values on the **dataIn** port.

Data Types: `Boolean`

rdyFromDown — Ready signal from downstream interface
Boolean scalar

Control signal that indicates if the block can send stream data to the downstream interface. When this value is `1` (true), the downstream interface is ready, and the block can send the stream data. When this value is `0` (false), the downstream interface is not ready, and the block cannot send the stream data.

Data Types: `Boolean`

Output

dataOut — Output stream data
scalar

Output stream data to the downstream interface. The data type of this output data is the same as the data type of the input data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

dValidOut — Indication of valid output stream data
Boolean scalar

Control signal that indicates if the output stream data is valid. When this value is `1` (true), the output stream data on the **dataOut** port is valid. When this value is `0` (false), the output stream data on the **dataOut** port is not valid.

Data Types: `Boolean`

rdyToUp — Ready signal to upstream interface
Boolean scalar

Control signal that indicates if the block is ready to receive stream data from the upstream interface. When this value is `1` (true), the block is ready to accept stream data from the upstream interface. When this value is `0` (false), the block is not ready to accept stream data from the upstream interface.

Data Types: `Boolean`

Parameters

Depth of FIFO — FIFO depth

16 (default) | positive integer

Specify the depth of the FIFO. This value must be a positive integer and is the maximum number of entries that can be buffered before data gets dropped.

Almost full threshold — Almost full threshold value

8 (default) | positive integer

Specify a value that asserts a back-pressure signal from the block to the data source.

To avoid dropping data, set a value allowing the data source enough time to react to backpressure. This value must be a positive integer and smaller than the FIFO depth.

Version History

Introduced in R2019a

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

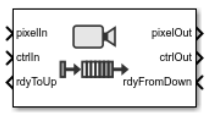
To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also
Video Stream FIFO

Video Stream FIFO

Control backpressure between hardware logic and upstream video interface



Libraries:
SoC Blockset / Hardware Logic Connectivity

Description

The Video Stream FIFO block controls the back-pressure from the hardware logic to the upstream video interface. It also controls the flow between the upstream and downstream pixel data interfaces of hardware logic. Integrate this block as a configurable first-in, first-out (FIFO) block for AXI4 video stream applications. The block enables you to configure its depth and set its almost full threshold value.

Ports

Input

pixelIn — Input pixel data
scalar

Input pixel data from the data source. Specify this value as a scalar.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

ctrlIn — Control signals accompanying input pixel data
`pixelcontrol` bus

Control signals accompanying the pixel stream, specified as a `pixelcontrol` bus containing five signals. The signals describe the validity of the pixel and its location in the frame.

Data Types: `pixelcontrol`

rdyFromDown — Ready signal from downstream interface
Boolean scalar

Control signal that indicates if the block can send pixel data to the downstream interface. When this value is `1` (true), the downstream interface is ready, and the block can send the pixel data. When this value is `0` (false), the downstream interface is not ready, and the block cannot send the pixel data.

Data Types: `Boolean`

Output

pixelOut — Output pixel data
scalar

Output pixel data to the downstream interface. The data type of this output data is the same as the data type of the input data.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

ctrlOut — Control signals accompanying output pixel data
`pixelcontrol` bus

Control signals accompanying output pixel stream, returned as a `pixelcontrol` bus containing five signals. The signals describe the validity of the pixel and its location in the frame.

Data Types: `pixelcontrol`

rdyToUp — Ready signal to upstream interface
Boolean scalar

Control signal that indicates if the block is ready to receive pixel data from the upstream interface. When this value is `1` (true), the block is ready to accept pixel data from the upstream interface. When this value is `0` (false), the block is not ready to accept pixel data from the upstream interface.

Data Types: `Boolean`

Parameters

Depth of FIFO — FIFO depth

16 (default) | positive integer

Specify the depth of the FIFO. This value must be a positive scalar integer and is the maximum number of entries that can be buffered before data gets dropped.

Almost full threshold — Almost full threshold value

8 (default) | positive integer

Specify a value that asserts a back-pressure signal from the block to the data source.

To avoid dropping data, set a value allowing the data source enough time to react to backpressure. This value must be a positive integer and smaller than the FIFO depth.

Version History

Introduced in R2019a

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion

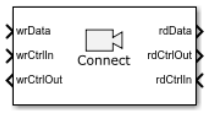
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Stream FIFO

Video Stream Connector

Connect two IPs with video streaming interfaces



Libraries:
SoC Blockset / Hardware Logic Connectivity

Description

The Stream Connector block connects two IPs with video streaming interfaces. Use this block in the FPGA model of an SoC application to connect two IPs.

Ports

Input

wrData — Input video data
scalar

Input video data from the data source. Specify this value as a scalar.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

wrCtrlIn — Input control signals accompanying pixel stream
`pixelControl` bus

Control signals accompanying the pixel stream, specified as a `pixelcontrol` bus containing five signals. The signals describe the validity of the pixel and its location in the frame. For additional information about the `pixelcontrol` bus type, see “AXI4-Stream Video Interface”.

Data Types: `pixelcontrol`

rdCtrlIn — Ready signal from downstream interface
boolean scalar

Control signal that indicates if the block can send video data to downstream interface. When this value is (true), the downstream block is ready to receive data.

Output

rdData — Output video data
scalar

Output video data to the downstream destination IP.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

rdCtrlOut — Output control signals accompanying output pixel stream
`pixelcontrol` bus

Control signals accompanying the output video data, specified as a `pixelcontrol` bus containing five signals. The signals describe the validity of the pixel and its location in the frame.

Data Types: `pixelcontrol`

wrCtrlOut — Ready signal to the upstream interface
boolean scalar

Control signal that indicates that the block can receive stream data from upstream interface.

Data Types: `Boolean`

Version History

Introduced in R2019a

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion

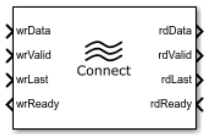
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Stream Connector

Stream Connector

Connect two IPs with data streaming interfaces



Libraries:
SoC Blockset / Hardware Logic Connectivity

Description

The Stream Connector block connects two IPs with data streaming interfaces. Use this block in the FPGA model of an SoC application to connect two IPs.

Ports

Input

wrData — Input stream data
scalar

Input stream data from the data source. Specify this value as a scalar.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

wrValid — Indication of valid input stream data
boolean scalar

Control signal that indicates if the input data from the data source is valid. When this value is (true), the block accepts the values on the **wrData** port. When this value is (false), the block ignores the value on the **wrData** port.

Data Types: `Boolean`

wrLast — Indication of last beat in burst
boolean scalar

Control signal that indicates the last beat of data from the upstream IP.

Data Types: `Boolean`

rdReady — Ready signal from downstream interface
boolean scalar

Control signal that indicates if the block can send stream data to the downstream interface. When this value is (true), the downstream block is ready to receive data.

Data Types: `Boolean`

Output

rdData — Output stream data
scalar

Output stream data to the downstream destination IP.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

rdValid — Indication of valid output stream data
boolean scalar

Control signal that indicates if the output stream data is valid.

Data Types: `Boolean`

rdLast — Indicates last beat in burst
boolean scalar

Control signal that indicates that the output stream data now has last beat of burst data.

Data Types: `Boolean`

wrReady — Ready signal to upstream interface
boolean scalar

Control signal that indicates if the block can receive stream data from the upstream interface.

Data Types: `Boolean`

Version History

Introduced in R2019a

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion

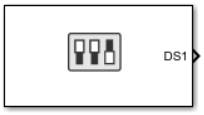
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Video Stream Connector

DIP Switch

Connect signals attached to DIP switches on hardware board



Libraries:
SoC Blockset / Hardware Logic I/O

Description

The DIP Switch block controls the hardware logic. The hardware logic signals connected to a DIP Switch block are equivalent to the signals connected to the dual inline package (DIP) switches on the hardware board.

Ports

Input

DSIn*x* — Input signal
`Boolean` scalar

Input signal to control the hardware logic. Using this port, you can dynamically control the hardware logic during simulation at run time. Each DIP switch has a port, named **DSIn1** to **DSIn*x***, where *x* is **Number of DIP switches**.

Dependencies

To enable this port, set the **Specify DIP switches via** parameter to `InputPort`.

Data Types: `Boolean`

Output

DS*x* — Output signal
`Boolean` scalar

Output signal that returns the state of the switch. Each DIP switch has a port, named **DS1** to **DS*x***, where *x* is **Number of DIP switches**.

Data Types: `Boolean`

Parameters

Hardware board — View selected hardware

None (default) | Supported Xilinx® or Intel ® boards | Custom boards

This parameter is read-only. To choose a hardware board and configure board parameters, see “Hardware Implementation Pane” on page 2-2.

View DIP switches location — View DIP switches

button

To view a diagram with the location of the DIP switches on the selected hardware board, click the **View DIP switches location** button.

This button is enabled only when you select specific Xilinx or Intel boards. For more information about these boards, refer to “Supported Third-Party Tools and Hardware”.

IO logic — IO logic indicator

None (default) | Active High | Active Low

This parameter is read-only. Indicates the IO logic level on the selected hardware board.

When the **IO logic** parameter is shown as **Active Low**, the DIP Switch block accepts and outputs active low signals when you set the **Specify DIP switches via** parameter to `InputPort` and outputs active low signals when you set the **Specify DIP switches via** parameter to `Dialog`. The block represents these port names prefixed with letter *n*. For example, **nDS1**.

Specify DIP switches via — DIP switch source
`Dialog` (default) | `InputPort`

To control the hardware logic by using the block parameters, select `Dialog`. To control the hardware logic from the input port, select `InputPort`.

Number of DIP switches — DIP switch selection
`1` (default) | list of integers in the range [1, n]

To specify the required number of DIP switch ports, select a value from the **Number of DIP switches** list. *n* represents the number of available DIP switches on the specified hardware board. For example, if you select `3` from the list, the block shows three DIP switch ports.

To use only the *n*th DIP switch, set the **Number of DIP switches** parameter to *n* and terminate the unused DIP switch ports.

DS*n* — Selected DIP switches
`Off` (default) | `On`

To enable the *n*th DIP switch port, select `On` for the DS*n* parameter. *n* represents the number of available DIP switches on the specified hardware board.

Dependencies

To enable this parameter, set the **Specify push buttons via** parameter to `Dialog`.

Sample time — System sample time
`-1` (default) | positive scalar

Specify the time interval a DIP switch toggles between `On` and `Off`.

Version History

Introduced in R2019a

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion

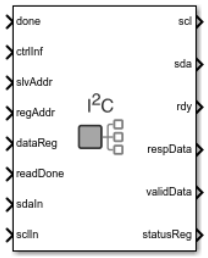
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

LED | Push Button

I2C Master

Configure and communicate with I2C slave device



Libraries:
SoC Blockset / Hardware Logic I/O

Description

The I2C Master block configures and communicates with an inter-integrated circuit communications (I2C) slave device connected to a field programmable gate array (FPGA). This block contains an I2C master controller with an AXI-Lite interface to perform the configuration.

The I2C Master block supports these features:

- AXI4-Lite interface support for configuration and access
- Single-master and multi-slave support
- Support 7-bit and 10-bit address I2C slave devices
- Burst mode support with a maximum burst size of 256 bytes
- Support multiple transmission speed modes
- An HDL-IP compatible model with code generation capability

The block uses the AXI-Lite interface to configure and create a control path interface to communicate with an I2C slave device. The hardware generated from the generation process contains an AXI-Lite register interface and two hardware interfaces, serial clock (SCL), and serial data (SDA). SCL and SDA connect the I2C Master block and the slave device.

Each port represented in the block is an AXI-Lite register, except the **sdaIn**, **sclIn**, **scl**, and **sda** ports. To communicate with a slave device, the AXI-Lite register interface configures the register information in the I2C Master block. This table contains the I2C Master AXI-Lite register information.

Register Address	Port and Register Name	Register Size in Bits	Operation Mode
0x100	**ctrlInf** — Control information	32	Write
0x104	**slvAddr** — Slave address	32	Write
0x108	**regAddr** — Register address	32	Write
0x10C	**dataReg** — Data register	32	Write

Register Address	Port and Register Name	Register Size in Bits	Operation Mode
0x110	**readDone** — Read done register	32	Write
0x114	**done** — Done register	32	Write
0x118	**rdy** — Ready register	32	Read
0x11C	**respData** — First response data register	32	Read
0x120	**validData** — Response data valid register	32	Read
0x124	**statusReg** — Status register	32	Read

To perform read and write operations using the I2C Master block, you need to follow a proper sequence. This section provides information about the sequence flow for read and write operations.

Read Sequence

To read data from an external slave device:

1. Send the **ctrlInf** register information.
2. Send the **slvAddr** register information.
3. Send the **regAddr** register information.
4. Set the **done** register to `1` after sending one set of register information to the block and then set it to `0`.
5. Read the response data from the external slave device. After reading the data from the **respData** register, set the **readDone** register to `1` and then set it to `0` immediately.
6. Set the **readDone** register to `1` again, to read more than 4 bytes of data. After the read operation, set it to `0` immediately.

In read sequence, one set of register information is a combination of **ctrlInf**, **slvAddr**, and **regAddr** registers.

Write Sequence

To write data to an external slave device:

1. Send the **ctrlInf** register information.
2. Send the **slvAddr** register information.
3. Send the **regAddr** register information.
4. Send the **dataReg** register that contains the data to write to the slave device register.
5. Set the **done** register to `1` after writing one set of register information to the block, and then set it to `0`.
6. Set the **done** register to `1` again, to write more than 4 bytes of data. After the write operation, set it to `0` immediately.

In write sequence, one set of register information is a combination of **ctrlInf**, **slvAddr**, **regAddr**, and **dataReg** registers.

Ports

Input

ctrlInf — Control information
scalar

Control information register contains configuration information on how the block communicates with the slave device, specified as a scalar. This register is a combination of read or write operation indication bit, number of bytes of slave-device register address, number of bytes of slave-device data register, and slave device address type bit. You can modify the configuration based on your requirement.

Bit	Purpose	Value Description
0	Set write or read mode.	To write to the slave-device register, set this value to `0`. To read from the slave-device register, set this value to `1`.
[2:1]	Set the size of the slave-device register address.	If the slave-device register address size is: • One byte (8 bits), set this value to 00 • Two bytes (16 bits), set this value to 01 • Three bytes (24 bits), set this value to 10 • Four bytes (32 bits), set this value to 11

Bit	Purpose	Value Description
[10:3]	Set the data size of the slave-device register.	If the slave-device register supports: • One byte of data, set this value to 00000000 • Two bytes of data, set this value to 00000001 • Three bytes of data, set this value to 00000010 • Four bytes of data, set this value to 00000011 • Five bytes of data, set this value to 00000100 • Six bytes of data, set this value to 00000101 • Seven bytes of data, set this value to 00000110 • Eight bytes of data, set this value to 00000111 • Nine bytes of data, set this value to 00001000 • Ten bytes of data, set this value to 00001001 • Eleven bytes of data, set this value to 00001010 • Twelve bytes of data, set this value to 00001011 • Thirteen bytes of data, set this value to 00001100 • Fourteen bytes of data, set this value to 00001101 • Fifteen bytes of data, set this value to 00001110 • Sixteen bytes of data, set this value to 00001111 ..... • 256 bytes of data, set this value to 11111111
11	Set the slave device type	To configure 7-bit address slave device, set this value to `0`. To configure 10-bit address slave device, set this value to `1`.

Data Types: `uint16`

slvAddr — Slave address
scalar

Slave-address register that contains the address of the slave device, specified as a scalar.

Data Types: `uint16`

regAddr — Register address
scalar

Register address of the slave device, specified as a scalar.

Data Types: `uint32`

dataReg — Data register
scalar

Data register, specified as a scalar. The block uses this port to write data to the slave-device register.

Data Types: `uint32`

readDone — Read done signal
`Boolean` scalar

Read done signal, specified as a `Boolean` scalar. When this value is `1` (true), the user is ready to read the response data from the block that is received from the slave device. When this value is `0` (false), the user is not ready to read the response data from the block.

Data Types: `Boolean`

done — Done signal
`Boolean` scalar

Done signal, specified as a `Boolean` scalar. This value indicates the block when to read the AXI-Lite register information.

Data Types: `Boolean`

sdaIn — Input serial data
`Boolean` scalar

Input serial data, returned as a `Boolean` scalar. This port provides a serial data signal to the block from the slave device.

Data Types: `Boolean`

sclIn — Input serial clock
`Boolean` scalar

Input serial clock, returned as a `Boolean` scalar. This port provides a serial clock signal to the block from the slave device.

Data Types: `Boolean`

Output

scl — Output serial clock
`Boolean` scalar

Output serial clock, specified as a `Boolean` scalar. This port provides a serial clock signal from the block to the slave device.

Data Types: `Boolean`

sda — Output serial data
`Boolean` scalar

Output serial data, specified as a `Boolean` scalar. This port provides a serial data signal from the block to the slave device.

Data Types: `Boolean`

rdy — Ready signal
`Boolean` scalar

Ready signal, returned as a `Boolean` scalar. When this value is `1` (true), the block is ready to accept the configuration data. When this value is `0` (false), the block is not ready to accept the configuration data.

Data Types: `Boolean`

respData — Response data register
scalar

Response data register containing the data from the slave-device register, returned as a scalar.

Data Types: `uint32`

validData — Indication of valid response data
`Boolean` scalar

Control signal that indicates if the response data is valid, returned as a `Boolean` scalar. When this value is `1` (true), the response data from response data registers is valid. When this value is `0` (false), the response data from response data registers is not valid.

Data Types: `Boolean`

statusReg — I2C bus status indicator
scalar

Indicates the status of the I2C bus, returned as a scalar.

Bit	Purpose	Value Description
[7:4]	Reserved	Reserved
3	Indicates the status of the I2C bus.	When this value is `1`, it indicates that the I2C bus is busy. When this value is `0`, it indicates that the I2C bus is idle and ready for configuration.
2	Indicates the acknowledgment status from the slave device to the I2C Master.	When this value is `1`, it indicates that the slave device has not acknowledged the I2C Master. When this value is `0`, it indicates that the slave device has acknowledged the I2C Master.
[1:0]	Reserved	Reserved

Data Types: `uint32`

Parameters

Speed — Speed-mode selection

`Standard Mode` (default) | `Fast Mode` | `Fast Plus Mode`

Specify the speed mode as one of these values:

- `Standard Mode` — Supports frequencies up to 100 KHz
- `Fast Mode` — Supports frequencies up to 400 KHz
- `Fast Plus Mode` — Supports frequencies up to 1 MHz

Version History

Introduced in R2019a

R2020a: I2C Master block has one data register input port and one data register output port

Behavior changed in R2020a

In R2019a, the I2C Master block has input data register ports **dataReg**, **dataReg1**, **dataReg2**, and **dataReg3** and output response data register ports **respData**, **respData1**, **respData2**, and **respData3**. These data register ports support a maximum of 16 bytes per transaction. In R2020a, these register ports are replaced with single data register ports: input port **dataReg** and output port **respData**. Each of these single data register ports support a maximum of 256 bytes per transaction.

In R2020a, Simulink® errors if you open a model that was created in an earlier release and that contains an I2C Master block. In this case, connections to ports **dataReg1**, **dataReg2**, **dataReg3**, **respData1**, **respData2**, and **respData3** are either missing or reconnected to empty ports on the block. Manually check and update the port connections in your model to proceed further.

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

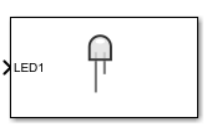
To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

LED

Connect signals attached to LEDs on hardware board



Libraries:
SoC Blockset / Hardware Logic I/O

Description

The LED block indicates the status of a signal. The hardware logic signals connected to an LED block are equivalent to the signals connected to the light emitting diodes (LED) on the hardware board.

Ports

Input

LED*x* — Input signal
`Boolean` scalar

Input signal from the hardware logic. Each LED has a port, named **LED1** to **LED*x***, where *x* is **Number of LEDs**.

Data Types: `Boolean`

Parameters

Hardware board — View selected hardware

None (default) | Supported Xilinx or Intel boards | Custom boards

This parameter is read-only. To choose a hardware board and configure board parameters, see “Hardware Implementation Pane” on page 2-2.

View LEDs location — View LEDs

button

To view a diagram of the location of the LEDs on the selected hardware board, click the **View LEDs location** button.

This button is enabled only when you select specific Xilinx or Intel boards. For more information about these boards, refer to “Supported Third-Party Tools and Hardware”.

IO logic — IO logic indicator

None (default) | Active High | Active Low

This parameter is read-only. Indicates the IO logic level on the selected hardware board.

When the **IO logic** parameter is shown as **Active Low**, the LED block accepts active low signals and represents the port names prefixed with letter *n*. For example, **nLED1**.

Number of LEDs — Number of LED ports

1 (default) | list of integers in the range [1, *n*]

Specify the required number of LED ports by specifying a value from this list. *n* is the number of available LEDs on the specified hardware board. For example, if you set this parameter to 4, the block shows four LED ports.

To use only the *n*th LED, set the **Number of LEDs** parameter to *n* and leave the unused LED ports unconnected.

For the Zynq® UltraScale+™ RFSoC ZCU216 Evaluation Kit, the first three LED ports represent the first set of red, green, and blue (RGB) LEDs out of the eight available sets, the next three LED ports represent the second set of RGB LEDs, and so on. For example, if you set the **Number of LEDs** parameter to 9, the **LED1**, **LED4**, and **LED7** ports connect to the red LEDs, the **LED2**, **LED5**, and **LED8** ports connect to the green LEDs, and the **LED3**, **LED6**, and **LED9** ports connect to the blue LEDs on the ZCU216 evaluation board.

Version History

Introduced in R2019a

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion

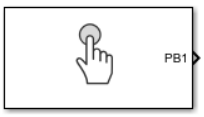
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

DIP Switch | Push Button

Push Button

Connect signals attached to push buttons on hardware board



Libraries:
SoC Blockset / Hardware Logic I/O

Description

The Push Button block controls the hardware mechanism. The hardware logic signals connected to a Push Button block are equivalent to the signals connected to the push buttons on the hardware board.

Ports

Input

PBIn*x* — Input signal
`Boolean` scalar

Input signal to control the hardware logic. Using these ports, you can dynamically control the hardware logic during simulation at run time. Each push button has a port, named **PBIn1** to **PBIn*x***, where *x* is **Number of push buttons**.

Dependencies

To enable this port, set the **Specify push buttons via** parameter to `InputPort`.

Data Types: `Boolean`

Output

PB*x* — Output signal
`Boolean` scalar

Output signal that returns the state of the push button. Each push button has a port, named **PB1** to **PB*x***, where *x* is **Number of push buttons**.

Data Types: `Boolean`

Parameters

Hardware board — View selected hardware

None (default) | Supported Xilinx or Intel boards | Custom boards

This parameter is read-only. To choose a hardware board and configure board parameters, see “Hardware Implementation Pane” on page 2-2.

View push buttons location — View push buttons

button

To view a diagram of the location of the push buttons on the selected hardware board, click the **View push buttons location** button.

This button is enabled only when you select specific Xilinx or Intel boards. For more information about these boards, refer to “Supported Third-Party Tools and Hardware”.

IO logic — IO logic indicator

None (default) | Active High | Active Low

This parameter is read-only. Indicates the IO logic level on the selected hardware board.

When the **IO logic** parameter is shown as **Active Low**, the Push Button block accepts and outputs active low signals when you set the **Specify push buttons via** parameter to `InputPort` and outputs active low signals when you set the **Specify push buttons via** parameter to `Dialog`. The block represents these port names prefixed with letter *n*. For example, **nPB1**.

Specify push buttons via — Push-button source

`Dialog` (default) | `InputPort`

To control the hardware logic by using the block parameters, select `Dialog`. To control the hardware logic from the input port, select `InputPort`.

Number of push buttons — Push-button selection

1 (default) | list of integers in the range [1, n]

To specify the required number of push-button ports, select a value from the **Number of push buttons** list. *n* represents the number of available push buttons on the specified hardware board. For example, if you select 3 from the list, the block shows three push-button ports.

To use only the *n*th push button, set the **Number of push buttons** parameter to *n* and terminate the unused push button ports.

PB*n* — Selected push buttons
`Off` (default) | `On`

To enable the *n*th push-button port, select `On` for the PB*n* parameter. *n* represents the number of available push buttons on the specified hardware board.

Dependencies

To enable this parameter, set the **Specify push buttons via** parameter to `Dialog`.

Sample time — Sampling interval

-1 (default) | positive scalar

Specify the time interval a push button toggles between `On` and `Off`.

Version History

Introduced in R2019a

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion

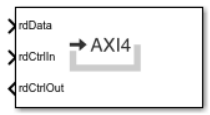
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

DIP Switch | LED

AXI4 Master Sink

Receive random access memory data



Libraries:
SoC Blockset / Hardware Logic Testbench

Description

The AXI4 Master Sink block receives random access memory data from AXI4-based data interface blocks. You can use this block as a test sink block for simulating AXI4-based data applications.

The block accepts data along with a control bus and outputs a control bus.

Ports

Input

rdData — Input data
scalar | vector

Input data from the data source. This value must be a scalar or vector.

Before reading the data, set the required data type. To set the data type, see the **Data type** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

rdCtrlIn — Input control bus
bus

Input control bus from the data producer, specified as a bus. This control bus comprises these control signals:

- rd_aready — Indicates the data source accepted the read request
- rd_dvalid — Indicates the data returned for the read request is valid

Data Types: `ReadControlS2MBusObj`

Output

rdCtrlOut — Output control bus
bus

Output control bus to the data source indicating the block is ready to accept data, returned as a scalar. This control bus comprises these control signals:

- rd_addr — Starting address for the read transaction that is sampled at the first cycle of the transaction

- rd_len — Number of data values you want to read, sampled at the first cycle of the transaction
- rd_avalid — Control signal that specifies whether the read request is valid
- rd_dready — Control signal that indicates when the block can read data

Data Types: `ReadControlM2SBusObj`

Parameters

Data type — Input data type

`uint8` (default) | `double` | `single` | `int8` | `int16` | `int32` | `int64` | `uint16` | `uint32` | `uint64` | `fixed point`

Select the data type format for the input AXI data.

Click the >> button to display the **Data Type Assistant**, which helps you to set the data type for the **rdData** input port. For details, see “Specify Data Types Using Data Type Assistant”.

Dimensions — Input data dimensions

1 (default) | positive integer | array

Specify the dimensions of the input data as a positive scalar or an array. This value defines the size of the data signal.

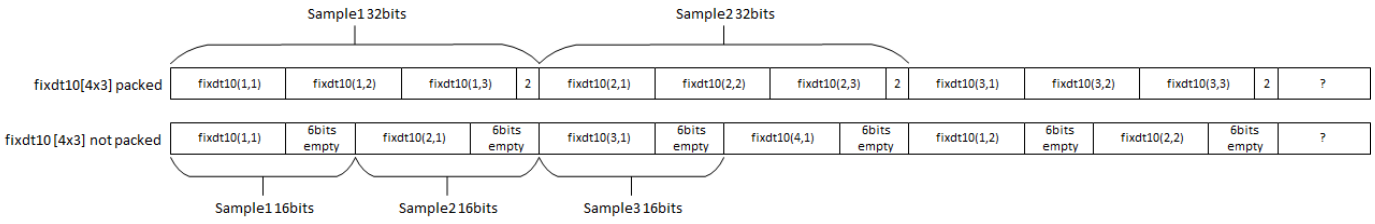
Example: `1` specifies a scalar sample.

Example: `[10 1]` specifies a vector of ten scalars.

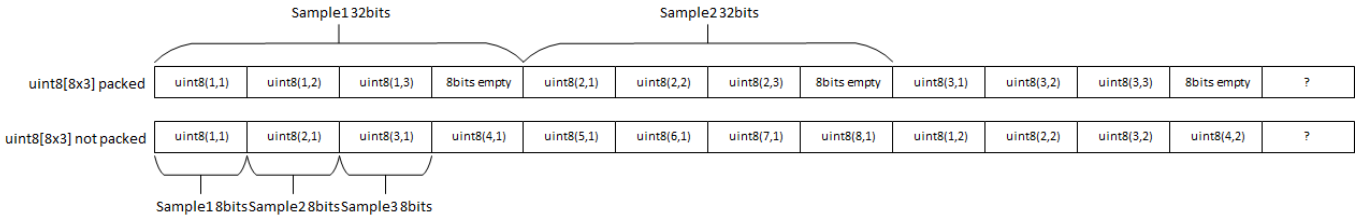
Enable sample packing (last signal dimension as channel) — Pack data on the last dimension of the signal
`off` (default) | `on`

Select this parameter to enable data packing across the last dimension of the signal. The Memory Channel block packs the data along the last dimension of the signal. For example, if the channel data type is `uint32`, the dimensions are `[1024 4]`, and if you select this parameter, then the memory channel generates 1024 read or write transactions of 128 bits. For this example, if you clear this sample packing parameter, the memory channel generates 4096 transactions of 32 bits each.

This figure shows how data is aligned for a signal with data type `fixdt10[4x3]`. When the data is packed, three 10-bit words are concatenated and extended by 2 bits to a 32-bit sample. When the data is not packed, each 10-bit word is extended to a 16-bit sample.



This figure shows how data is aligned for a signal with data type `uint8[8x3]`. When the data is packed, three 8-bit words are concatenated and extended by 8 bits to a 32-bit sample. When the data is not packed, each 8-bit word is represented as an 8-bit sample.



The combined width of the flattened signal must not exceed 512 bits.

Number of transfers — Number of read requests to send
`1` (default) | positive integer

Specify the number of read requests for the block to send.

Initial address — Start address

`0` (default) | nonnegative scalar integer

Specify the address from which the block reads the data. This value must be a nonnegative integer.

Initial delay — Initial delay

`0` (default) | nonnegative scalar

Specify the initial time after which the read operation starts.

Sample time — Time interval of sampling

`1` (default) | positive scalar | vector

Specify a time interval in seconds to define how often the block updates.

Specify the **Sample time** parameter as a scalar when you do not want the output to have a time offset. To add a time offset to the output, specify the **Sample time** parameter as a 1-by-2 vector where the first element is the sampling period and the second element is the offset. For more information about sample times in Simulink, see “Specify Sample Time”.

Save data in workspace — Save to workspace

`off` (default) | `on`

Select this parameter to save the input data to the MATLAB® workspace.

Variable name — Workspace variable name

`simOut` (default) | any MATLAB-supported variable name

Specify the workspace variable to which input data is saved. This parameter can be any MATLAB-supported variable name.

Dependencies

To enable this parameter, select the **Save data in workspace** parameter.

Version History

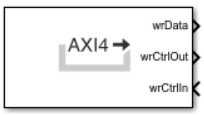
Introduced in R2019a

See Also

AXI4 Master Source

AXI4 Master Source

Generate random access memory data



Libraries:
SoC Blockset / Hardware Logic Testbench

Description

The AXI4 Master Source block generates random access memory data to AXI4-based data interface blocks. You can use this block as a test source block for simulating AXI4-based data applications.

The block accepts a control bus and outputs data along with a control bus.

Ports

Input

wrCtrlIn — Input control bus
bus

Control bus from the data consumer signaling that data consumer is ready to accept data, specified as a scalar. This control bus comprises these control signals:

- wr_ready — Indicates the block can send data to the data consumer
- wr_complete — Indicates the write transaction has completed at the data consumer
- wr_bvalid — Indicates the data consumer has accepted the transaction

Data Types: `WriteControlS2MBusObj`

Output

wrData — Output AXI data
scalar | vector

Output AXI data to the data consumer. This value is returned as a scalar or vector.

You can change the data type of the output data. For more information, see the **Data type** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

wrCtrlOut — Output control bus
bus

Control bus to the data consumer, returned as a bus. This control bus comprises these control signals:

- wr_addr — Specifies the starting address that the block writes

- wr_len — Specifies the number of data elements in the write transaction
- wr_valid — Indicates the data sampled at the **wrData** output port is valid

Data Types: `WriteControlM2SBusObj`

Parameters

Data type — Output data type

`uint8` (default) | `double` | `single` | `int8` | `int16` | `int32` | `int64` | `uint16` | `uint32` | `uint64` | `fixed point`

Select the data type format for the output AXI data.

Click the `>>` button to display the **Data Type Assistant**, which helps you to set the data type for the **wrData** output port. For details, see “Specify Data Types Using Data Type Assistant”.

Dimensions — Output data dimensions

`1` | positive scalar | array

Specify the dimensions of the output data as a positive scalar or an array. This value defines the size of the data signal.

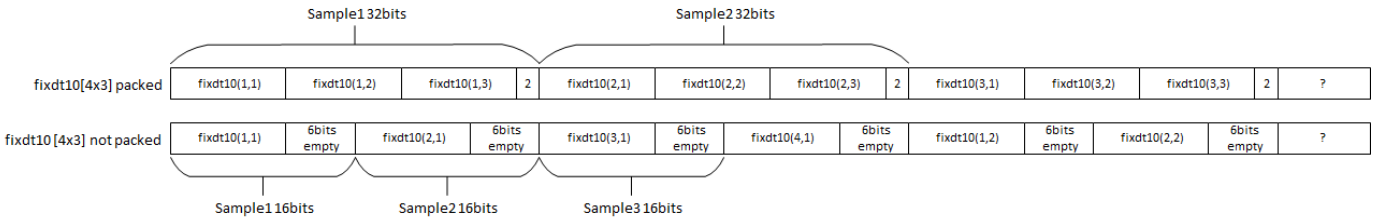
Example: `1` specifies a scalar sample.

Example: `[10 1]` specifies a vector of ten scalars.

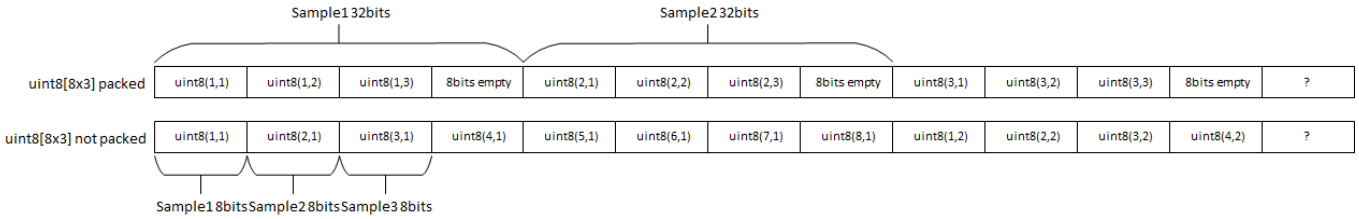
Enable sample packing (last signal dimension as channel) — Pack data on the last dimension of the signal
`off` (default) | `on`

Select this parameter to enable data packing across the last dimension of the signal. The Memory Channel block packs the data along the last dimension of the signal. For example, if the channel data type is `uint32`, the dimensions are `[1024 4]`, and if you select this parameter, then the memory channel generates 1024 read or write transactions of 128 bits. For this example, if you clear this sample packing parameter, the memory channel generates 4096 transactions of 32 bits each.

This figure shows how data is aligned for a signal with data type `fixdt10[4x3]`. When the data is packed, three 10-bit words are concatenated and extended by 2 bits to a 32-bit sample. When the data is not packed, each 10-bit word is extended to a 16-bit sample.



This figure shows how data is aligned for a signal with data type `uint8[8x3]`. When the data is packed, three 8-bit words are concatenated and extended by 8 bits to a 32-bit sample. When the data is not packed, each 8-bit word is represented as an 8-bit sample.



The combined width of the flattened signal must not exceed 512 bits.

Number of transfers — Number of write requests to send
`1` (default) | positive integer

Specify the number of write requests for the block to send.

Initial address — Start address

`0` (default) | nonnegative integer

Specify the address to which the block writes the data. This value must be a nonnegative integer.

Initial delay — Initial delay

`0` (default) | nonnegative scalar

Specify the initial time after which the write operation starts. This value must be a nonnegative scalar.

Data generation — Output generation type

`counter` (default) | `random` | `ones` | `workspace`

Specify the generation type for the output as one of these values:

- `counter` — Generate data from a counter, based on the selected data type.
- `random` — Generate random data.
- `ones` — Generate data with all the bits as ones, based on the selected data type.
- `workspace` — Generate data from the MATLAB workspace.

Counter init value — Initial counter value

`0` (default) | scalar

Specify the value from which the counter starts. The valid range of counter values depends on the selected value for the **Data type** parameter. If this value is out of the valid range, it is rounded off to the nearest valid value.

For example, if **Data type** is `uint8` and this value is `6.787`, this value is rounded to `7`.

Dependencies

To enable this parameter, set the **Data generation** parameter to `counter`.

Variable name — Workspace variable name

`simOut` (default) | any MATLAB-supported variable name

Specify the workspace variable from which output data is generated. This parameter can be any MATLAB-supported variable name.

Note The workspace variable must be a numerical array.

Dependencies

To enable this parameter, set the **Data generation** parameter to `workspace`.

Sample time — Time interval of sampling

1 (default) | positive scalar | vector

Specify a time interval in seconds to define how often the block updates.

Specify the **Sample time** parameter as a scalar when you do not want the output to have a time offset. To add a time offset to the output, specify the **Sample time** parameter as a 1-by-2 vector where the first element is the sampling period and the second element is the offset. For more information about sample times in Simulink, see “Specify Sample Time”.

Version History

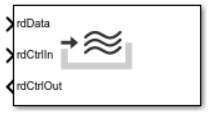
Introduced in R2019a

See Also

AXI4 Master Sink

Stream Data Sink

Receive continuous stream data



Libraries:
SoC Blockset / Hardware Logic Testbench

Description

The Stream Data Sink block receives continuous stream data from advanced extensible interface AXI4-based stream data interface blocks. You can use this block as a test sink block for simulating AXI4-based stream data applications.

The block accepts stream data along with a control bus and outputs a control bus.

Ports

Input

rdData — Input stream data
scalar | vector

Input stream data from the data source. This value must be a scalar or vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

rdCtrlIn — Input control bus
bus

Input control bus from the data source. This control bus comprises the following control signals:

- valid — Indicates the input stream data on the **rdData** input port is valid
- tlast — Indicates the end of the data transaction

Data Types: `StreamM2SBusObj`

Output

rdCtrlOut — Output control bus
bus

Output control bus to the data source, indicating that the block is ready to accept stream data. This control bus comprises a ready signal.

Data Types: `StreamS2MBusObj`

Parameters

Save data in workspace — Save data in workspace

`off` (default) | `on`

Select this parameter to save the input stream data to the MATLAB workspace.

Variable name — Workspace variable name

`simOut` (default) | any MATLAB-supported variable name

Specify the workspace variable to which input stream data is saved. This parameter can be any MATLAB-supported variable name.

Dependencies

To enable this parameter, select the **Save data in workspace** parameter.

Version History

Introduced in R2019a

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion

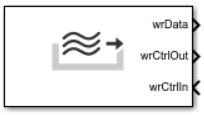
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Stream Data Source

Stream Data Source

Generate continuous stream data



Libraries:
SoC Blockset / Hardware Logic Testbench

Description

The Stream Data Source block generates stream data to advanced extensible interface AXI4-based stream data interface blocks. You can use this block as a test source block for simulating AXI4-based stream data applications.

The block accepts a control bus and outputs stream data along with a control bus.

Ports

Input

wrCtrlIn — Input control bus
bus

Control bus from the data consumer signaling that data consumer is ready to accept stream data. This control bus comprises a ready signal.

Data Types: `StreamS2MBusObj`

Output

wrData — Output stream data
scalar | vector

Output stream data to the data consumer. This value is returned as a scalar or vector.

You can change the data type of the output stream data. For more information, see the **Data type** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `fixed point`

wrCtrlOut — Output control bus
bus

Control bus to the data consumer, returned as a bus. This control bus comprises these control signals:

- valid — Indicates the output data on the **wrData** output port is valid
- tlast — Indicates the end of the data transaction

Data Types: `StreamM2SBusObj`

Parameters

Data type — Output data type

`uint8` (default) | `double` | `single` | `int8` | `int16` | `int32` | `int64` | `uint16` | `uint32` | `uint64` | `fixdt(1,16,0)`

Select the data type format for the output stream data.

Click the >> button to display the **Data Type Assistant**, which helps you to set the data type for the **wrData** output port. For details, see “Specify Data Types Using Data Type Assistant”.

Dimensions — Output data dimensions

`10` (default) | positive integer | array

Specify the dimensions of the output stream data as a positive scalar or an array.

Example: `1` specifies a scalar sample.

Example: `[10 1]` specifies a vector of ten scalars.

Burst length — Length of single burst

`20` (default) | positive integer

Length of the single burst, specified as a positive integer.

Total bursts — Total number of bursts

`4` (default) | positive integer

Total number of bursts generated from the block, specified as a positive integer.

Data generation — Output generation type

`counter` (default) | `random` | `ones` | `workspace`

Specify the generation type for the output as one of these values:

- `counter` — Generate data from a counter, based on the selected data type.
- `random` — Generate a random data.
- `ones` — Generate data with all the bits as ones, based on the selected data type.
- `workspace` — Generate data from the MATLAB workspace.

Counter init value — Initial counter value

`0` (default) | scalar

Specify the value from which the counter starts. The valid range of counter values depends on the selected value for the **Data type** parameter. If this value is out of the valid range, it is rounded off to the nearest valid value.

For example, if **Data type** is `uint8` and this value is `6.787`, this value is rounded to `7`.

Dependencies

To enable this parameter, set the **Data generation** parameter to `counter`.

Variable name — Workspace variable name

`simOut` (default) | any MATLAB supported variable name

Specify the variable name from which output stream data is generated. This parameter can be any MATLAB-supported variable name.

Note The workspace variable must be a numerical array.

Dependencies

To enable this parameter, set the **Data generation** parameter to `workspace`.

Sample time — Time interval of sampling

1 (default) | positive scalar | vector

Specify a time interval in seconds to define how often the block updates.

Specify the **Sample time** parameter as a scalar when you do not want the output to have a time offset. To add a time offset to the output, specify the **Sample time** parameter as a 1-by-2 vector where the first element is the sampling period and the second element is the offset. For more information about sample times in Simulink, see “Specify Sample Time”.

Transfer delay (in samples) — Delay between bursts

`0` (default) | nonnegative integer

Time after which the next burst occurs. This value must be a nonnegative integer.

Version History

Introduced in R2019a

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion

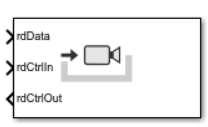
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Stream Data Sink

Video Test Sink

Receive continuous video stream data



Libraries:
SoC Blockset / Hardware Logic Testbench

Description

The Video Test Sink block receives continuous video stream data from advanced extensible interface AXI4-based video stream data interface blocks. You can use this block as a test sink block for simulating AXI4-based video stream data applications.

The block accepts video stream data along with a control bus and outputs a control bus.

Ports

Input

rdData — Input video stream data
vector

Input video stream data from the data source. This value must be a vector.

Data Types: `uint8`

rdCtrlIn — Input control bus
bus

Input control bus from the data source, specified as a bus. This control bus comprises these signals:

- hStart — First pixel in a horizontal line of a frame
- hEnd — Last pixel in a horizontal line of a frame
- vStart — First pixel in the first (top) line of a frame
- vEnd — Last pixel in the last (bottom) line of a frame
- valid — Indicates the input pixel data on **rdData** input port is valid

Data Types: `pixelcontrol`

Output

rdCtrlOut — Output control bus
bus

Output control bus to the data source signaling that the block is ready to accept video stream data. This control bus comprises a ready signal.

Data Types: `StreamVideoS2MBusObj`

Parameters

Frame size — Frame dimensions

`160x120p` (default) | ...

Select the frame dimensions as one of these values:

- `576p SDTV (720x576p)`
- `720p HDTV (1280x720p)`
- `1080p HDTV (1920x1080p)`
- `160x120p`
- `320x240p`
- `640x480p`
- `800x600p`
- `1024x768p`
- `1280x768p`
- `1280x1024p`
- `1360x768p`
- `1366x768p`
- `1400x1050p`
- `1600x1200p`
- `1680x1050p`
- `1920x1200p`

The **Frame size** value must be same as that of the data source.

Color space — Color space

`YCbCr422` (default) | `RGB` | `YOnly`

Select the type of color space as `YCbCr422`, `RGB`, or `YOnly`. The **Color space** value must be same as that of the data source.

Reorder input frame — Input frame reorder

`off` (default) | `on`

Select this option to reorder input pixels. Streaming pixel format order is from left to right across each line, then down to the next line, or *row-major*. However, some matrix operations use *column-major* order, that is, from top to bottom and then right to the next column. Depending on how your design has manipulated the pixels, you may need to reorder them for correct display.

Save data in workspace — Save data in workspace

`off` (default) | `on`

Select this parameter to save the input video stream data to the MATLAB workspace.

Variable name — Workspace variable name

`simOut` (default) | any MATLAB-supported variable name

Specify the workspace to which input video stream data is saved. This parameter can be any MATLAB-supported variable name.

Dependencies

To enable this parameter, select the **Save data in workspace** parameter.

View input — Display input in MATLAB

`off` (default) | `on`

Select this parameter to view the input video stream data in the MATLAB viewer.

Version History

Introduced in R2019a

Extended Capabilities

Fixed-Point Conversion
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Video Test Source

Video Test Source

Generate continuous video stream data

Libraries:
SoC Blockset / Hardware Logic Testbench

Description

The Video Test Source block generates continuous video stream data to advanced extensible interface AXI4-based video stream data interface blocks. You can use this block as a test source block for simulating AXI4-based video stream data applications.

The block accepts a control bus and outputs video stream data along with a control bus.

Ports

Input

wrCtrlIn — Input control bus
bus

Control bus from the data consumer signaling that data consumer is ready to accept video stream data. This control bus comprises a ready signal.

Data Types: `StreamVideoS2MBusObj`

Output

wrData — Output video stream data
vector

Output video stream data to the data consumer. This value is returned as a vector.

Data Types: `uint8`

wrCtrlOut — Output control bus
bus

Control bus to the data consumer, returned as a bus. This control bus comprises these control signals:

- hStart — First pixel in a horizontal line of a frame
- hEnd — Last pixel in a horizontal line of a frame
- vStart — First pixel in the first (top) line of a frame
- vEnd — Last pixel in the last (bottom) line of a frame
- valid — Indicates the output pixel data on the **wrData** output port is valid

Data Types: `pixelcontrol`

Parameters

Frame size — Frame dimensions

`160x120p` (default) | ...

Select the frame dimensions as one of these values:

- `480p SDTV (720x480p)`
- `576p SDTV (720x576p)`
- `720p HDTV (1280x720p)`
- `1080p HDTV (1920x1080p)`
- `160x120p`
- `320x240p`
- `640x480p`
- `800x600p`
- `1024x768p`
- `1280x768p`
- `1280x1024p`
- `1360x768p`
- `1366x768p`
- `1400x1050p`
- `1600x1200p`
- `1680x1050p`
- `1920x1200p`

Video source — Select video source type

`Video file` (default) | `Color bar` | `Ramp`

Select the type of video source as `Video file`, `Color bar`, or `Ramp`.

Input file name — Select input video file

`handshake_left.avi` (default) | any supported video file format

Select the input video file by clicking the **Browse** button and navigating to the video file location.

Dependencies

To enable this parameter, set the **Video source** parameter to `Video file`.

Color space — Color space

`YCbCr422` (default) | `RGB` | `YOnly`

Select the type of color space as `YCbCr422`, `RGB`, or `YOnly`.

Reorder output frame — Output frame reordering

`off` (default) | `on`

Select this option to reorder output pixels to *column-major* order. Streaming pixel format order is from left to right across each line, then down to the next line, or *row-major*. However, some matrix operations use *column-major* order, that is, from top to bottom and then right to the next column.

Frame sample time — Frame sample time

`1/60` (default) | positive scalar

Specify the frame sample time as a positive scalar. The denominator in default value `1/60` denotes the number of frames per second.

Version History

Introduced in R2019a

Extended Capabilities

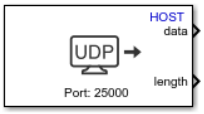
Fixed-Point Conversion
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Video Test Sink

UDP Read (HOST)

Receive UDP packets on local host computer from remote host



Libraries:
SoC Blockset / Host I/O

Description

The UDP Read (HOST) block receives UDP (User Datagram Protocol) packets from remote host on the local host. The local host in the host computer on which you want to receive UDP packets. The remote host is the host computer or hardware from which you want to receive UDP packets.

Ports

Output

data — UDP packet received from remote host
numeric vector

UDP packet received on local host computer, returned as a numeric vector. The **Data type for Message** and **Length** parameters set this output data type and packet length, respectively.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

length — Length of UDP packet
nonnegative scalar

Length of UDP packet returned on the **data** port.

This port is unnamed until you clear the **Output variable-size signal** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Parameters

Local IP port — IP port number of local host
`25000` (default) | integer from 1 to 65,535

Specify the IP port number of local host.

Note On Linux®, to set the local IP port number to a value less than 1024, run MATLAB with root privileges. For example, at the Linux command line, enter:

```
sudo matlab
```

Remote IP address ('0.0.0.0' to accept all) — IP address of remote host
`'0.0.0.0'` (default) | dotted-quad expression

Specify the IP address of the remote host. Set this value to a specific IP address, to block UDP packets from all other IP addresses. To accept UDP packets from all IP addresses, use the default value `'0.0.0.0'`.

Receive buffer size (bytes) — Maximum number of data bytes in received data
`8192` (default) | positive integer

Specify the maximum number of data bytes of UDP packets you want to store in the local buffer. Set this value large enough to avoid data loss caused by buffer overflows.

Maximum length for Message — Maximum length of data
`255` (default) | positive integer scalar

Specify the maximum length of the output UDP packet. Set this parameter to a value equal to or greater than the data size of the UDP packet. The block truncates any data that exceeds this length.

The maximum payload size of a UDP packet is 65,507 bytes. The **Maximum length for Message** is equal to the maximum payload size of a UDP packet in bytes divided by the data type size of the UDP packet. For example, if the output data type is `double`, then set **Maximum length for Message** value to 65507/8 = 8118.

Data type for Message — Data type of output UDP packet
`uint8` (default) | `single` | `double` | `int8` | `int16` | `uint16` | `int32` | `uint32` | `boolean`

Select the data type for the vector elements of output UDP packet. Match this data type with the data type of the UDP packets sent by the remote host.

Blocking time (seconds) — Time to wait for UDP packet
`0` (default) | nonnegative scalar

Specify the duration of time to wait for a UDP packet before returning control to the scheduler for each sample.

Sample time (seconds) — Sample time
`0.01` (default) | nonnegative scalar

Specify how often the scheduler runs this block.

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

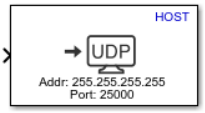
Generate C and C++ code using Simulink® Coder™.

See Also

UDP Write (HOST)

UDP Write (HOST)

Send UDP packets from host computer to remote host



Libraries:
SoC Blockset / Host I/O

Description

The UDP Write (HOST) block sends UDP (User Datagram Protocol) packets from a local host to a remote host. The local host in the host computer from which you want to send UDP packets. The remote host is the host computer or hardware to which you want to send UDP packets. The remote host is identified by the remote IP address and remote IP port parameters from host computer.

Ports

Input

Port_1 — Input signal
numeric vector

Input signal, specified as a numeric vector. The block sends this data as a UDP packets to the specified remote IP address and port.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Parameters

Remote IP address ('255.255.255.255' for broadcast) — IP address of remote host
`'255.255.255.255'` (default) | dotted-quad expression

Specify the IP address of the remote host. To broadcast UDP packets, use the default value, `'255.255.255.255'`.

Remote IP port — IP port number of remote host
`25000` (default) | integer from 1 to 65,535

Specify the IP port number of the remote host.

Local IP port source — Source of local IP port source
`Automatically determine` (default) | `Specify via dialog`

Set the source of Local IP port for the block by selecting one of these values:

- `Automatically determine` — Assigns an available local IP port number randomly from which UDP packets are sent.
- `Specify via dialog` — Allows you to specify the local IP port number using the **Local IP port** parameter.

Local IP port — IP port number of local host
-1 (default) | integer from 1 to 65,535

Specify the port number of the local host. If this value is set to `-1` (default), the block sets the local IP port number to a random available port number and uses that port to send the UDP packets. If the remote host accepts UDP packets from a particular IP port number, specify that IP port number for this value.

Version History

Introduced in R2019a

Extended Capabilities

C/C++ Code Generation

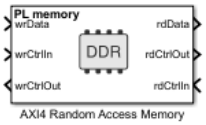
Generate C and C++ code using Simulink® Coder™.

See Also

UDP Read (HOST)

AXI4 Random Access Memory

Model random access through external memory

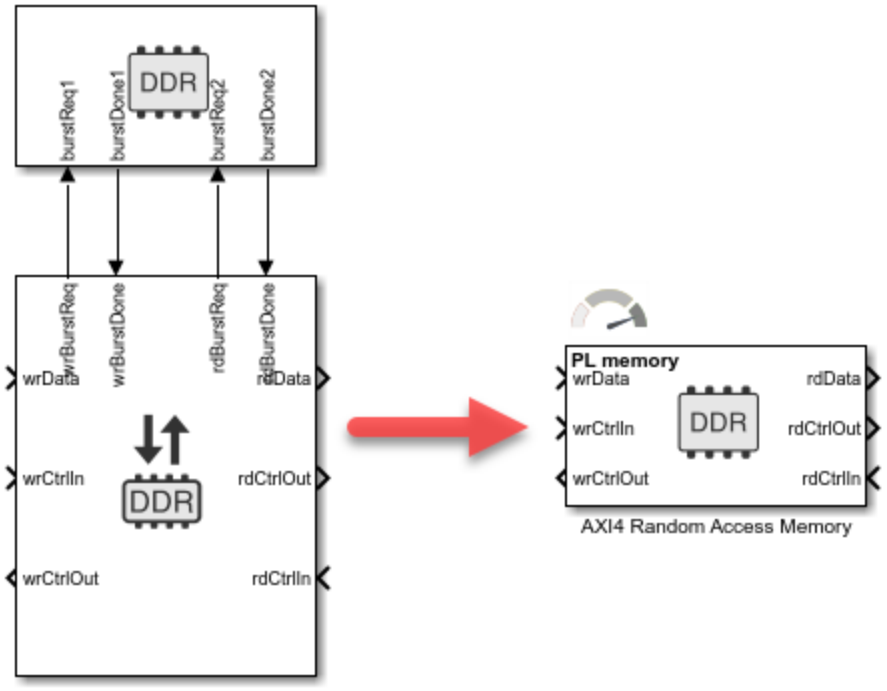


Libraries:
SoC Blockset / Memory

Description

The AXI4 Random Access Memory block models a connection between two hardware algorithms through external memory, using the MathWorks® simplified AXI4 manager protocol. Both the writer and the reader are managers, sending read and write requests to memory through the block. The external memory is unmanaged (no logical buffers, no circular buffer). To ensure the integrity of the data, the reader and writer coordinate timing on accesses.

This block is equivalent to a Memory Channel block with the **Channel type** parameter set to `AXI4 Random Access` connected to a Memory Controller block.



For more information, see “Simplified AXI4 Master Interface”.

Ports

Input

wrData — Writer data bus signal
scalar | vector | matrix

This signal contains the data to the memory.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

rdCtrlIn — Reader input control signal
bus

This port accepts a control bus from a data consumer block, signaling that the consumer block is ready to accept read data. This input is a read-request from the reader. To create the control bust, use the SoC Bus Creator block.

Data Types: `ReadControlM2SBusObj`

wrCtrlIn — Writer input control signal
bus

This port accepts a control bus from a data producer block, signaling that the producer block is ready to send data. This input is a write-request from the writer. To create the control bus, use the SoC Bus Creator block.

Data Types: `WriteControlM2SBusObj`

Output

rdData — Output data signal to data consumer
scalar | vector | matrix

This signal contains the data read from the memory.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

rdCtrlOut — Reader control signal from memory channel to data consumer
bus

This bus represents the protocol bus from the memory channel to the data consumer. To separate the signal from the bus, use the SoC Bus Selector block.

Data Types: `ReadControlS2MBusObj`

wrCtrlOut — Writer control signal from memory channel to data producer
bus

This bus represents the protocol bus from the memory channel to the data producer. To separate the signal from the bus, use the SoC Bus Selector block.

Data Types: `WriteControlS2MBusObj`

Parameters

Memory simulation — Set simulation timing accuracy
`Burst accurate` (default) | `Protocol accurate`

Memory simulation provides two levels of timing resolution. Select one of these options:

- **Burst accurate** — Simulates memory contention and high-resolution timing.
- **Protocol accurate** — Simulates AXI4 protocol hand-shaking sequencing and low-resolution timing.

Main

Memory selection — Choose between memory regions
`PS memory` (default) | `PL memory`

Select between processing subsystem (PS) or programming logic (PL) memory.

- If the selected board supports only a PL memory, then the default value is `PL memory`.
- If the selected board supports only PS memory or only PL memory, then this parameter is read-only.
- If the selected board is not a supported SoC board, then this parameter is not visible.

Region size (bytes) — Size of buffer, in bytes
4096 (default) | scalar

Specify the size of the memory buffer in bytes.

Initialize memory content — Initialize memory for simulation
`off` (default) | `on`

Select this parameter to enable presimulation memory initialization.

Initial value — Initial memory value
0 (default) | scalar | vector

Specify an initial value for memory simulation, as one of the following:

- Integer scalar value between 0–255 — The block replicates and loads this value to all memory locations.
- Vector of `uint8` — The block writes each element of the vector into a memory location.
  - If the vector is smaller than the memory region, the vector is padded with zeroes to match the size of the memory region.
  - If the vector is larger than the memory region, the vector is truncated to match the size of the memory region.

Example: `0` — Initializes all memory locations with zeros.

Example: `1:10` — Initializes the first ten memory locations with values 1–10 and the other locations with zeros.

Dependencies

To enable this parameter, select **Initialize memory**.

Log memory content — Log memory contents
`off` (default) | `on`

Select this parameter to enable postsimulation memory logging.

Log variable name — Variable name for memory log
string | character vector

Name of variable to hold the data of the memory content, specified as a string or character vector.

Example: `'mem_content'`

Dependencies

To enable this parameter, select **Log memory ending value**.

Signal Attributes

Write data signal

Dimensions — Dimensions of input data signal
scalar | array

wrData can be a multidimensional array. Specify the dimension for the array as a whole number.

Example: 1 — A scalar sample.

Example: [10 1] — A vector of ten scalars.

Example: [1080 1920 3] — A 1080p frame. The frame includes 1080 lines of 1920 pixels per line, and each pixel is represented by three values (for red, green, and blue).

Data type — Data type of writer data
uint32 (default) | double | single | int8 | int16 | int32 | int64 | uint8 | uint16 | uint64 | boolean | fixed point

Specify the data type of the **wrData** port. For help, click the **...** button and select **Data Type Assistant**.

Sample time — Time interval of sampling
1 (default) | positive scalar | vector

Specify a time interval in seconds to define how often the block updates.

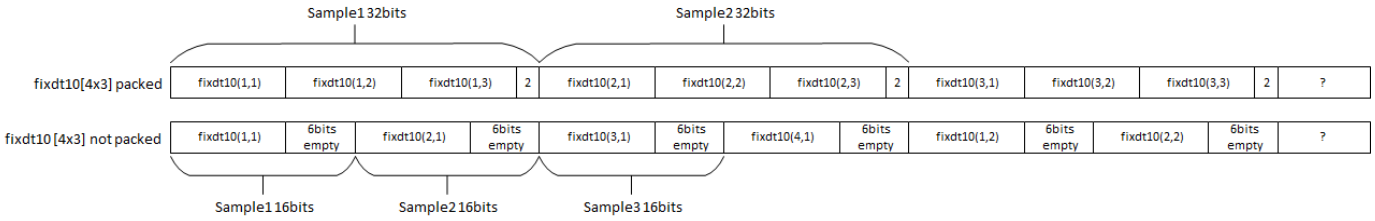
When you do not want the output to have a time offset, specify the **Sample time** parameter as a scalar. To add a time offset to the output, specify the **Sample time** parameter as a 1-by-2 vector, where the first element is the sampling period and the second element is the offset. For more information about sample times in Simulink, see “Specify Sample Time”.

Enable sample packing (last signal dimension as channel) — Pack data on the last dimension of the signal
`off` (default) | `on`

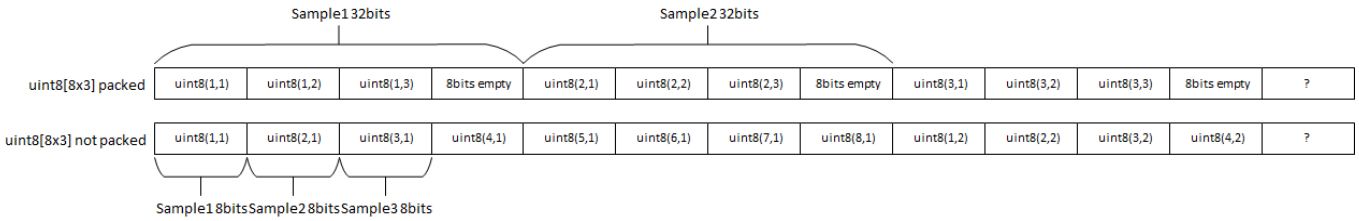
Select this parameter to enable data packing across the last dimension of the signal. The AXI4-Random Access Memory block packs the data along the last dimension of the signal. For example, if the channel data type is `uint32`, the dimensions are `[1024 4]`. If you select this sample packing parameter, then the memory channel generates 1024 read or write transactions of 128 bits. If you

clear this sample packing parameter, the memory channel generates 4096 transactions of 32 bits each.

This figure shows how data is aligned for a signal with data type `fixdt10[4x3]`. When the data is packed, three 10-bit words are concatenated and extended by 2 bits to a 32-bit word. When the data is not packed, each 10-bit word is extended to a 16-bit word.



This figure shows how data is aligned for a signal with data type `uint8[8x3]`. When the data is packed, three 8-bit words are concatenated and extended by 8 bits to a 32-bit word. When the data is not packed, each 8-bit word is represented as an 8-bit sample.



The combined width of the flattened signal must not exceed 512 bits.

Read data signal

Output data signal matches input — Reader and writer use the same values
on (default) | off

Select this box to use the same dimensions and data type for the reader and the writer of this channel. Clear the box to customize different dimensions and data type for the reader and writer interfaces.

Dimensions — Dimensions of output data signal
scalar | array

rdData can be a multidimensional array. Specify the dimension for the array as a whole number.

Example: 1 — A scalar sample.

Example: [10 1] — A vector of ten scalars.

Example: [1080 1920 3] — A 1080p frame. The frame includes 1080 lines of 1920 pixels per line, and each pixel is represented by three values (for red, green, and blue).

Data type — Data type of reader data
uint32 (default) | double | single | int8 | int16 | int32 | int64 | uint8 | uint16 | uint64 | boolean | fixed point

Specify the data type of the **rdData** port. For help, click the **...** button and select **Data Type Assistant**.

Dependencies

To enable this parameter, clear the **Output data signal matches input** check box.

Sample time — Time interval of sampling
1 (default) | positive scalar | vector

Specify a time interval in seconds to define how often the block updates.

When you do not want the output to have a time offset, specify the **Sample time** parameter as a scalar. To add a time offset to the output, specify the **Sample time** parameter as a 1-by-2 vector, where the first element is the sampling period and the second element is the offset. For more information about sample times in Simulink, see “Specify Sample Time”.

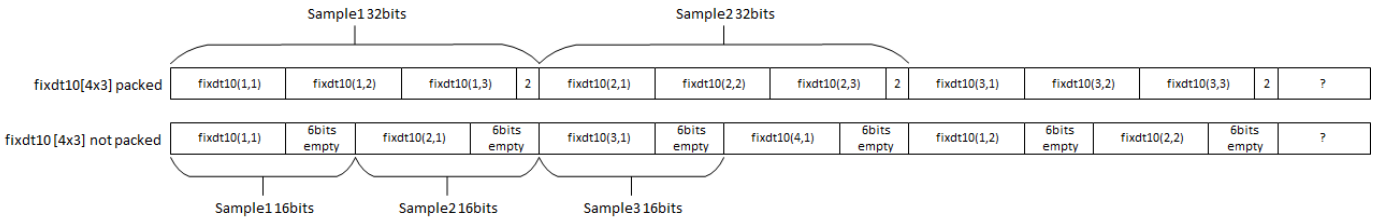
Dependencies

To enable this parameter, clear the **Output data signal matches input** check box.

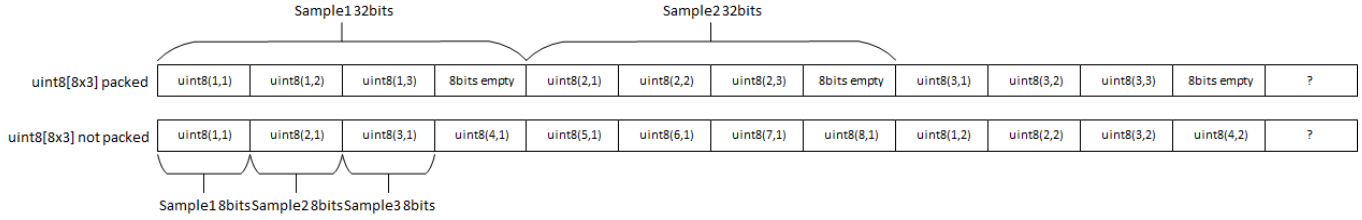
Enable sample packing (last signal dimension as channel) — Pack data on last dimension of the signal
`off` (default) | `on`

Select this parameter to enable data packing across the last dimension of the signal. The AXI4 Random Access Memory block packs the data along the last dimension of the signal. For example, if the channel data type is `uint32`, the dimensions are `[1024 4]`. If you select this sample packing parameter, then the memory channel generates 1024 read or write transactions of 128 bits. If you clear this sample packing parameter, the memory channel generates 4096 transactions of 32 bits each.

This figure shows how data is aligned for a signal with data type `fixdt10[4x3]`. When the data is packed, three 10-bit words are concatenated and extended by 2 bits to a 32-bit word. When the data is not packed, each 10-bit word is extended to a 16-bit word.



This figure shows how data is aligned for a signal with data type `uint8[8x3]`. When the data is packed, three 8-bit words are concatenated and extended by 8 bits to a 32-bit word. When the data is not packed, each 8-bit word is represented as an 8-bit sample.



The combined width of the flattened signal must not exceed 512 bits.

Dependencies

To enable this parameter, clear the **Output data signal matches input** check box.

Performance

View performance plots — Display performance metrics
button

Clicking the button opens the **Performance Plots for Memory Controller** window. You can then select to plot bandwidth, bursts, or latencies. For more information about performance graphs, see "Memory Controller Latency Plots".

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

Version History

Introduced in R2022b

R2023a: Support added for memory initialization and logging

You can now load an initial value before simulation or log memory content after simulation.

- To initialize the memory, open the block mask and select **Initialize memory** in the main tab. Then, specify the initial value in the **Initial value** box.
- To log the contents of the memory after simulation, open the block mask and select **Log memory ending value** in the main tab. Then, specify the initial value in the **Initial value** box.

R2023a: Buffer size and Region size parameters merged

Behavior changed in R2023a

In previous releases, you can specify the size of the memory region in the **Buffer size** box, and that was reflected as a read-only **Region size** parameter. In R2023a, specify that size in the **Region size (bytes)** parameter.

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

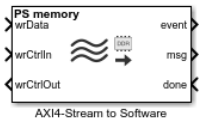
Memory Channel | AXI4-Stream to Software | Software to AXI4-Stream | AXI4 Video Frame Buffer

Topics

"External Memory Channel Protocols"

AXI4-Stream to Software

Stream AXI4 data from FPGA to software

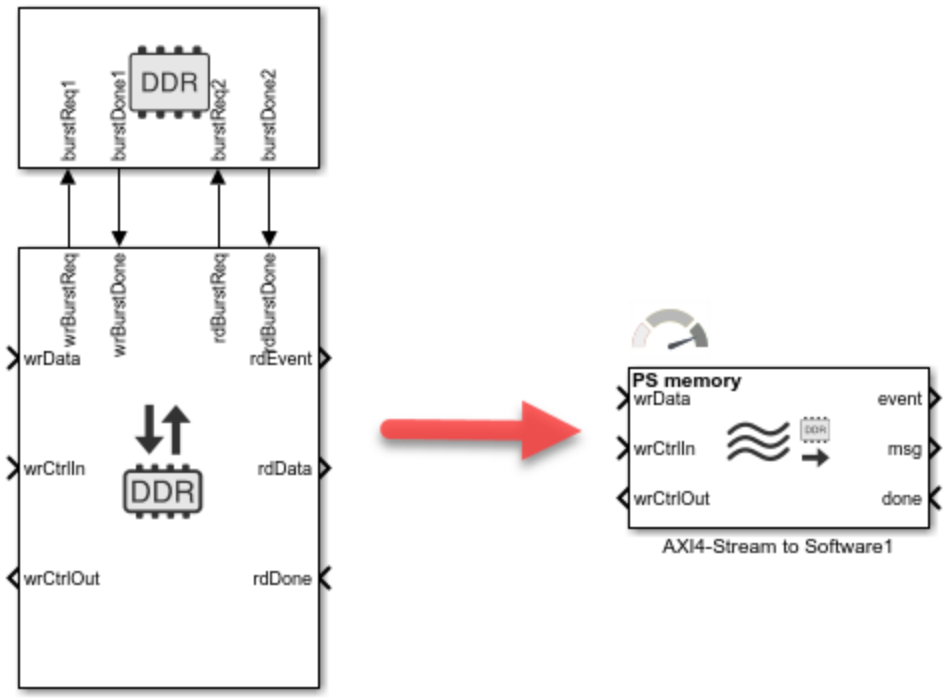


Libraries:
SoC Blockset / Memory

Description

The AXI4-Stream to Software block models a connection between hardware logic and a software task through external memory. The writer puts data into the channel using a MathWorks simplified AXI stream protocol and the reader (processor) gets data from a DMA driver interface. The block models the datapath and software stack of that connection, including a FIFO, DMA engine, interconnect and external memory, interrupts, kernel buffer management of the DMA driver, and data transfers to the software task.

This block is equivalent to a Memory Channel block with the **Channel type** parameter set to `AXI4-Stream to Software via DMA` connected to a Memory Controller block.



For more information about the MathWorks simplified AXI stream protocol, see “AXI4-Stream Interface”.

Ports

Input

wrData — Writer data bus signal
scalar | vector | matrix

This signal contains the data to the memory.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

wrCtrlIn — Writer input control signal
bus

This port represents the protocol from the data producer to the processor. The AXI4-Stream to Software block checks this signal when using **wrData**. The signals on the control bus are of type `StreamM2SBusObj`. To create this control bus, use the SoC Bus Creator block. For more information about bus types, see “External Memory Channel Protocols”.

Data Types: `StreamM2SBusObj`

done — Notification message of completed read
scalar

This message port receives a notification from the connected Stream Read block. The notification indicates that a read transaction completed. For more information on messages, see “Messages”.

Data Types: `Boolean`

Output

event — Task read event signal
scalar

This port sends a task event signal that triggers the Task Manager block to execute the associated event-driven read.

Data Types: `rteEvent`

msg — Output data signal to software
scalar | vector | matrix

This signal contains the data read from the memory, sent as a message to the connected Stream Read block. For more information on messages, see “Messages”.

Data Types: `SoCData`

wrCtrlOut — Writer control signal to data producer
bus

This bus represents the protocol bus from the memory channel to the data producer. The signals on the bus are of type `StreamS2MBusObj`. To separate the signal from the bus, use the SoC Bus Selector block.

Data Types: `StreamS2MBusObj`

Parameters

Memory simulation — Set simulation timing accuracy
`Burst accurate` (default) | `Protocol accurate` | `Behavioral`

Memory simulation provides three levels of timing resolution. Select one of these options:

- **Burst accurate** — Simulates memory contention and high-resolution timing.
- **Protocol accurate** — Simulates AXI4 protocol hand-shaking sequencing and low-resolution timing.
- **Behavioral** — Simulates data transactions only and no timing.

  When this parameter is set to **Behavioral** mode, the value of the `TLAST` signal is ignored.

Main

Memory selection — Choose between memory regions
`PS memory` (default) | `PL memory`

Select between processing subsystem (PS) or programming logic (PL) memory.

- If the selected board supports only PL memory, then the default value is `PL memory`.
- If the selected board supports only PS memory or only PL memory, then this parameter is read-only.
- If the selected board is not a supported SoC board, then this parameter is not visible.

Region size (bytes) — Size of the allocated memory region, in bytes
calculated

This parameter is read-only.

The of the region size in bytes. This value is calculated as the number of buffers multiplied by the size of the buffer. The size of the buffer is defined as the size of the data multiplied by the software frame size. Define this value by setting the **Data type** and **Dimensions** parameters on the **Signal Attributes** tab.

Example: A software frame size of 1024 uint32 defines a buffer of 4096 Bytes. If the number of buffers is set to 2, the region size is 8192 bytes.

Number of buffers — Number of buffers in region
8 (default) | integer

Define the number of buffers in the memory.

The memory access has a ring-buffer pattern. The writer can continually write as long as buffers are available. When a buffer is completed, it becomes available for the reader. The writer and reader traverse the buffers in a circular pattern. As long as the writer and reader maintain similar rates, the buffering prevents blockage.

A disparate rate between a reader and a writer slows down the faster device. For example, a slow reader causes the writer to run out of buffers and blocks the writer, effectively slowing down the writer to the reader rate. Likewise, a slow writer causes the reader to run out of buffers and blocks the reader, effectively slowing down the reader to the writer rate.

The **Number of buffers** parameter must be an integer from 3 to 64.

Burst length — Burst length for memory transactions
256 (default) | scalar

The length of bursts for this connection on the memory bus in units of scalar data. The scalar unit is the packed data type. Specify the burst size for both writer and reader access to the channel.

The channel data is always transferred to the memory model using burst transactions. For the AXI4 configuration, the algorithm logic is responsible for defining the burst through the protocol signals.

The **Burst length** parameter determines the burst size to the memory, and the **wrData** signal defines the size of each transfer on the interface.

FIFO depth (number of bursts) — Depth of FIFO for data
8 (default) | scalar

Specify the depth of the data FIFO, in units of bursts. When the writer has no buffers to write to, the FIFO can absorb data until a buffer becomes available. This value is the maximum number of bursts that the FIFO can buffer before it drops data.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

FIFO almost-full depth (number of bursts) — Depth of FIFO when backpressure is asserted
6 (default) | scalar

Specify a number that asserts a backpressure signal from the channel to the data source. To avoid dropping data, set a high watermark, allowing the data producer enough time to react to backpressure. This number must be smaller than the FIFO depth.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

Clock frequency (MHz) — Interconnect frequency of manager datapath
`100` (default)

Specify the frequency of the manager datapath in MHz.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

Data width (bits) — Data width of manager datapath
64 (default) | scalar

Specify the data width of the manager datapath to the interconnect controller in bits.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

Signal Attributes

Stream data signal

Dimensions — Dimensions of input data signal
1 (default) | scalar | array

wrData can be a multidimensional array. Specify the dimension for the array as a whole number.

Example: 1 — A scalar sample.

Example: [10 1] — A vector of ten scalars.

Example: [1080 1920 3] — A 1080p frame. The frame includes 1080 lines of 1920 pixels per line, and each pixel is represented by three values (for red, green, and blue).

Data type — Data type of writer data
uint32 (default) | double | single | int8 | int16 | int32 | int64 | uint8 | uint16 | uint64 | boolean | fixed point

Specify the data type of the **wrData** port. For help, click the **...** button and select **Data Type Assistant**.

Sample time — Time interval of sampling
1 (default) | positive scalar | vector

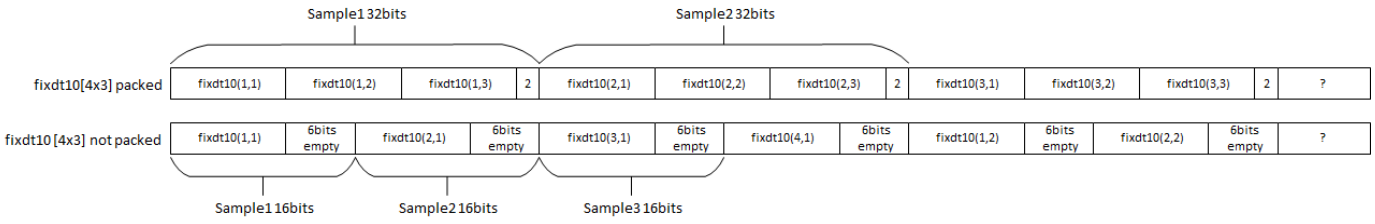
Specify a time interval in seconds to define how often the block updates.

When you do not want the output to have a time offset, specify the **Sample time** parameter as a scalar. To add a time offset to the output, specify the **Sample time** parameter as a 1-by-2 vector, where the first element is the sampling period and the second element is the offset. For more information about sample times in Simulink, see “Specify Sample Time”.

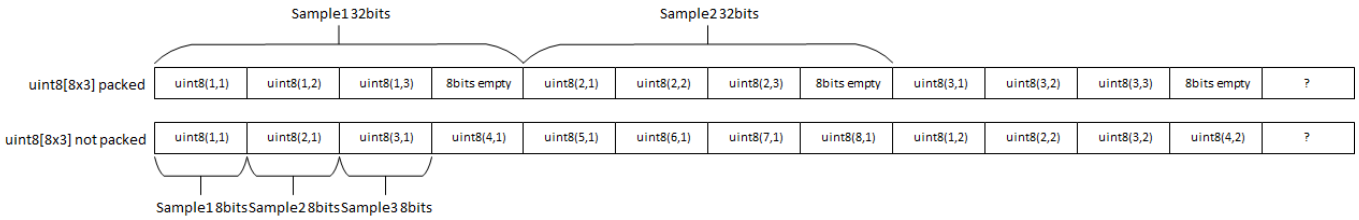
Enable sample packing (last signal dimension as channel) — Pack data on the last dimension of the signal
`off` (default) | `on`

Select this parameter to enable data packing across the last dimension of the signal. The AXI4-Stream to Software block packs the data along the last dimension of the signal. For example, if the channel data type is `uint32`, the dimensions are `[1024 4]`. If you select this sample packing parameter, then the memory channel generates 1024 read or write transactions of 128 bits. If you clear this sample packing parameter, the memory channel generates 4096 transactions of 32 bits each.

This figure shows how data is aligned for a signal with data type `fixdt10[4x3]`. When the data is packed, three 10-bit words are concatenated and extended by 2 bits to a 32-bit word. When the data is not packed, each 10-bit word is extended to a 16-bit word.



This figure shows how data is aligned for a signal with data type `uint8[8x3]`. When the data is packed, three 8-bit words are concatenated and extended by 8 bits to a 32-bit word. When the data is not packed, each 8-bit word is represented as an 8-bit sample.



The combined width of the flattened signal must not exceed 512 bits.

Software data signal

Dimensions — Dimensions of software data
scalar

Specify the dimension for the software data (reader) as a whole number.

The default value is `1024`.

Data type — Data type of software data
Inherit: Same as input (default) | uint16 | uint32 | uint64 | `fixdt(0,128,0)`

Specify the data type of the software data (reader). For help, click the **...** button and select **Data Type Assistant**. By default, this value is set to inherit the data type from the source signal.

Performance

Local master

View performance plots — Display performance metrics
button

Clicking the button opens performance plots for the local master in a new window. For more information about performance graphs, see “Memory Channel Latency Plots”.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

Memory Controller

View performance plots — Display performance metrics
button

Clicking the button opens the **Performance Plots for Memory Controller** window. You can then select to plot bandwidth, bursts, or latencies. For more information about performance graphs, see “Memory Controller Latency Plots”.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

Version History

Introduced in R2022b

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

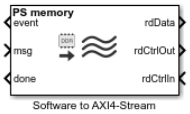
Memory Channel | AXI4 Random Access Memory | Software to AXI4-Stream | AXI4 Video Frame Buffer

Topics

“External Memory Channel Protocols”

Software to AXI4-Stream

Stream AXI4 data from software to FPGA

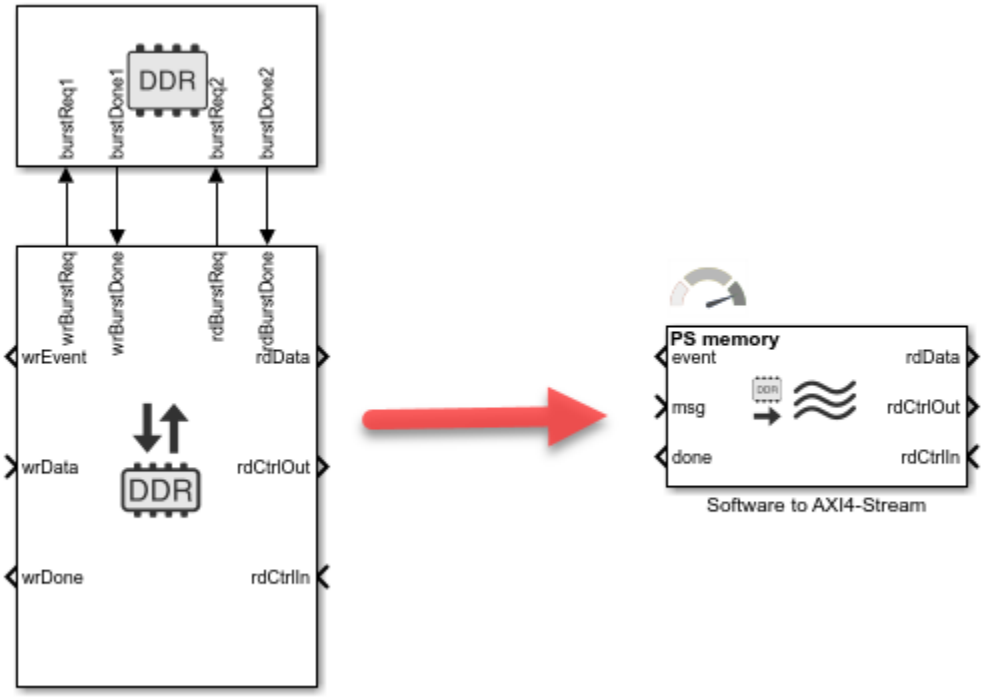


Libraries:
SoC Blockset / Memory

Description

The Software to AXI4-Stream block models a connection between hardware logic and a software task through external memory. The writer (processor) streams data into the channel through a DMA driver using a MathWorks simplified AXI stream protocol. The block models the datapath and software stack of that connection, including a FIFO, DMA engine, interconnect and external memory, interrupts, kernel buffer management of the DMA driver, and data transfers from the software task.

This block is equivalent to a Memory Channel block with the **Channel type** parameter set to `Software to AXI4-Stream via DMA` connected to a Memory Controller block.



For more information about the MathWorks simplified AXI stream protocol, see “AXI4-Stream Interface”.

Ports

Input

msg — Input data signal from software
scalar | vector | matrix

This signal contains the data sent from the processor to the algorithm.

Data Types: `SoCData`

rdCtrlIn — Reader input control signal
bus

This port accepts a control bus from a data consumer block, signaling that the consumer block is ready to accept read data. The **rdCtrlIn** port is a backpressure signal from a data consumer to the Software to AXI4-Stream block. To create this control bus, use the SoC Bus Creator block.

Data Types: `StreamS2MBusObj`

Output

rdData — Output data signal to data consumer
scalar | vector | matrix

This signal contains the data read from the processor through the memory.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

rdCtrlOut — Reader control signal to data consumer
bus

This signal represents the protocol bus from the memory channel to the data consumer. Connect this signal to the data consumer. To separate the signal from the control bus, use the SoC Bus Selector block.

Data Types: `StreamM2SBusObj`

event — Task write event signal
scalar

This port sends a task event signal that triggers the Task Manager block to execute the associated event-driven write.

Data Types: `rteEvent`

done — Notification of freed buffer in memory
scalar

This message port sends a notification to the connected Stream Write block. This notification indicates that a read transaction completed, and that a buffer in memory is available for writing.

Data Types: `Boolean`

Parameters

Memory simulation — Set simulation timing accuracy
`Burst accurate` (default) | `Protocol accurate` | `Behavioral`

Memory simulation provides three levels of timing resolution. Select one of these options:

- **Burst accurate** — Simulates memory contention and high-resolution timing.
- **Protocol accurate** — Simulates AXI4 protocol hand-shaking sequencing and low-resolution timing.
- **Behavioral** — Simulates data transactions only and no timing.

  When this parameter is set to **Behavioral** mode, the value of the `TLAST` signal is ignored.

Main

Memory selection — Choose between memory regions
`PS memory` (default) | `PL memory`

Choose between processing subsystem (PS) or programming logic (PL) memory.

- If the selected board supports only PL memory, then the default value is `PL memory`.
- If the selected board supports only PS memory or only PL memory, then this parameter is read-only.
- If the selected board is not a supported SoC board, then this parameter is not visible.

Region size (bytes) — Size of the allocated memory region, in bytes
calculated

This parameter is read-only.

The size of the region in bytes. This value is calculated as the number of buffers multiplied by the size of the buffer. The size of the buffer is defined as the size of the data multiplied by the software frame size. Define this value by setting the **Data type** and **Dimensions** parameters on the **Signal Attributes** tab.

Example: A software frame size of 1024 uint32 defines a buffer of 4096 Bytes. If the number of buffers is set to 2, the region size is 8192 bytes.

Number of buffers — Number of buffers in region
8 (default) | integer

Define the number of buffers in the memory.

The memory access has a ring-buffer pattern. The writer can continually write as long as buffers are available. When a buffer is completed, it becomes available for the reader. The writer and reader traverse the buffers in a circular pattern. As long as the writer and reader maintain similar rates, the buffering prevents blockage.

A disparate rate between a reader and a writer slows down the faster device. For example, a slow reader causes the writer to run out of buffers and blocks the writer, effectively slowing down the writer to the reader rate. Likewise, a slow writer causes the reader to run out of buffers and blocks the reader, effectively slowing down the reader to the writer rate.

The **Number of buffers** parameter must be an integer from 3 to 64.

Burst length — Burst length for memory transactions
256 (default) | scalar

The length of bursts for this connection on the memory bus in units of scalar data. The scalar unit is the packed data type. Specify the burst size for both writer and reader access to the channel.

The channel data is always transferred to the memory model using burst transactions. For the AXI4 configuration, the algorithm logic is responsible for defining the burst through the protocol signals.

The **Burst length** parameter determines the burst size to the memory, and the **rdData** signal defines the size of each transfer on the interface.

FIFO depth (number of bursts) — Depth of FIFO for data
8 (default) | scalar

Specify the depth of the data FIFO, in units of bursts. When the writer has no buffers to write to, the FIFO can absorb data until a buffer becomes available. This value is the maximum number of bursts that the FIFO can buffer before it drops data.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

FIFO almost full depth (number of bursts) — Depth of FIFO when backpressure is asserted
6 (default) | scalar

Specify a number that asserts a backpressure signal from the channel to the data source. To avoid dropping data, set a high watermark, allowing the data producer enough time to react to backpressure. This number must be smaller than the FIFO depth.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

Clock frequency (MHz) — Interconnect frequency of manager datapath
`100` (default)

Specify the frequency of the manager datapath in MHz.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

Data width (bits) — Data width of manager datapath
64 (default) | scalar

Specify the data width of manager datapath to the interconnect controller in bits.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

Signal Attributes

Software data signal

Dimensions — Dimensions of software data
`1024` (default) | scalar

Specify the dimension for the software data (writer) as a whole number.

The default value is `1024`.

Data type — Data type of software data
uint32 (default) | uint16 | uint64 | `fixdt(1,16,0)`

Specify the data type of the software data (writer). For help, click the **...** button and select **Data Type Assistant**. By default, this value is set to inherit the data type from the source signal.

Stream data signal

Dimensions — Dimensions of output data signal
1 (default) | scalar | array

rdData can be a multidimensional array. Specify the dimension for the array as a whole number.

Example: 1 — A scalar sample.

Example: [10 1] — A vector of ten scalars.

Example: [1080 1920 3] — A 1080p frame. The frame includes 1080 lines of 1920 pixels per line, and each pixel is represented by three values (for red, green, and blue).

Data type — Data type of reader data
Inherit: Same as input (default) | double | single | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | boolean | fixed point

Specify the data type of the **rdData** port. For help, click the **...** button and select **Data Type Assistant**. By default, this value is set to inherit the data type from the source signal.

Sample time — Time interval of sampling
1 (default) | positive scalar | vector

Specify a time interval in seconds to define how often the block updates.

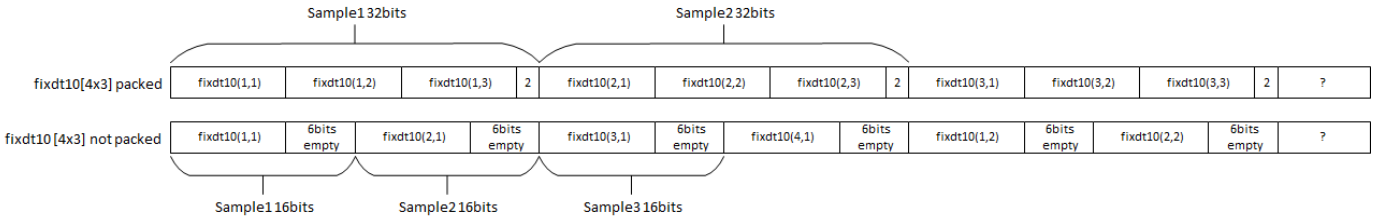
When you do not want the output to have a time offset, specify the **Sample time** parameter as a scalar. To add a time offset to the output, specify the **Sample time** parameter as a 1-by-2 vector, where the first element is the sampling period and the second element is the offset. For more information about sample times in Simulink, see “Specify Sample Time”.

Enable sample packing (last signal dimension as channel) — Pack data on last dimension of signal
`off` (default) | `on`

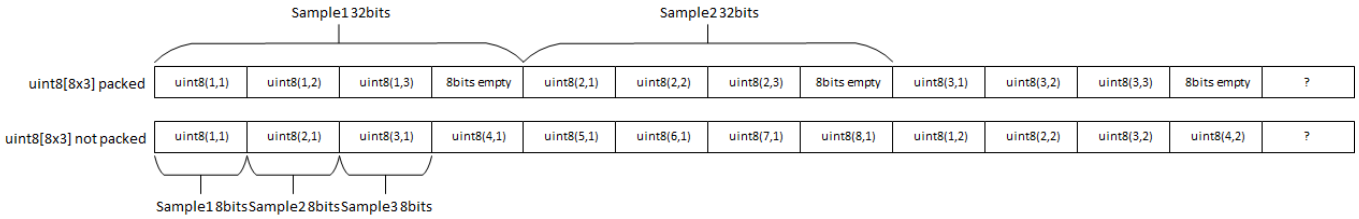
Select this parameter to enable data packing across the last dimension of the signal. The AXI4-Stream to Software block packs the data along the last dimension of the signal. For example, if the channel data type is `uint32`, the dimensions are `[1024 4]`. If you select this sample packing parameter, then the memory channel generates 1024 read or write transactions of 128 bits. If you

clear this sample packing parameter, the memory channel generates 4096 transactions of 32 bits each.

This figure shows how data is aligned for a signal with data type `fixdt10[4x3]`. When the data is packed, three 10-bit words are concatenated and extended by 2 bits to a 32-bit word. When the data is not packed, each 10-bit word is extended to a 16-bit word.



This figure shows how data is aligned for a signal with data type `uint8[8x3]`. When the data is packed, three 8-bit words are concatenated and extended by 8 bits to a 32-bit word. When the data is not packed, each 8-bit word is represented as an 8-bit sample.



The combined width of the flattened signal must not exceed 512 bits.

Performance

Local Master

View performance plots — Display performance metrics
button

Clicking the button opens performance plots for the local manager in a new window. For more information about performance graphs, see “Memory Channel Latency Plots”.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

Memory Controller

View performance plots — Display performance metrics
button

Clicking the button opens the **Performance Plots for Memory Controller** window. You can then select to plot bandwidth, bursts, or latencies. For more information about performance graphs, see “Memory Controller Latency Plots”.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

Version History

Introduced in R2022b

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

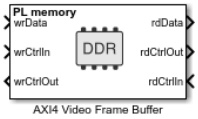
Memory Channel | AXI4 Random Access Memory | AXI4-Stream to Software | AXI4 Video Frame Buffer

Topics

“External Memory Channel Protocols”

AXI4 Video Frame Buffer

Model connection between two hardware algorithms through external memory

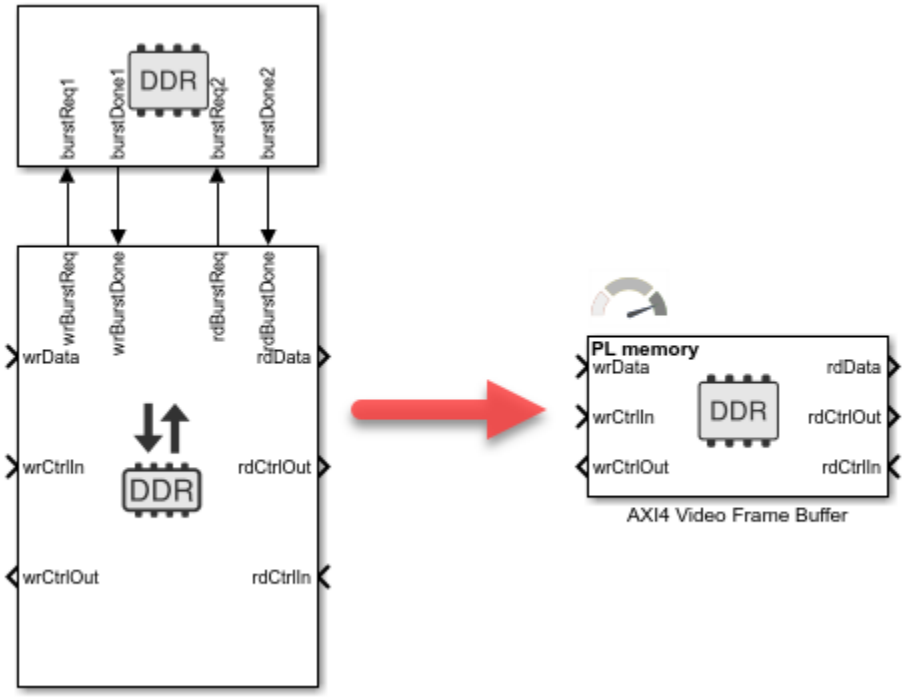


Libraries:
SoC Blockset / Memory

Description

The AXI Video Frame Buffer block models a connection between two hardware algorithms through external memory, using full video frame buffers. The protocol is the MathWorks streaming pixel protocol with back pressure. Also, the reader can ensure that the frame buffer is synchronized with downstream video timings by asserting an FSYNC protocol signal. The datapath includes a Video-DMA (VDMA) engine and the external memory buffers are managed as a circular buffer of full video frames.

This block is equivalent to a Memory Channel block with the **Channel type** parameter set to `AXI4 Video Frame Buffer` connected to a Memory Controller block.



For more information, see “AXI4-Stream Video Interface”.

Ports

Input

wrData — Writer data bus signal
scalar | vector | matrix

This signal contains the data to the memory.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

wrCtrlIn — Writer input control signal
bus

This port represents the protocol from the data producer to the memory channel. The AXI4 Video Frame Buffer block checks this signal when using **wrData**. To create this control bus, use the SoC Bus Creator block. For more information about bus types, see “External Memory Channel Protocols”.

Data Types: `pixelcontrol`

rdCtrlIn — Reader input control signal
bus

This port accepts a bus from a data consumer block, signaling that the consumer block is ready to accept read data. The **rdCtrlIn** port is a backpressure signal from a data consumer to the memory. To create this control bus, use the SoC Bus Creator block.

Data Types: `StreamVideoFSyncS2MBusObj`

Output

rdData — Output data signal to data consumer
scalar | vector | matrix

This signal contains the data read from the memory.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point` | `SoCData`

rdCtrlOut — Reader control signal from memory channel to data consumer
bus

This bus represents the protocol bust from the memory channel to the data consumer. To separate the signal from the bus, use the SoC Bus Selector block.

Data Types: `pixelcontrol`

wrCtrlOut — Writer control signal from memory channel to data producer
bus

This bus represents the protocol bus from the memory channel to the data producer. To separate the signal from the bus, use the SoC Bus Selector block.

Data Types: `StreamVideoS2MBusObj`

Parameters

Memory simulation — Set simulation timing accuracy
`Burst accurate` (default) | `Protocol accurate` | `Behavioral`

Memory simulation provides three levels of timing resolution. Select one of these options:

- **Burst accurate** — Simulates memory contention and high-resolution timing.
- **Protocol accurate** — Simulates AXI4 protocol hand-shaking sequencing and low-resolution timing.
- **Behavioral** — Simulates data transactions only and no timing.

Main

Memory selection — Choose between memory regions
`PS memory` (default) | `PL memory`

Select between processing subsystem (PS) or programming logic (PL) memory.

- If the selected board supports only PL memory, then the default value is `PL memory`.
- If the selected board supports only PS memory or only PL memory, then this parameter is read-only.
- If the selected board is not a supported SoC board, then this parameter is not visible.

Region size (bytes) — Size of the allocated memory region, in bytes
calculated

This parameter is read-only.

The size of the region in bytes. This value is calculated as the number of frames multiplied by the frame size multiplied by the size of the data. Define this value by setting the **Frame size** and **Number of Frames** parameters on the **Main** tab, and the **Data type** and **Dimensions** parameters on the **Signal Attributes** tab.

Example: A software frame size of 1024 uint32 defines a buffer of 4096 bytes. If the number of buffers is set to 2, the region size is 8192 bytes.

Frame size — Frame dimensions
`1080p HDTV (1920x1080p)` (default)

Select or enter the frame size in a combo box.

Select a frame format from the list or enter frame dimension as a 2 element vector such as `[1920 1080]`.

- `480p SDTV (720x480p)`
- `576p SDTV (720x576p)`
- `720p HDTV (1280x720p)`
- `1080p HDTV (1920x1080p)`
- `160x120p`
- `320x240p`

- `640x480p`
- `800x600p`
- `1024x768p`
- `1280x768p`
- `1280x1024p`
- `1360x768p`
- `1400x1050p`
- `1600x1200p`
- `1680x1050p`
- `1920x1200p`

Number of Frames — Number of buffers in region
3 (default) | integer

Define the number of video frames in the memory.

The memory access has a ring-buffer pattern. The writer can continually write as long as buffers are available. When a buffer is completed, it becomes available for the reader. The writer and reader traverse the buffers in a circular pattern. As long as the writer and reader maintain similar rates, the buffering prevents blockage.

A disparate rate between a reader and a writer slows down the faster device. For example, a slow reader causes the writer to run out of buffers and blocks the writer, effectively slowing down the writer to the reader rate. Likewise, a slow writer causes the reader to run out of buffers and blocks the reader, effectively slowing down the reader to the writer rate.

The **Number of Frames** parameter must be an integer from 3 to 64.

Burst length — Burst length for memory transactions
256 (default) | scalar

The length of bursts for this connection on the memory bus in units of scalar data. The scalar unit is the packed data type. Specify the burst size for both writer and reader access to the channel.

The channel data is always transferred to the memory model using burst transactions. For the AXI4 configuration, the algorithm logic is responsible for defining the burst through the protocol signals.

The **Burst length** parameter determines the burst size to the memory, and the **rdData** signal defines the size of each transfer on the interface.

FIFO depth (number of bursts) — Depth of FIFO for data
8 (default) | scalar

Specify the depth of the data FIFO, in units of bursts. When the writer has no buffers to write to, the FIFO can absorb data until a buffer becomes available. This value is the maximum number of bursts that the FIFO can buffer before it drops data.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

FIFO almost-full depth (number of bursts) — Depth of FIFO when backpressure is asserted
`6` (default) | scalar

Specify a number that asserts a backpressure signal from the channel to the data source. To avoid dropping data, set a high watermark, allowing the data producer enough time to react to backpressure. This number must be smaller than the FIFO depth.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

Clock frequency (MHz) — Interconnect frequency of manager datapath
`100` (default)

Specify the frequency of the manager datapath in MHz.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

Data width (bits) — Data width of manager datapath
`64` (default) | scalar

Specify the data width of the manager datapath to the interconnect controller in bits.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

Signal Attributes

Data signal

Dimensions — Dimensions of video data
`[1 3]` (default) | scalar | vector

Specify the dimension for the video data as a whole scalar or vector.

Example: `[1 3]` — A pixel represented by three values (for red, green, and blue).

Example: `[1080 1920 3]` — A 1080p frame. The frame includes 1080 lines of 1920 pixels per line, and each pixel is represented by three values (for red, green, and blue).

Data type — Data type of video pixel
uint8 (default) | uint16 | uint32 | uint64 | `fixdt(1,16,0)`

Specify the data type of the video pixel. For help, click the **...** button and select **Data Type Assistant**. By default, this value is set to inherit the data type from the source signal.

Sample time — Time interval of sampling
1 (default) | positive scalar | vector

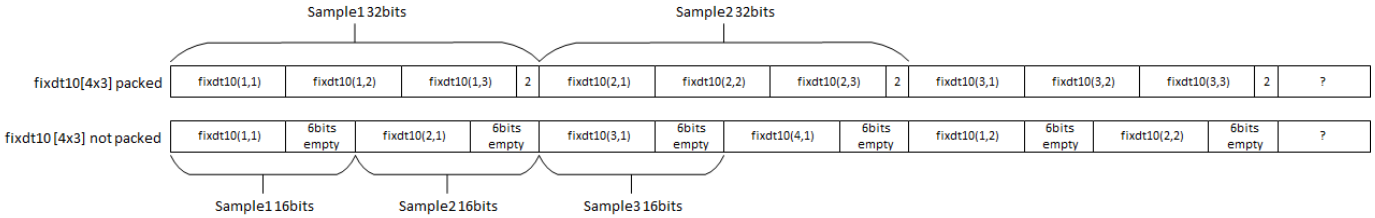
Specify a time interval in seconds to define how often the block updates.

When you do not want the output to have a time offset, specify the **Sample time** parameter as a scalar. To add a time offset to the output, specify the **Sample time** parameter as a 1-by-2 vector, where the first element is the sampling period and the second element is the offset. For more information about sample times in Simulink, see “Specify Sample Time”.

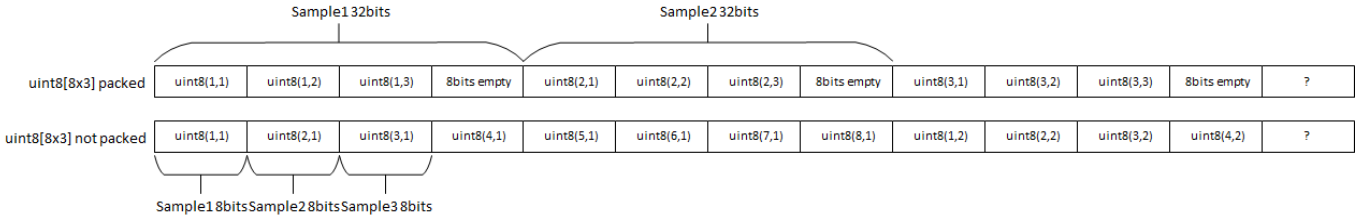
Enable sample packing (last signal dimension as channel) — Pack data on last dimension of signal
on (default) | off

Select this parameter to enable data packing across the last dimension of the signal. For example, if the channel data type is `uint32`, the dimensions are `[1024 4]`. If you select this sample packing parameter, then the memory channel generates 1024 read or write transactions of 128 bits. If you clear this sample packing parameter, the memory channel generates 4096 transactions of 32 bits each.

This figure shows how data is aligned for a signal with data type `fixdt10[4x3]`. When the data is packed, three 10-bit words are concatenated and extended by 2 bits to a 32-bit word. When the data is not packed, each 10-bit word is extended to a 16-bit word.



This figure shows how data is aligned for a signal with data type `uint8[8x3]`. When the data is packed, three 8-bit words are concatenated and extended by 8 bits to a 32-bit word. When the data is not packed, each 8-bit word is represented as an 8-bit sample.



The combined width of the flattened signal must not exceed 512 bits.

Inherit sample time offset — Make reader inherit sample time from writer
on (default) | off

Select this box to make the reader inherit the sample time offset from the writer. Clear the box to use a different sample time offset from the writer. For more information about sample time offsets, see Sample Time Offset.

Use pixel clock sample times — Use pixel clock sample time
off (default) | on

Select this box to use the pixel clock sample time. To use the pixel clock sample time, you must use scalar pixel dimensions. The pixel clock sample time is relevant only when streaming pixels. If both the reader and the writer are streaming frames, you get an error when checking this box.

Performance

Local Masters

View performance plots — Display performance metrics
button

Clicking the button opens performance plots for the local manager in a new window. For more information about performance graphs, see “Memory Channel Latency Plots”.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

Memory Controller

View performance plots — Display performance metrics
button

Clicking the button opens the **Performance Plots for Memory Controller** window. You can then select to plot bandwidth, bursts, or latencies. For more information about performance graphs, see “Memory Controller Latency Plots”.

Dependencies

To enable this parameter, select **Burst accurate** under **Memory simulation**.

Version History

Introduced in R2022b

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

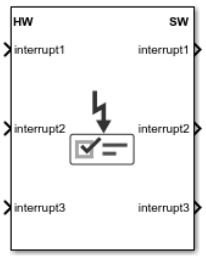
Memory Channel | AXI4 Random Access Memory | AXI4-Stream to Software | Software to AXI4-Stream

Topics

“External Memory Channel Protocols”

Interrupt Channel

Send interrupt to processor from hardware



Libraries:
SoC Blockset / Memory

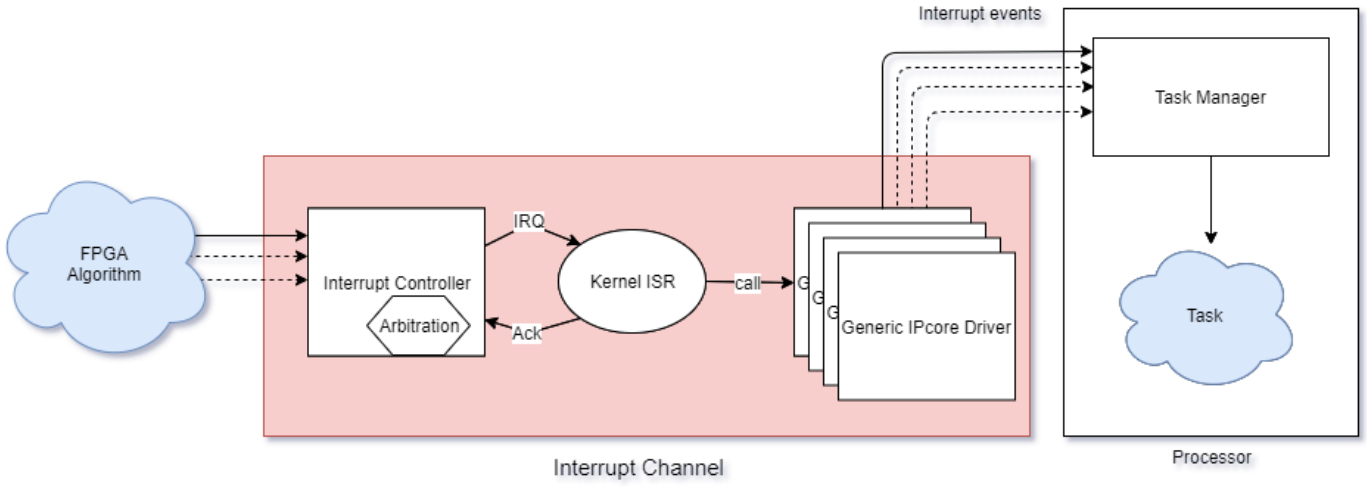
Description

The Interrupt Channel block receives interrupt requests from FPGA logic or the Memory Channel block, arbitrates the requests, and triggers an event-driven software task to the Task Manager block. You can connect up to 16 devices to the interrupt controller, with one interrupt per device. The block consists of these three parts:

- Interrupt Controller – This part accepts interrupt requests (IRQs) and arbitrates them according to a user-specified priority. When concurrent requests to the interrupt controller exist, requests with a higher priority are processed before those with a lower priority.
- Kernel Interrupt Service Routine (ISR) – This part receives an interrupt request from the Interrupt Controller, serves the interrupt, and sends an acknowledge signal back to the Interrupt Controller, so that it can process the next IRQ.
- IPCore Driver (one per interrupt) – This part receives a request from the Kernel ISR and triggers an event-driven task in the processor.

Even though the interrupt channel can have more than one interrupt output toward the processor, it sends no more than one active interrupt event to the processor at any given time.

This image shows a conceptual view of an Interrupt Channel block, that accepts interrupt requests from an FPGA algorithm. After arbitration, the kernel serves the request and triggers an event to a processor algorithm.



Ports

Input

interruptN — Interrupt request from hardware
`True` | `False`

Each interrupt is assigned a port pair: one input port and one output port. By default, the *N*th interrupt port is named interrupt*N*. You can change interrupt names by clicking **Edit** in the **Interrupts** parameter.

Connect this port to a Boolean signal from the FPGA logic or an event from a Memory Channel or Event Source block.

Dependencies

The number of input ports depends on the number of interrupts in the interrupt table.

Data Types: `rteEvent` | `Boolean`

Output

interruptN — Interrupt request signal to processor
scalar

Each interrupt is assigned a port pair: one input port and one output port. By default, the *N*th interrupt port is named interrupt*N*. You can change interrupt names by clicking **Edit** in the **Interrupts** parameter.

Connect this port to a task event input port in the Task Manager block.

Dependencies

The number of output ports depends on the number of interrupts in the interrupt table.

Data Types: `rteEvent`

Parameters

Interrupts — Interrupt name, trigger type, and priority
table

This parameter includes a table, where each of its lines corresponds to an interrupt in the Interrupt Channel block. Edit the table to add or edit an interrupt. The interrupt channel can have up to 16 interrupts.

For each interrupt, you can edit these values.

- **Interrupt Name** – Specify the interrupt name. This value changes the input and output port names for this interrupt.
- **Trigger Type** – Select the trigger type for the interrupt by choosing either of these options.
  - `Rising edge` – When the interrupt originates in FPGA logic
  - `SoC event` – When the interrupt originates in the Memory Channel block or Event Source block
- **Priority** – Set the priority for each interrupt is set in the **Priority** column. This value remains static. The top row represents the highest interrupt. Click **Move Up** to increase the priority of an interrupt. Click **Move Down** to decrease the priority of an interrupt.

Interrupt processing time — Processing time for interrupt access
`100e-6` (default) | positive scalar

This sample time represents the time required for the interrupt channel to arbitrate and execute an interrupt request. It is defined as the time required for the Interrupt Controller arbitration, Kernel ISR execution, and additional delay for the device driver.

Specify the processing time by entering a number, in seconds.

Version History

Introduced in R2020b

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Blocks

Task Manager | Memory Channel | Event Source

IP Core Register Read

Model register writes from software to hardware

Libraries:
SoC Blockset / Memory

Description

The IP Core Register Read block models a write operation from a processor to hardware logic. The block receives data sent with a Register Write block from the processor. You can define the register offset in the **Memory Mapper** tool.

Ports

Output

data — Data output
vector

This port outputs the data vector received from the processor, starting at the offset address from the base address of the IP core. Set the offset address in the **Memory Mapper** tool.

Data Types: `single` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Parameters

Register name — Name of register
`RegA` (default) | character vector

Match this name to the **Register name** parameter specified in the Register Write block.

Example: `AddressReg1`

Output data type — Data type of output data
`uint16` (default) | `single` | `int8` | `uint8` | `int16` | `int32` | `uint32` | `boolean` | `fixed point data type`

Select the data type for the output data. This value must match the value selected for the Register Write block.

Output vector size — Vector size of output data
1 (default) | positive integer

Specify the vector size of the output data as a positive integer. This value must match the value selected for the Register Write block.

Sample time — Simulation interval of sampling
`-1` (default) | nonnegative scalar

Specify a discrete time interval, in seconds, at which the block outputs data. If this value is `-1` (default), the sample time is inherited from the model.

Version History

Introduced in R2020a

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion

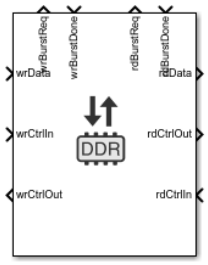
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Register Write | Register Channel

Memory Channel

Stream data through a memory channel



Libraries:
SoC Blockset / Memory

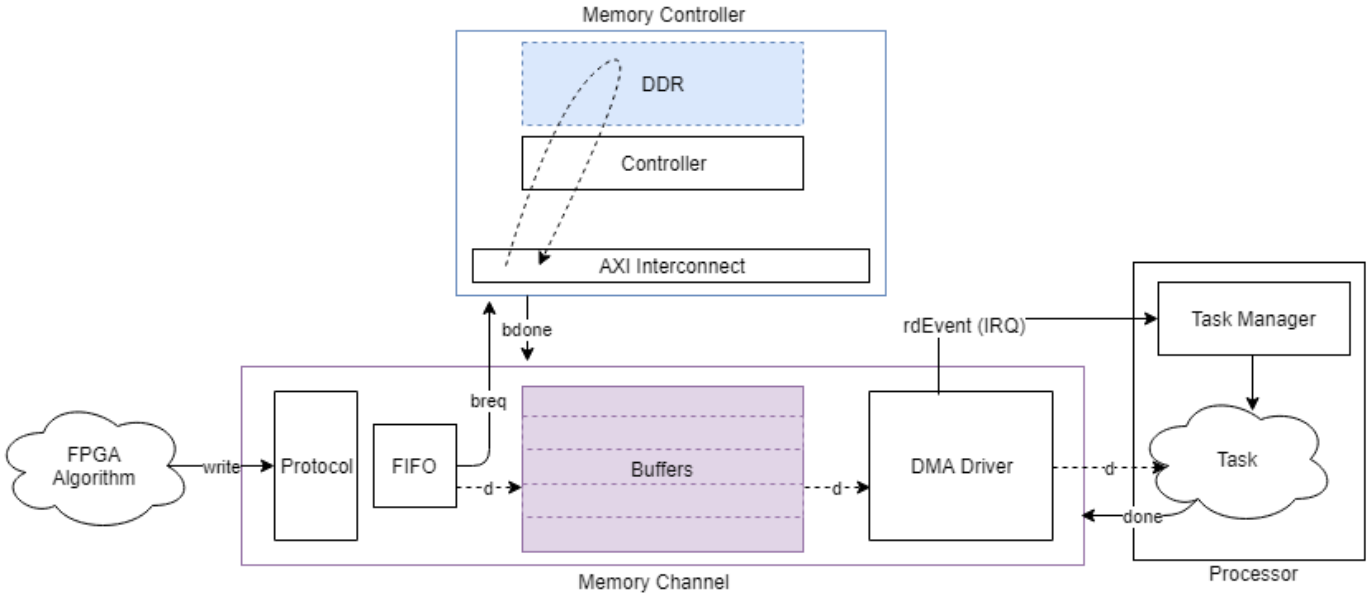
Description

The Memory Channel block streams data through external memory. Conceptually, it models data transfer between one algorithm and another, through shared memory. The algorithm can be hardware logic (HW), a processor model, or I/O devices. The writer algorithm requests access to memory from the Memory Controller block. After access is granted the writer algorithm writes to a memory buffer. In the model, the data storage is modeled as buffers in the channel. When deploying on hardware, the data is routed to an external shared memory.

This block can be configured to support any of these protocols:

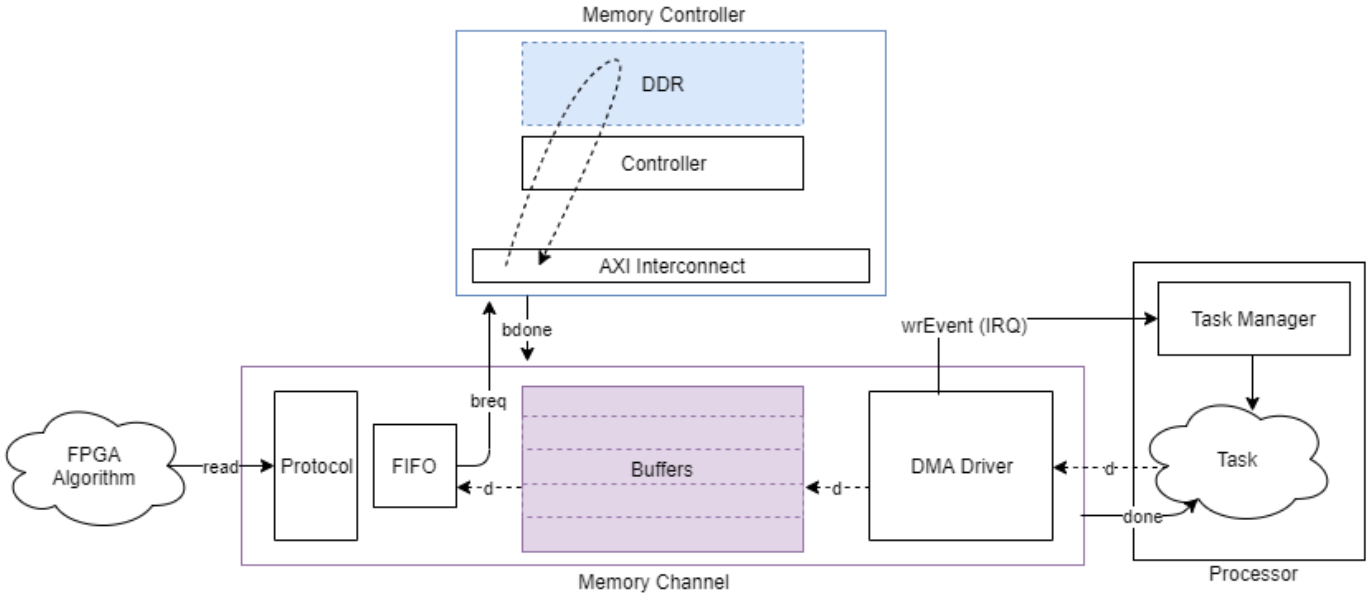
- **AXI4-Stream to Software via DMA** – Model a connection between hardware logic and a software task through external memory. The writer puts data into the channel using a MathWorks simplified AXI stream protocol and the reader (processor) gets data from a DMA driver interface. The channel models the datapath and software stack of that connection including a FIFO, DMA engine, interconnect and external memory, interrupts, kernel buffer management of the DMA driver, and data transfers to the software task. For more information about MathWorks simplified AXI stream protocol, see “AXI4-Stream Interface”.

  This image is a conceptual view of a Memory Channel block, streaming data from an FPGA algorithm to a processor algorithm.



- **Software to AXI4-Stream via DMA** – Model a connection between hardware logic and a software task through external memory. The writer (processor) streams data into the channel via a DMA driver using a MathWorks simplified AXI stream protocol. The channel models the datapath and software stack of that connection including a FIFO, DMA engine, interconnect and external memory, interrupts, kernel buffer management of the DMA driver, and data transfers from the software task. For more information about the MathWorks simplified AXI stream protocol, see “AXI4-Stream Interface”.

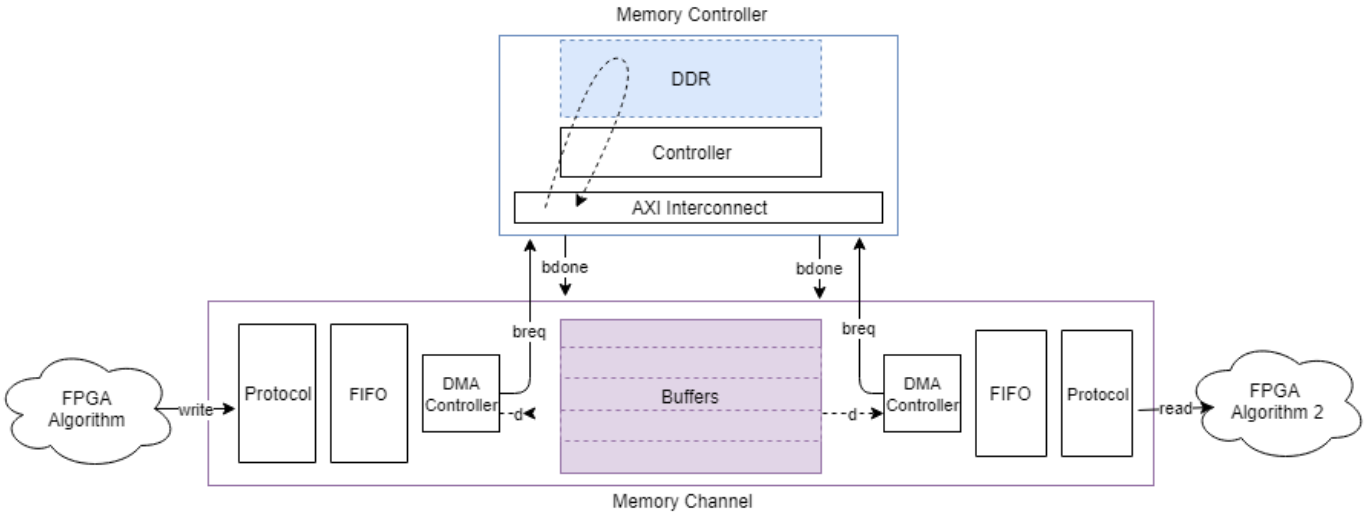
  This image is a conceptual view of a Memory Channel block, streaming data from a processor algorithm to an FPGA algorithm.



- **AXI4-Stream FIFO** – Model a connection between two FPGA algorithms through external memory. The writer puts data into the channel as a master using the MathWorks simplified AXI stream protocol and the reader receives data from the channel as a slave using the same protocol.

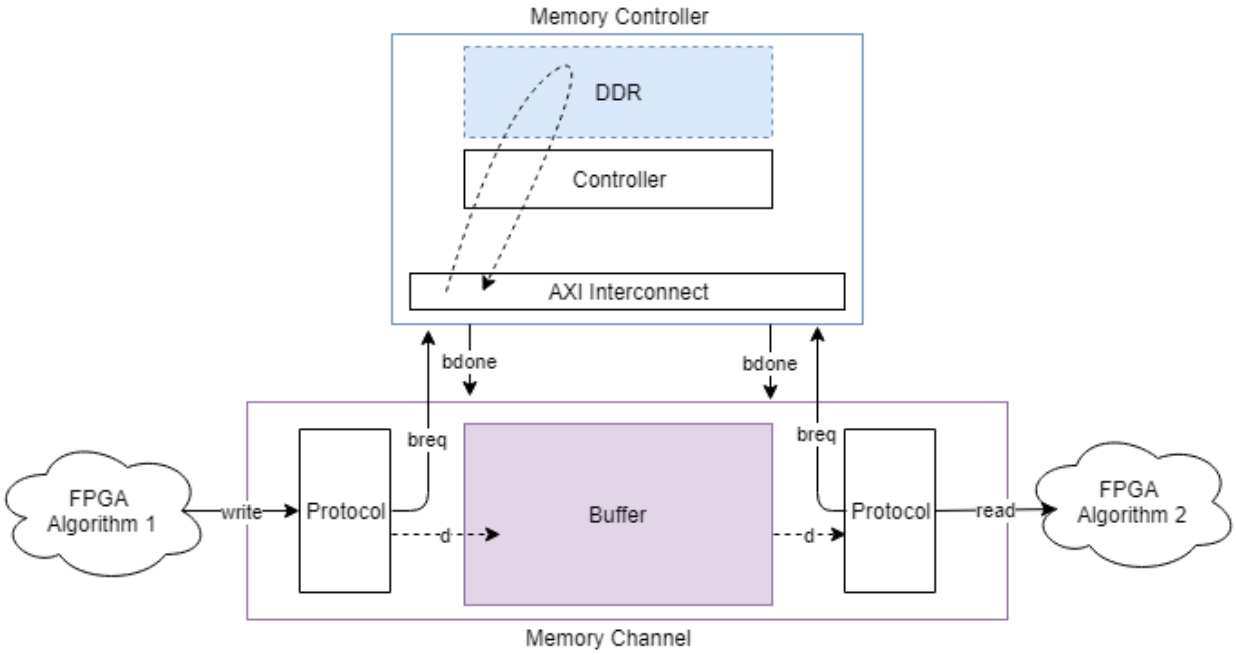
The channel behaves as a first in first out (FIFO) memory. The channel models the datapath of the connection. The Memory Channel block includes an intermediate burst-level FIFO, DMA engine, interconnect, and external memory. The external memory itself is managed as a circular buffer, where a buffer must be written before it can be read. For more information about the MathWorks simplified AXI stream protocol, see “AXI4-Stream Interface”.

This image is a conceptual view of a Memory Channel block, streaming data from one FPGA algorithm to another FPGA algorithm.



- **AXI4-Stream Video FIFO** – Model a connection between two hardware algorithms through external memory. This channel structure is similar to the **AXI4 Stream FIFO** configuration, but the writer and reader are using the MathWorks streaming pixel protocol, along with a back-pressure signal. For more information, see “AXI4-Stream Video Interface”.
- **AXI4-Stream Video Frame Buffer** – Model a connection between two hardware algorithms through external memory, using full video frame buffers. The protocol is the MathWorks streaming pixel protocol with back pressure. Also, the reader can ensure that the frame buffer is synchronized with downstream video timings by asserting an FSYNC protocol signal. The datapath includes a Video-DMA (VDMA) engine and the external memory buffers are managed as a circular buffer of full video frames. The channel structure is identical to the structure of **AXI4 Stream FIFO** channel type.
- **AXI4-Random Access** – Model a connection between two hardware algorithms through external memory, using the MathWorks simplified AXI4-Master protocol. Both the writer and the reader are masters, the channel is a slave in both cases. The external memory is unmanaged (there are no logical buffers, and no circular buffer). It is up to the reader and writer to coordinate timing on accesses to ensure the integrity of the data. For more information, see “Simplified AXI4 Master Interface”.

  This image is a conceptual view of a Memory Channel block, with random-access to the memory for writing, and random-access to the memory for reading.



For more information on the available protocols, see “External Memory Channel Protocols”.

Limitations

- A model containing a Memory Channel blocks does not support simulation stepping. For more information on simulation stepping, see “Debug Simulations in the Simulink Editor”.

Ports

Input

wrData — Writer data bus signal
scalar | vector | matrix

This signal contains the data to the memory.

Note When the **Channel type** parameter is set to `Software to AXI4-Stream via DMA`, this port receives the input data, as a message, from the connected Stream Write block. For more information on messages, see “Messages”.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point` | `SoCData`

wrCtrlIn — Writer input control signal
bus

This port represents the protocol from the data producer to the memory channel. The Memory Channel block checks this signal when using **wrData**. The signals on the bus depend on the **Channel**

type parameter. Use the SoC Bus Creator block to create this control bus. For more information about bus types, see “External Memory Channel Protocols”.

Channel Type Configuration	Bus Type
`AXI4-Stream to Software via DMA`	`StreamM2SBusObj`
`AXI4 Stream FIFO`	`StreamM2SBusObj`
`AXI4 Stream Video FIFO`	`pixelcontrol`
`AXI4 Stream Video Frame Buffer`	`pixelcontrol`
`AXI4 Random Access`	`WriteControlM2SBusObj`

Dependencies

To enable this port, set the **Channel type** parameter to a value other than `Software to AXI4-Stream via DMA`.

Data Types: `StreamM2SBusObj` | `pixelcontrol` | `WritecontrolM2SBusObj`

rdCtrlIn — Reader input control signal
bus

This port accepts a bus from a data consumer block, signaling that the consumer block is ready to accept read data. For streaming protocols, the **rdCtrlIn** port is a backpressure signal from a data consumer to the Memory Channel block. For the `AXI4 Random Access` protocol, this input is a read-request from the reader. The signals on the bus depend on the **Channel type** parameter. Use the SoC Bus Creator block to create this control bus.

Channel Type Configuration	Bus Type
`Software to AXI4-Stream via DMA`	`StreamS2MBusObj`
`AXI4 Stream FIFO`	`StreamS2MBusObj`
`AXI4 Stream Video FIFO`	`StreamVideoS2MBusObj`
`AXI4 Stream Video Frame Buffer`	`StreamVideoFSyncS2MBusObj`
`AXI4 Random Access`	`ReadControlM2SBusObj`

Dependencies

To enable this port, set the **Channel type** parameter to a value other than `AXI4-Stream to Software via DMA`.

Data Types: `StreamS2MBusObj` | `StreamVideoS2MBusObj` | `StreamVideoFSyncS2MBusObj` | `ReadControlM2SBusObj`

rdDone — Notification message of completed read
scalar

This message port receives a notification from the connected Stream Read block. The notification indicates that a read transaction completed. For more information on messages, see “Messages”.

Dependencies

To enable this port, set the **Channel type** parameter to `AXI4-Stream to Software via DMA`.

Data Types: `Boolean`

wrBurstDone — Writer control input from memory controller
scalar

This message port receives control messages from a connected Memory Controller block that the requested burst transaction completed. Connect the **burstDone** output signal from the Memory Controller block to this port. For more information on messages, see “Messages”.

Data Types: `BurstRequest2BusObj`

rdBurstDone — Reader control input from memory controller
scalar

This message port receives control messages from a connected Memory Controller block that the requested burst transaction completed. Connect the **burstDone** output signal from the Memory Controller block to this port. For more information on messages, see “Messages”.

Data Types: `BurstRequest2BusObj`

Output

rdData — Output data signal to data consumer
scalar | vector | matrix

This signal contains the data read from the memory.

Note When the **Channel type** parameter is set to `AXI4-Stream to Software via DMA`, this port sends the output data, as a message, to the connected Stream Read block. For more information on messages, see “Messages”.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point` | `SoCData`

rdEvent — Task read event signal
scalar

This port sends a task event signal that triggers the Task Manager block to execute the associated event-driven read.

Dependencies

To enable this port, set the **Channel type** parameter to `AXI4-Stream to Software via DMA`.

Data Types: `rteEvent`

wrEvent — Task write event signal
scalar

This port sends a task event signal that triggers the Task Manager block to execute the associated event-driven write.

Dependencies

To enable this port, set the **Channel type** parameter to `Software to AXI4-Stream via DMA`.

Data Types: `rteEvent`

wrDone — Notification of freed buffer in memory
scalar

This message port sends a notification to the connected Stream Write block. This notification indicates that a read transaction completed, and that a buffer in memory is available for writing.

Dependencies

To enable this port, set the **Channel type** parameter to `Software to AXI4-Stream via DMA`.

Data Types: `Boolean`

rdCtrlOut — Reader control signal from memory channel to data consumer
bus

Control signal from channel to data consumer. The contents of this signal depend on the `Channel type` parameter. Connect this signal to the data consumer. Use the SoC Bus Selector block to separate the signal from the bus.

Channel Type Configuration	Bus Type
`Software to AXI4-Stream via DMA`	`StreamM2SBusObj`
`AXI4 Stream FIFO`	`StreamM2SBusObj`
`AXI4 Stream Video FIFO`	`pixelcontrol`
`AXI4 Stream Video Frame Buffer`	`pixelcontrol`
`AXI4 Random Access`	`ReadControlS2MBusObj`

Dependencies

To enable this port, set the **Channel type** parameter to a value other than `AXI4-Stream to Software via DMA`.

Data Types: `StreamM2SBusObj` | `ReadControlS2MBusObj` | `pixelcontrol`

wrCtrlOut — Writer control signal from memory channel to data producer
bus

This bus represents the protocol bus from the memory channel to the data producer. The signals on the bus depend on the **Channel type** parameter. Use the SoC Bus Selector block to separate the signal from the bus.

Channel Type Configuration	Bus Type
`AXI4-Stream to Software via DMA`	`StreamS2MBusObj`
`AXI4 Stream FIFO`	`StreamS2MBusObj`
`AXI4 Stream Video FIFO`	`StreamVideoS2MBusObj`
`AXI4 Stream Video Frame Buffer`	`StreamVideoS2MBusObj`
`AXI4 Random Access`	`WriteControlS2MBusObj`

Dependencies

To enable this port, set the **Channel type** parameter to a value other than `Software to AXI4-Stream via DMA`.

Data Types: `StreamS2MBusObj` | `WriteControlS2MBusObj` | `StreamVideoS2MBusObj`

wrBurstReq — Write burst request
scalar

This message port sends control signal requesting burst access from the memory controller. Connect it to the **burstReq** input of the Memory Controller block. For more information on messages, see "Messages".

Data Types: `BurstRequestBusObj`

rdBurstReq — Read burst request
scalar

This message port sends control signal requesting burst access from the memory controller. Connect it to the **burstReq** input of the Memory Controller block. For more information on messages, see "Messages".

Data Types: `BurstRequestBusObj`

Parameters

Hardware board — View or modify current hardware settings
name of selected hardware board

This parameter is read-only.

This parameter shows a link to the currently selected hardware board. Click the link to open the configuration parameters, and adjust the settings, or choose a different board.

To learn more about configuration parameters, see "FPGA design (mem channels)" on page 2-7.

Show implementation info — View channel information
text window

This parameter is read-only.

This parameter shows a link to the implementation information specific to the model. Click the link to view the information (opens in new window).

Enable memory simulation — Control memory simulation accuracy
`on` (default) | `off`

Select this parameter to enable memory simulation with high accuracy. Clear for faster simulation performance.

- `on` — The block simulates memory transactions with burst accuracy. Bursts are constructed and arbitrated for access to memory.
- `off` — Memory transactions are purely behavioral, allowing fast simulation.

To see an example that uses this feature go to "Accelerate SoC Simulation by Varying Abstraction Levels".

Main

Channel type — Choose channel protocol
`AXI4-Stream FIFO` (default) | `AXI4-Stream to Software via DMA` | `Software to AXI4-Stream via DMA` | `AXI4-Stream Video FIFO` | `AXI4-Stream Video Frame Buffer` | `AXI4 Random Access`

Specify the protocol for the channel. Choose one of the following values:

- AXI4-Stream to Software via DMA
- Software to AXI4-Stream via DMA
- AXI4 Stream FIFO
- AXI4 Stream Video FIFO
- AXI4 Stream Video Frame Buffer
- AXI4 Random Access

Note You can use a simplified memory block for the following configurations (recommended):

- AXI4-Stream to Software via DMA — Use an AXI4-Stream to Software block.
- Software to AXI4-Stream via DMA — Use a Software to AXI4-Stream block.
- AXI4 Stream Video Frame Buffer — Use an AXI4 Video Frame Buffer block.
- AXI4 Random Access — Use an AXI4 Random Access Memory block.

These blocks provide a preconfigured and simplified view per each channel type.

For additional information about memory channel protocols, see “External Memory Channel Protocols”.

Region size (bytes) — Size of memory allocated for region, in bytes
`calculated`

This parameter is read-only.

The size in bytes of the region. This value is calculated as the number of buffers multiplied by buffer size.

Example: If Buffer size is 1024, and the number of buffers is set to 8, then Region size is 8192.

Buffer size (bytes) — Size of buffer, in bytes
1024 (default) | scalar

Specify the size in bytes of each buffer in the region.

The following rules apply when setting burst and buffer sizes.

1. The Burst Length of a given channel interface, calculated in bytes, must be less than 4096 bytes. To calculate the burst size in bytes, the channel interface scalar datatype is converted to bytes and then multiplied by the Burst Length.
2. The Burst Length can be set above 256, but will warn if generating to an AXI-based target platform. AXI-based memory systems have a maximum burst length of 256.

3. The Channel Length must be an integer multiple of burst length or the burst length must be an integer multiple of channel length. That is, it must be possible to either chunk the incoming channel data to a whole number of bursts or to gather a whole number of incoming channel data to a single burst.
4. The Buffer Size must be a whole number of bursts. This must be true for both the writer's burst size (after conversion of its Burst Length to bytes) and the reader's burst size (after conversion of its Burst Length to bytes).
5. The calculated number of bursts in a buffer must not exceed 5000. This is a temporary restriction based on the event processing internal to the memory model. This can happen with shared memory regions that have large buffer sizes (such as for 1080p video frames) and channel interfaces that specify smaller burst sizes. Generally, with larger frames, bursts sizes near the 4096 byte limit must be used.
6. The scalar datatype of the channel interface as converted to a flattened channel data width (i.e. *tdata* in the implementation) cannot exceed 128 bits.

The following table provides examples of good and bad parameter sets.

Burst and Buffer parameter examples

Channel Datatype	Channel Dimensions	Burst Length	Burst Size	Good / Bad	Why?
uint8	[1 1]	1024	2048	Good	This is a simple 8-bit data transaction.
uint8	[1 3]	1024	4096	Good	This might represent an RGB pixel from a Vision HDL Toolbox block. It is converted to 24-bit packed data and padded with 8 bits to become a 32-bit (4-Byte) *tdata* bus to the memory. The Burst size is 1024*4B = 4096B.
fixdt(0,10,0)	[1 3]	1024	4096	Good	This is converted to a 30-bit packed pixel with 2 bits of padding.
fixdt(0,12,0)	[1 3]	1024	8192	Good	This results in a 36-bit packed pixel which extends to 64-bit *tdata*. This data is compliant with the current limit of 128-bit *tdata*.
fixdt(0,48,0)	[1 3]	1024	8192	Bad	This results in a 144-bit packed pixel violates the current limit of 128-bit *tdata*.
uint8	[120 160 3]	1024	4096	Bad	The scalar data is 24-bit, padded to a 32-bit *tdata*. The Channel Length is 120*160=19200. The burst length of 1024 does not evenly divide 19200.
uint8	[120 160 3]	120	480	Good	The scalar data is 24-bit, padded to a 32-bit *tdata*. The Channel Length is 120*160, and since the burst length is 120, Channel length is 160 bursts in size. The buffer size is exactly 1 frame (120*160*4) as calculated in bytes.

Number of buffers — Number of buffers in region
8 (default) | integer

Divide the region into buffers. A disparate rate between a reader and a writer slows down the faster device. For example, a slow reader causes the writer to run out of buffers and block the writer,

effectively slowing down to the reader rate. Likewise, a slow writer causes the reader to run out of buffers and block the reader, effectively slowing it down to the writer rate.

- Specifying 1 – With a single buffer, access is controlled to ensure that a buffer is written, then it is read, then the next buffer is written, and so on.
- Specifying 2: With two buffers, memory access switches in a back-and-forth pattern. The writer writes the first buffer, then, while the reader is reading it, the writer can write the second buffer.
- Specifying *N* – With *N* buffers, the memory access has a ring-buffer pattern. The writer can continually write as long as buffers are available. When a buffer is completed, it becomes available for the reader. The writer and reader traverse the N buffers in a circular pattern. As long as the writer and reader maintain similar rates, the buffering prevents blockage.

Limitations

When you set the **Channel type** parameter to `AXI4-Stream to Software via DMA` or `Software to AXI4-Stream via DMA`, the **Number of buffers** parameter must be an integer from 3 to 64.

Burst length — Burst length for memory transactions
1 (default) | scalar

The length of bursts for this connection on the memory bus in units of scalar data. The scalar unit is the packed data type. Specify the burst size for both **Writer** and **Reader** access to the channel.

The channel data is always transferred to the memory model using burst transactions, regardless of the channel-type. For the AXI4 configuration, the algorithm-logic is responsible for defining the burst through the protocol signals.

For the streaming data configurations, the **Burst Length** parameter determines the burst size to the memory, and the channel **data** signal defines the size of each transfer on the interface.

When setting burst length, you must consider the “Buffer size (bytes)” on page 1-0 parameter.

Dependencies

To enable this parameter, select **Enable memory simulation**.

- This parameter is not visible when the **Channel type** parameter is set to `AXI4 Random Access`.
- The writer **Burst length** parameter is not visible when the **Channel Type** parameter is set to `Software to AXI4-Stream via DMA`
- The reader **Burst length** parameter is not visible when the **Channel Type** parameter is set to `AXI4-Stream to Software via DMA`

Use hardware board settings — Use the Hardware Implementation settings from the configuration parameters
on (default) | off

To use the same model-wide setting as in configuration parameters, select this box. Clear the box to customize the setting for this channel. When using channel-specific settings, values are still checked against hardware-specific constraints. For setting these values in the configuration parameters, see “FPGA design (mem channels)” on page 2-25.

Dependencies

To enable this parameter, select **Enable memory simulation**.

This parameter is not visible when **Channel type** is set to `AXI4 Random Access`.

Reader/Writer use same values — Reader and writer use the same values
on (default) | off

Select this box to use the same interconnect setting for the reader and the writer of this channel. Clear the box to customize different settings for the reader and the writer. Clearing the **Reader/Writer use same values** allows you to enter a value for the writer side and a value for the reader side, for the following parameters:

- **FIFO depth (number of bursts)**
- **Almost-full depth**
- **Clock Frequency (MHz)**
- **Data width (bits)**

Dependencies

To enable this parameter, select **Enable memory simulation**.

This parameter is visible when **Channel type** is set to `AXI4-Stream FIFO`, `AXI4-Stream Video FIFO`, or `AXI4-Stream Video Frame Buffer`.

FIFO depth (number of bursts) — Depth of FIFO for data
12 (default) | scalar

Specify depth of data FIFO, in units of bursts. When the writer has no buffers to write to, the FIFO can absorb data until a buffer becomes available. This value is the maximum number of bursts that can be buffered before data gets dropped.

Dependencies

- To enable this parameter, clear the **Use hardware board settings** check box and select **Enable memory simulation**.

  .
- When **Reader/Writer use same values** is cleared, there are two text boxes: one for **Writer** and one for **Reader**.
- This parameter is not visible when the **Channel type** parameter is set to `AXI4 Random Access`.
- The writer **FIFO depth** parameter is not visible when the **Channel Type** parameter is set to `Software to AXI4-Stream via DMA`
- The reader **FIFO depth** parameter is not visible when the **Channel Type** parameter is set to `AXI4-Stream to Software via DMA`

Almost full depth — Depth of FIFO when backpressure is asserted
8 (default) | scalar

Specify a number that asserts a backpressure signal from the channel to the data source. To avoid dropping data, set a high watermark, allowing the data producer enough time to react to backpressure. This number must be smaller than the FIFO depth.

Dependencies

- To enable this parameter, clear the **Use hardware board settings** check box and select **Enable memory simulation**.

- When **Reader/Writer use same values** is cleared, there are two text boxes: one for **Writer** and one for **Reader**.
- This parameter is not visible when the **Channel type** parameter is set to `AXI4 Random Access`.
- The writer **Almost full depth** parameter is not visible when the **Channel Type** parameter is set to `Software to AXI4-Stream via DMA`
- The reader **Almost full depth** parameter is not visible when the **Channel Type** parameter is set to `AXI4-Stream to Software via DMA`

Clock frequency (MHz) — Interconnect frequency of master datapath
`100` (default)

Frequency of the master datapath to the interconnect controller in MHz.

Dependencies

- To enable this parameter, clear the **Use hardware board settings** check box and select **Enable memory simulation**.
- When **Reader/Writer use same values** is cleared, there are two text boxes: one for **Writer** and one for **Reader**.
- This parameter is not visible when the **Channel type** parameter is set to `AXI4 Random Access`.
- The writer **Clock frequency (MHz)** parameter is not visible when the **Channel Type** parameter is set to `Software to AXI4-Stream via DMA`
- The reader **Clock frequency (MHz)** parameter is not visible when the **Channel Type** parameter is set to `AXI4-Stream to Software via DMA`

Data width (bits) — Data width of master datapath
`64` (default) | scalar

Data width of master datapath to interconnect controller in bits.

Dependencies

- To enable this parameter, clear the **Use hardware board settings** check box and select **Enable memory simulation**.
- When **Reader/Writer use same values** is cleared, there are two text boxes: one for **Writer** and one for **Reader**.
- When the **Channel type** parameter is set to `AXI4 Random Access`, the **Data width (bits)** parameter is set to the bit width corresponding to the **Data type** parameter, and the **Enable sample packing** parameter.
- The writer **Data width (bits)** parameter is not visible when the **Channel Type** parameter is set to `Software to AXI4-Stream via DMA`
- The reader **Data width (bits)** parameter is not visible when the **Channel Type** parameter is set to `AXI4-Stream to Software via DMA`

Signal Attributes

Write data signal

Dimensions — Dimensions of input data signal
scalar | array

wrData can be a multidimensional array. Specify the dimension for the array as a whole number.

When **Channel type** is set to `Software to AXI4-Stream via DMA`, the **Dimensions** parameter must be scalar.

Example: 1 – a scalar sample.

Example: [10 1] – a vector of ten scalars.

Example: [1080 1920 3] – a 1080p frame. The frame includes 1080 lines of 1920 pixels per line, and each pixel is represented by three values (for red, green and blue).

Data type — Data type of writer data
double | single | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | boolean | fixed point

Specify the data type of the **wrData** port. For help, click the **...** button. This expands the menu and shows a **Data Type Assistant**.

When the **Channel type** parameter is set to `Software to AXI4-Stream via DMA`, the data type must be set to `uint16`, `uint32`, `uint64`, or `fixdt(0,128,0)`.

Sample time — Time interval of sampling
1 (default) | positive scalar | vector

Specify a time interval in seconds to define how often the block updates.

Specify the **Sample time** parameter as a scalar when you do not want the output to have a time offset. To add a time offset to the output, specify the **Sample time** parameter as a 1-by-2 vector where the first element is the sampling period and the second element is the offset. For more information about sample times in Simulink, see “Specify Sample Time”.

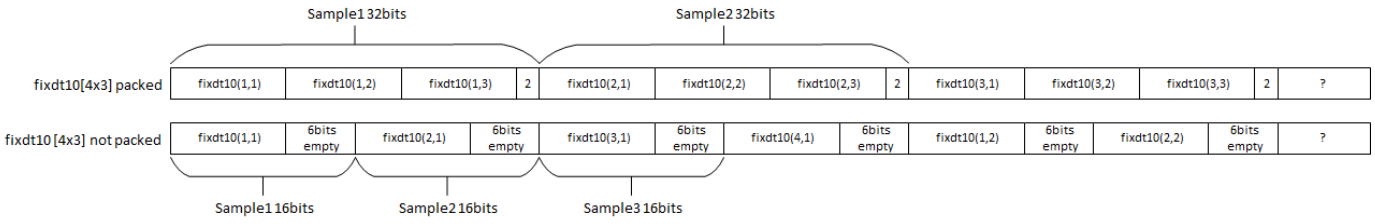
Dependencies

This parameter is not visible when the **Channel type** parameter is set to `Software to AXI4-Stream via DMA`.

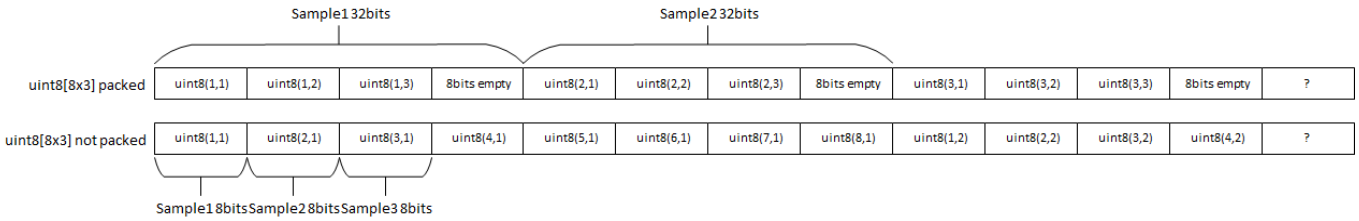
Enable sample packing (last signal dimension as channel) — Pack data on the last dimension of the signal
`off` (default) | `on`

Select this parameter to enable data packing across the last dimension of the signal. The Memory Channel block packs the data along the last dimension of the signal. For example, if the channel data type is `uint32`, the dimensions are `[1024 4]`, and if you select this parameter, then the memory channel generates 1024 read or write transactions of 128 bits. For this example, if you clear this sample packing parameter, the memory channel generates 4096 transactions of 32 bits each.

This figure shows how data is aligned for a signal with data type `fixdt10[4x3]`. When the data is packed, three 10-bit words are concatenated and extended by 2 bits to a 32-bit word. When the data is not packed, each 10-bit word is extended to a 16-bit word.



This figure shows how data is aligned for a signal with data type `uint8[8x3]`. When the data is packed, three 8-bit words are concatenated and extended by 8 bits to a 32-bit word. When the data is not packed, each 8-bit word is represented as an 8-bit sample.



The combined width of the flattened signal must not exceed 512 bits.

Dependencies

This parameter is not visible when the **Channel type** parameter is set to `Software to AXI4-Stream via DMA`.

Read data signal

Output data signal matches input — Reader and writer use the same values
on (default) | off

Select this box to use the same dimensions and data type for the reader and the writer of this channel. Clear the box to customize different settings for the reader and the writer. Clear the box to customize different dimensions and data type for the reader and writer interfaces.

Dimensions — Dimensions of output data signal
scalar | array

rdData can be a multidimensional array. Specify the dimension for the array as a whole number.

When **Channel type** is set to `AXI4-Stream to Software via DMA`, the **Dimensions** parameter must be scalar.

Example: 1 – a scalar sample.

Example: [10 1] – a vector of ten scalars.

Example: [1080 1920 3] – a 1080p frame. The frame includes 1080 lines of 1920 pixels per line, and each pixel is represented by three values (for red, green and blue).

Dependencies

To enable this parameter, clear the **Output data signal matches input** check box.

Data type — Data type of reader data
double | single | int8 | int16 | int32 | int64 | uint8 | uint16 | uint32 | uint64 | boolean | fixed point

Specify the data type of the **rdData** port. For help, click the **...** button. This expands the menu and shows a **Data Type Assistant**.

When the **Channel type** parameter is set to `AXI4-Stream to Software via DMA`, the data type must be set to `uint16`, `uint32`, `uint64`, or `fixdt(0,128,0)`.

Dependencies

To enable this parameter, clear the **Output data signal matches input** check box.

Sample time — Time interval of sampling
1 (default) | positive scalar | vector

Specify a time interval in seconds to define how often the block updates.

Specify the **Sample time** parameter as a scalar when you do not want the output to have a time offset. To add a time offset to the output, specify the **Sample time** parameter as a 1-by-2 vector where the first element is the sampling period and the second element is the offset. For more information about sample times in Simulink, see “Specify Sample Time”.

Dependencies

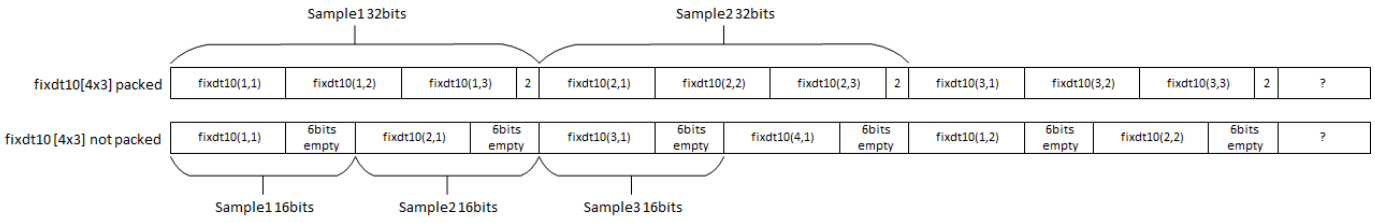
To enable this parameter, do one of the following:

- Set **Channel type** to `Software to AXI4-stream via DMA`.
- Set **Channel type** to `AXI4 Random Access` and clear the **Output data signal matches input** check box.

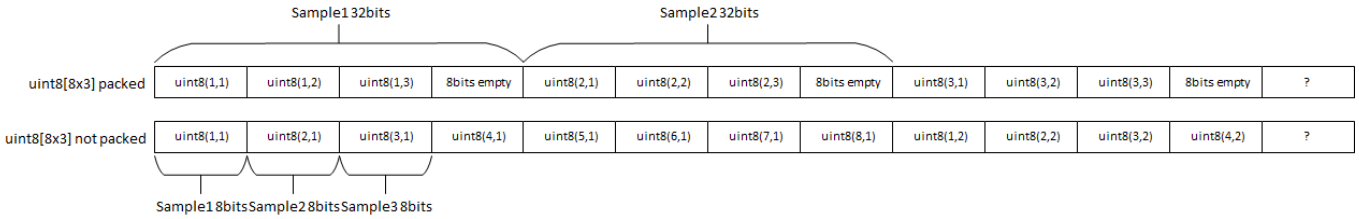
Enable sample packing (last signal dimension as channel) — Pack data on the last dimension of the signal
`off` (default) | `on`

Select this parameter to enable data packing across the last dimension of the signal. The Memory Channel block packs the data along the last dimension of the signal. For example, if the channel data type is `uint32`, the dimensions are `[1024 4]`, and if you select this parameter, then the memory channel generates 1024 read or write transactions of 128 bits. For this example, if you clear this sample packing parameter, the memory channel generates 4096 transactions of 32 bits each.

This figure shows how data is aligned for a signal with data type `fixdt10[4x3]`. When the data is packed, three 10-bit words are concatenated and extended by 2 bits to a 32-bit word. When the data is not packed, each 10-bit word is extended to a 16-bit word.



This figure shows how data is aligned for a signal with data type `uint8[8x3]`. When the data is packed, three 8-bit words are concatenated and extended by 8 bits to a 32-bit word. When the data is not packed, each 8-bit word is represented as an 8-bit sample.



The combined width of the flattened signal must not exceed 512 bits.

Dependencies

To enable this parameter, clear **Output data signal matches input** check box, and set **Channel type** to a value other than `AXI4-Stream to Software via DMA`.

Inherit sample time offset — Make reader inherit sample time from writer
`on` (default) | `off`

Select this box to make the reader inherit the sample time offset from the writer. Clear the box to use a different sample time offset from the writer. For more information about sample time offsets, see Sample Time Offset.

Dependencies

To enable this parameter, clear the **Output data signal matches input** check box and set **Channel type** to one of the following values:

- `AXI4-Stream FIFO`
- `AXI4-Stream Video FIFO`
- `AXI4-Stream Video Frame Buffer`

Use pixel clock sample times — Use the pixel clock sample time
on (default) | off

Select this box to use the pixel clock sample time. To use the pixel clock sample time, you must use scalar pixel dimensions. It is only relevant when streaming pixels. If both the reader and the writer are streaming frames, you get an error when checking this box.

Note If both reader and writer are using framed signals, the signal dimensions are not scalar and pixel timing cannot be inferred. Selecting **Use pixel clock sample times** in this case creates an error.

Dependencies

To enable this parameter, set **Channel type** to `AXI4-Stream Video FIFO` or `AXI4-Stream Video Frame Buffer`.

Frame size — Frame dimensions
`480p SDTV (720x480p)` (default) | ...

For video-streaming applications, **Frame size** can often be inferred, and this parameter shows as a read-only value. When it cannot be inferred, select the **Frame size** from a drop-down menu.

- When the reader or the writer are using framed signals of a frame with known porch and blanking timings, the **Frame size** is inferred from those timings. When the reader or the writer is a scalar and the other is a non-standard frame size, the **Frame size** cannot be inferred and you get an error.
- When **Channel type** is set to `AXI4-Stream Video Frame Buffer` and both reader and writer are using scalar dimensions for pixel streams, **Frame size** is inferred from **BufferSize** and TDATA and it is then a read-only value.
- When **Channel type** is set to `AXI4-Stream Video FIFO` and both reader and writer are using scalar dimensions for pixel streams, select the **Frame size** as one of these values:
  - `160x120p`
  - `480p SDTV (720x480p)`
  - `576p SDTV (720x576p)`
  - `720p HDTV (1280x720p)`
  - `1080p HDTV (1920x1080p)`
  - `320x240p`
  - `640x480p`
  - `800x600p`
  - `1024x768p`
  - `1280x768p`
  - `1280x1024p`
  - `1360x768p`
  - `1400x1050p`
  - `1600x1200p`
  - `1680x1050p`
  - `1920x1200p`
  - `16x12p (test mode)`

Dependencies

To enable this parameter, set **Channel type** to `AXI4-Stream Video FIFO` or `AXI4-Stream Video Frame Buffer`, and select **Use pixel clock sample times**.

Performance

Launch performance plots — Display performance metrics
button

Clicking the button opens Performance plots for the memory channel in a new window. For more information about performance graphs, see “Simulation Diagnostics”.

Dependencies

To enable this parameter, select **Enable memory simulation**.

Version History

Introduced in R2019a

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

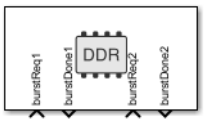
Memory Controller | AXI4-Stream to Software | Software to AXI4-Stream | AXI4 Video Frame Buffer | AXI4 Random Access Memory | Memory Traffic Generator | Interrupt Channel

Topics

“External Memory Channel Protocols”

Memory Controller

Arbitrate memory transactions for one or more Memory Channel blocks

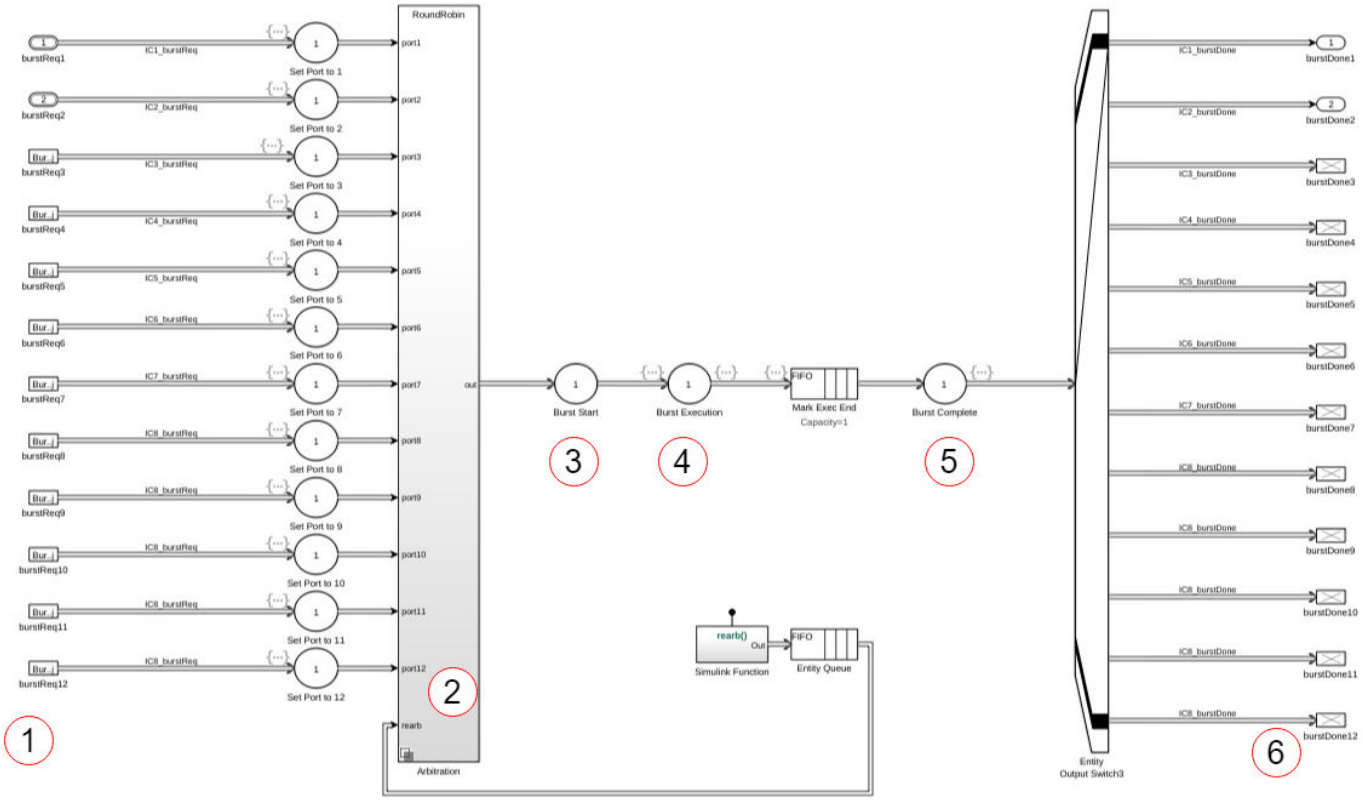


Libraries:
SoC Blockset / Memory

Description

The Memory Controller block arbitrates between masters and grants them unique access to shared memory. Configure this block to support multiple channels with various arbitration protocols. The Memory Controller block is also instrumented to log and display performance data, enabling you to debug and understand the performance of your system at simulation time.

The following image shows the implementation of the Memory Controller block.



The numbers in the image represent different latency stages of the block.

1. A burst-request enters the block.
2. The request may be delayed by arbitration until it is granted access to the bus. Set the arbitration policy in “Interconnect arbitration” on page 1-0 .

3. If your model requires an additional delay before the first transfer starts, set that value in "Request to first transfer (in clocks)" on page 1-0 .
4. The burst execution latency is calculated by the burst size, the data-width, the clock frequency, and the "Bandwidth derating (%)" on page 2-23 value.
5. If your model requires a delay from burst completion until a burst response is issued to the channel, set that value in "Last transfer to transaction complete (in clocks)" on page 1-0 .

The memory controller has an internal state, which is visible when using a **Logic Analyzer** to view simulation and execution metrics. The state values are:

- `BurstIdle`: At start of simulation, before the block receives a burst request.
- `BurstRequest`: A burst request enters the block.
- `BurstAccepted`: The arbiter accepted the burst for processing.
- `BurstExecuting`: A burst is executing.
- `BurstDone`: A burst request is done executing.
- `BurstComplete`: A burst response is done, and the burst is complete. The **burstDone** signal is now sent to the master.

For information about visualizing memory controller latencies, see "Memory Controller Latency Plots".

Limitations

- Soc models do not support backward stepping. For more information on simulation stepping, see "Debug Simulations in the Simulink Editor".
- The following memory blocks already include a memory controller:
  - AXI4 Random Access Memory
  - Software to AXI4-Stream
  - AXI4-Stream to Software
  - AXI4 Video Frame Buffer

  Therefore, when you use one of these blocks, you will get an error if you also add a Memory Controller block to the design.

Ports

Input

burstReq*N* — Request for memory access
scalar

This port receives requests for memory access as messages. Connect this input port to one of the burst request message ports (`wrBurstReq` or `rdBurstReq`) from a Memory Channel or Memory Traffic Generator block. For more information on messages, see "Messages".

The number of **burstReq*N*** input ports is defined by the **Number of masters** parameter. **burstReq*N*** represents the *N*th input port.

Data Types: `BurstRequest2BusObj`

Output

burstDone*N* — Signal toward master
scalar

After a master is granted access to the memory and the burst transaction has completed, this port sends a message that the transaction completed. Memory access is then given to the next master according to the arbitration scheme. For more information on messages, see “Messages”.

The number of **burstDone*N*** output ports is defined by the **Number of masters** parameter. **burstDone*N*** represents the *N*th input port

Data Types: `BurstRequest2BusObj`

Parameters

Hardware board — View or modify current hardware settings
name of current hardware board

This parameter is read-only.

This parameter shows a link to the selected hardware board. Click the link to open the configuration parameters, and adjust the settings or choose a different board.

To learn more about configuration parameters for the memory controller, see “FPGA design (PS mem controllers)” on page 2-6.

Main

Memory selection — Choose between memory regions
`PS memory` (default) | `PL memory`

Select between processing subsystem (PS) or programming logic (PL) memory.

- If the selected board supports only a PL memory, then the default value is `PL memory`.
- If the selected board supports only PS memory or only PL memory, then this parameter is read-only.
- If the selected board is not a supported SoC board, then this parameter is not visible.

Number of masters — Number of masters connected to this controller
2 (default) | positive integer

Set this parameter to generate the interface accordingly, and specify how many masters connect to the memory.

Advanced

Interconnect arbitration — Arbitration policy
`Round robin` (default) | `Fixed port priority`

Set the arbitration policy for the memory-interconnect block. When multiple masters request for memory access, the policy is determined by the value of this parameter.

- `Round robin` sets a fair arbitration based on last service time.

- `Fixed port priority` sets a fixed priority of **burstReq1**, **burstReq2**, **burstReq3**, and so on, where **burstReq1** gets the highest priority.

Use hardware board settings — Use hardware implementation settings from the configuration parameters
`off` (default) | `on`

Select this parameter to use the same model-wide settings as set in the configuration parameters. Clear this parameter to customize the settings for this memory controller. When using customized settings, values are still checked against hardware-specific constraints. For more information, see “FPGA design (mem controllers)” on page 2-23.

Bandwidth — Bandwidth for transactions towards external memory
scalar

This parameter is read-only.

This value shows the calculated bandwidth between the memory controller and the external memory. It is calculated as **Frequency (MHz)** multiplied by **Data width (bits)**.

Frequency (MHz) — Controller clock frequency, in MHz
`200` (default) | scalar

The clock rate of the bus used to drive interactions with the external memory. The controller frequency determines the overall system bandwidth for external memory that must be shared among all the masters in the model.

Dependencies

To enable this parameter, clear the **Use hardware board settings** parameter.

Data width (bits) — Bit width of datapath
`64` (default) | positive integer

Set the width, in bits, of the datapath between the memory controller and the memory interconnect.

Dependencies

To enable this parameter, clear the **Use hardware board settings** parameter.

Bandwidth derating (%) — Memory transaction inefficiencies
0-100

Model memory transaction inefficiencies specified by a derating percentage value. For every 100 clocks, memory transaction execution is paused for the number of clocks equal to **Bandwidth derating**. To set this parameter, measure the maximum bandwidth on your board and reflect the bandwidth derating from your board in this parameter. See an example in “Analyze Memory Bandwidth Using Traffic Generators”.

Dependencies

To enable this parameter, clear the **Use hardware board settings** parameter.

Request to first transfer (in clocks) — Number of clock cycles between request and start of transfer
nonnegative integer

Specify the delay, in clock cycles, between a read or write request and the start of a transfer. Specify nonnegative integer values in both **Write** and **Read** boxes.

This delay is the number of clock cycles between making a request to the memory controller and until it returns a response. It is reflected in the **Logic Analyzer** waveforms as the time that the memory controller state remains as `BurstAccepted`. For more information about viewing waveforms in simulation, see “Buffer and Burst Waveforms”.

To set this value, measure the clock cycles between the burst-request and start of transfer on your board. For instructions for extracting this information from a hardware execution, see “Configuring and Querying the AXI Interconnect Monitor”.

Dependencies

To enable this parameter, clear the **Use hardware board settings** parameter.

Last transfer to transaction complete (in clocks) — Number of clock cycles between the end of transfer and completion of transaction
nonnegative integer

Specify the delay in clock cycles between the end of a memory transfer and the end of a transaction. Specify nonnegative integer values in both **Write** and **Read** boxes.

To set this value, measure the clock cycles between the end of the burst and the completion of the transaction on your board. For instructions for extracting this information from a hardware execution, see “Configuring and Querying the AXI Interconnect Monitor”.

Dependencies

To enable this parameter, clear the **Use hardware board settings** parameter.

Performance

Click **Launch performance app** to open the Performance Metrics window. For additional information, see “Simulation Performance Plots”.

Version History

Introduced in R2019a

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

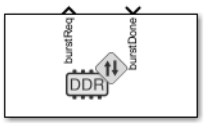
See Also

Memory Channel | Memory Traffic Generator | Register Channel

Topics
“External Memory Channel Protocols”

Memory Traffic Generator

Generate traffic towards memory controller



Libraries:
SoC Blockset / Memory

Description

When connected to a memory controller, the Memory Traffic Generator block generates read or write requests to the memory, acting as a master. Use this block to model the impact that a master's memory accesses has on your algorithm without explicitly simulating the behavior of that master. You can also use the Memory Traffic Generator block to characterize performance of your memory subsystem under varying levels of memory access contention.

Note To model memory contention, the Memory Traffic Generator block gains memory access, competes in arbitration, and releases access. The Memory Traffic Generator block does not actively read or write from memory.

Ports

Input

burstDone — End of burst and access to memory
scalar

This message port receives control messages from a connected Memory Controller block that the requested burst transaction completed. Connect the **burstDone** output signal from the Memory Controller block to this port. For more information on messages, see "Messages".

Dependencies

This port is visible when you select the **Show Memory Controller ports** parameter.

Data Types: `BurstRequest2BusObj`

Output

burstReq — Request memory access from memory controller
scalar

This message port sends a message requesting burst access from the memory controller. Connect this port to the **burstReq** input port of the Memory Controller block. For more information on messages, see "Messages".

Dependencies

This port is visible when you select the **Show Memory Controller ports** parameter.

Data Types: `BurstRequest2BusObj`

Parameters

Show Memory Controller ports — Show input and output ports
`on` (default) | `off`

Select this option when using the Memory Traffic Generator block in a system with a Memory Channel block and a Memory Controller block.

Clear this option when using a Memory Traffic Generator block with one of these specialized memory blocks:

- AXI4 Random Access Memory
- Software to AXI4-Stream
- AXI4-Stream to Software
- AXI4 Video Frame Buffer

When you use a specialized memory block in your design, that block includes a memory controller which the Memory Traffic Generator block communicates with implicitly and ports are not necessary.

Memory selection — Choose between memory regions
`PS memory` (default) | `PL memory`

Select between processing subsystem (PS) or programming logic (PL) memory.

- If the selected board supports only a PL memory, then the default value is `PL memory`.
- If the selected board supports only PS memory or only PL memory, then this parameter is read-only.
- If the selected board is not a supported SoC board, then this parameter is not visible.

Dependencies

To enable this parameter, clear the **Show Memory Controller ports** parameter.

Request type — Choose between write or read request
`Writer` (default) | `Reader`

Choose between a write or read request type for the block to generate. Specify `Writer` or `Reader`, respectively.

Total burst requests — Number of burst requests to generate
`100` (default) | integer greater than 1

Generate recurring traffic patterns by setting this value to an integer greater than one.

Burst size (bytes) — Size of generated burst transactions
`256` (default) | scalar

Specify the size of each burst transaction in bytes. This parameter, along with the width of the datapath (as configured in the configuration parameters), controls the burst length.

For example, if burst size is 256 bytes, and the Memory Channel block is configured with **Data width (bits)** set to 64 (8 bytes), then **Burst length** is calculated as 256/8 = 32.

Time between bursts (s) — Simulation time between burst requests
`1e-6` (default) | time, in seconds

Specify simulation time between burst requests, in seconds.

Dependencies

To enable this parameter, clear the **Allow simulation only parameters** parameter.

Tip If you cleared **Allow simulation only parameters** and this parameter is not visible – click **Apply** at the bottom of the Block Parameters dialog box.

Allow simulation only parameters — Configure additional parameters for simulation only
`on` (default) | `off`

Select this parameter to enable configuration of simulation-only parameters.

First burst time — Simulation time for initial burst request
`10e-6` (default) | time, in seconds

Specify simulation time, in seconds, for sending the initial burst request. This value must be a positive real scalar.

Dependencies

To enable this parameter, select **Allow simulation only parameters** parameter.

Random time between bursts (s) — Range of simulation time for recurring requests
`[1e-6 1e-6]` (default) | vector of the form [*min max*]

Specify the range of simulation time between burst requests with a vector of the form [*min max*].

- *min* is the minimum time, in seconds, between recurring requests.
- *max* is the maximum time, in seconds, between recurring requests.

min and *max* must be nonnegative, and *max* must be greater than *min*.

To specify a deterministic rate, set the minimum and maximum time between requests to the same value. If you want reproducible randomization, specify a seed in the configuration parameters, on the **Hardware Implementation** pane. For more information on setting the seed value, see “Simulation Settings” on page 2-4.

Dependencies

To enable this parameter, select the **Allow simulation only parameters** parameter.

Wait for burst done — Wait for burst-done signal before generating next request
`off` (default) | `on`

Select this parameter to wait for a burst-done signal from the previous burst before generating the next burst request. Clear this parameter to generate burst requests regardless of other master traffic. To get a known data rate, clear this parameter.

Enable assertion — Enable verbose information
`off` (default) | `on`

Select this parameter to view diagnostic messages when the Traffic Generator block drops a packet. Clearing this parameter enhances simulation performance.

View performance plots — Display performance metrics
button

Clicking the button opens the **Performance Plots for Memory Controller** window. You can then select to plot bandwidth, bursts, or latencies. For more information about performance graphs, see "Memory Controller Latency Plots".

Dependencies

To enable this parameter, clear the **Show Memory Controller ports** parameter.

Version History

Introduced in R2019a

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See "Generate SoC Design".

Fixed-Point Conversion

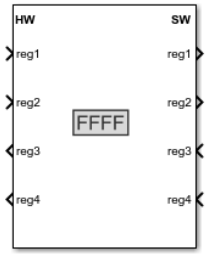
Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Memory Channel | Memory Controller | AXI4-Stream to Software | Software to AXI4-Stream | AXI4 Video Frame Buffer | AXI4 Random Access Memory | Register Channel

Register Channel

Timing model for transfer of register values



Libraries:
SoC Blockset / Memory

Description

The Register Channel block provides a timing model for the transfer of register values between a processor and hardware logic. The register channel represents the datapath between a processor and a hardware IP via a common configuration bus. Configure the block to include one or more registers, and configure the direction for each register as write if the processor writes to it, or read if the processor reads from it.

Ports

Input

regN — Register input
scalar

Each register is assigned a port pair: an input and an output. You can configure the processor to be a writer or a reader. If the register is a read register, then the input comes from the hardware (HW) side. If the register is a write register, the input comes from the software (SW) side. By default, the Nth register port is named reg***N***. You can change a register name by clicking **Edit** in the **Registers** parameter dialog box.

Dependencies

The number of input ports depends on the number of registers in the register table.

Output

regN — Register output
scalar

Each register is assigned a port pair: an input and an output. You can configure the processor to be a writer or a reader. If the register is configured as a read register, then the output goes to the software (SW) side. If the register is a write register, the output goes to the hardware (HW) side. By default, the Nth register port is named reg***N***. You can change a register name by clicking **Edit** in the **Registers** parameter dialog box.

Dependencies

The number of output ports depends on the number of registers in the register table.

Parameters

Registers — Edit register name, direction, data type, and dimension
table

This parameter includes a table, where each of its lines corresponds to a register in your IP. Edit the table to add or edit a register configuration, up to 32 registers.

For each register, you can edit these values:

- **Register Name** – Specify the register name. This changes the input and output ports for this register.
- **Direction** – Choose `write` if the processor writes the register. Choose `read` if the processor reads the register.
- **Data Type** – Select the data type for the register. Supported data types are
  - `single`
  - `int8`
  - `uint8`
  - `int16`
  - `uint16`
  - `int32`
  - `int64`
  - `uint32`
  - `uint64`
  - `boolean`
  - `fixdt(1,16,0)`
  - `fixdt(1,16,2^0,0)`
  - `fixed point`
- **Dimension** – Select the vector size of the register. The default value is 1.

Register write sample time — Sample time for register access
`-1` (default) | two element vector

This sample time represents the clock period on the hardware side. Specify an offset time by entering a two-element vector for discrete blocks or configurable subsystems. The first element is the sample time, and the second element is the offset time. For example, an entry of [1.0 0.1] specifies a 1.0-second sample time with a 0.1-second offset. If no offset is specified, the default offset is zero.

When the value is `-1`, the block inherits its sample time value from the model.

Note When the **Direction** of a register is set to `Write`, it implies that software is the writer and hardware is the reader, but **Register write sample time** determines the sample time of the signal on the hardware side.

Version History

Introduced in R2019a

Extended Capabilities

HDL Code Generation

Generate Verilog and VHDL code for FPGA and ASIC designs using HDL Coder™.

To automatically generate HDL code for your design, and execute on an SoC device, use the **SoC Builder** tool. See “Generate SoC Design”.

Fixed-Point Conversion

Design and simulate fixed-point systems using Fixed-Point Designer™.

See Also

Memory Channel | Memory Traffic Generator | Memory Controller

Topics

“Memory and Register Data Transfers”

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”
“Integrate MCU Scheduling and Peripherals in Motor Control Application”

External Websites

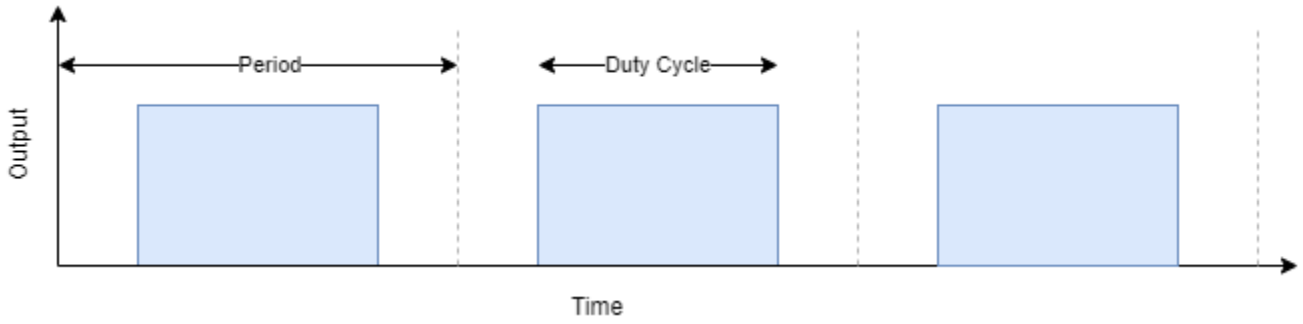
https://en.wikipedia.org/wiki/Analog-to-digital_converter

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.

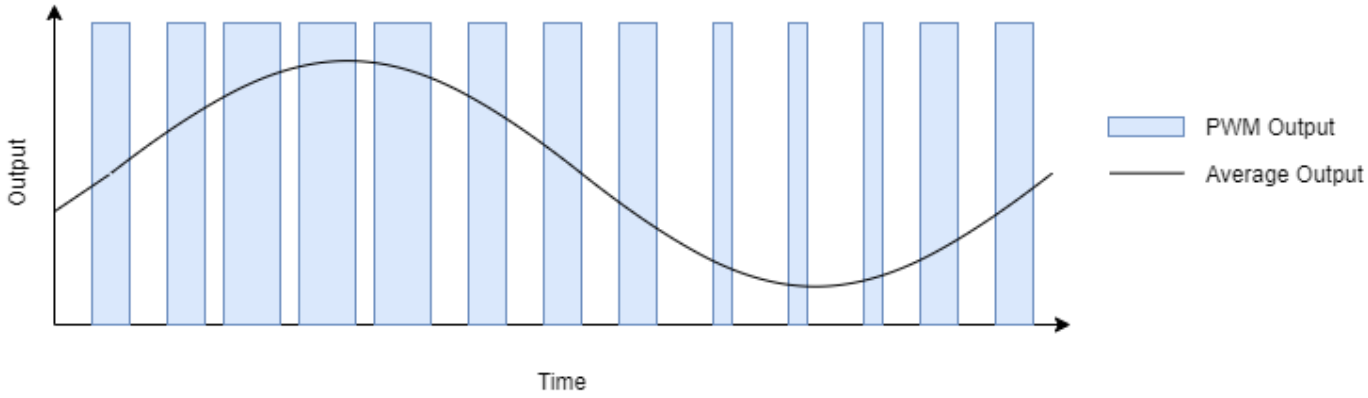
Dependencies

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%.

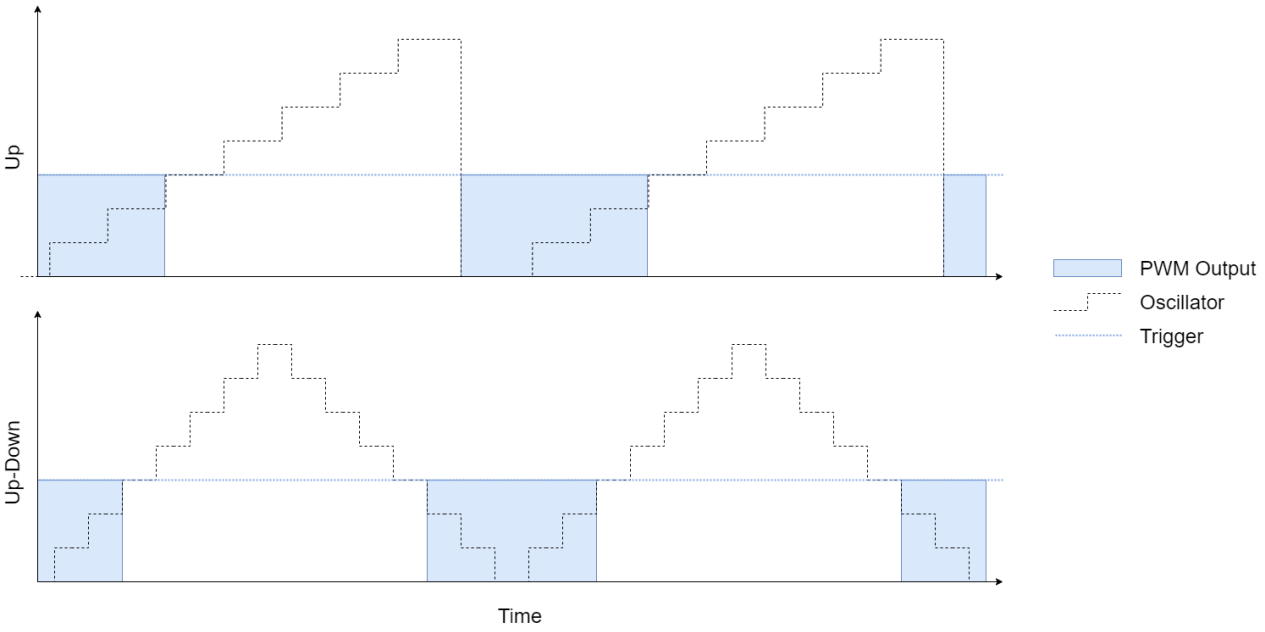
Output mode — Output mode
`Switching` (default) | `Average`

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`

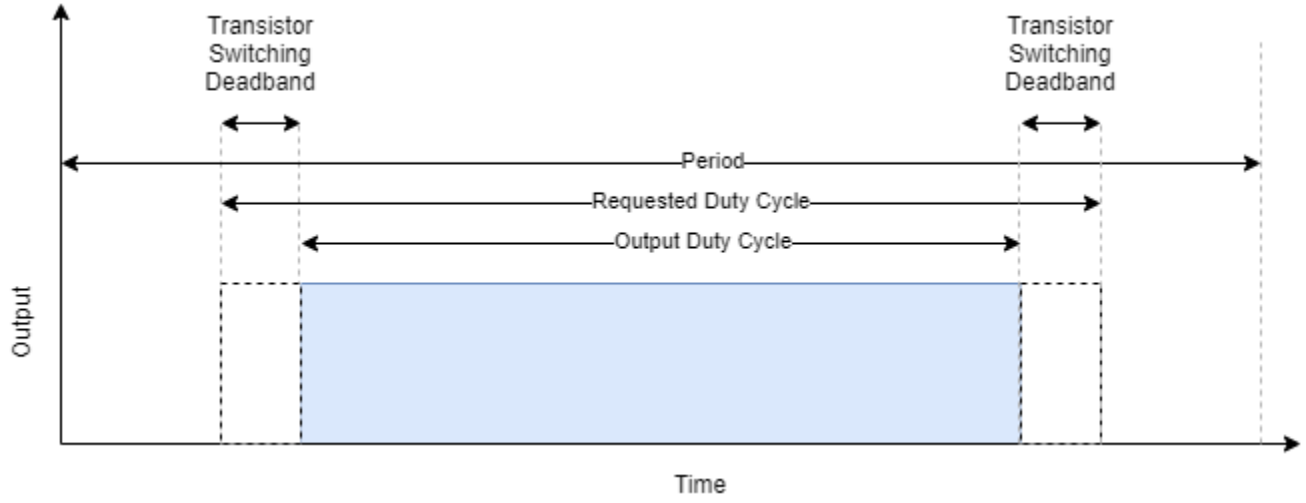
Sampling mode — Sampling mode
`End of PWM period` (default) | `Mid or End of PWM period` | `Immediate (at compare matches)`

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`

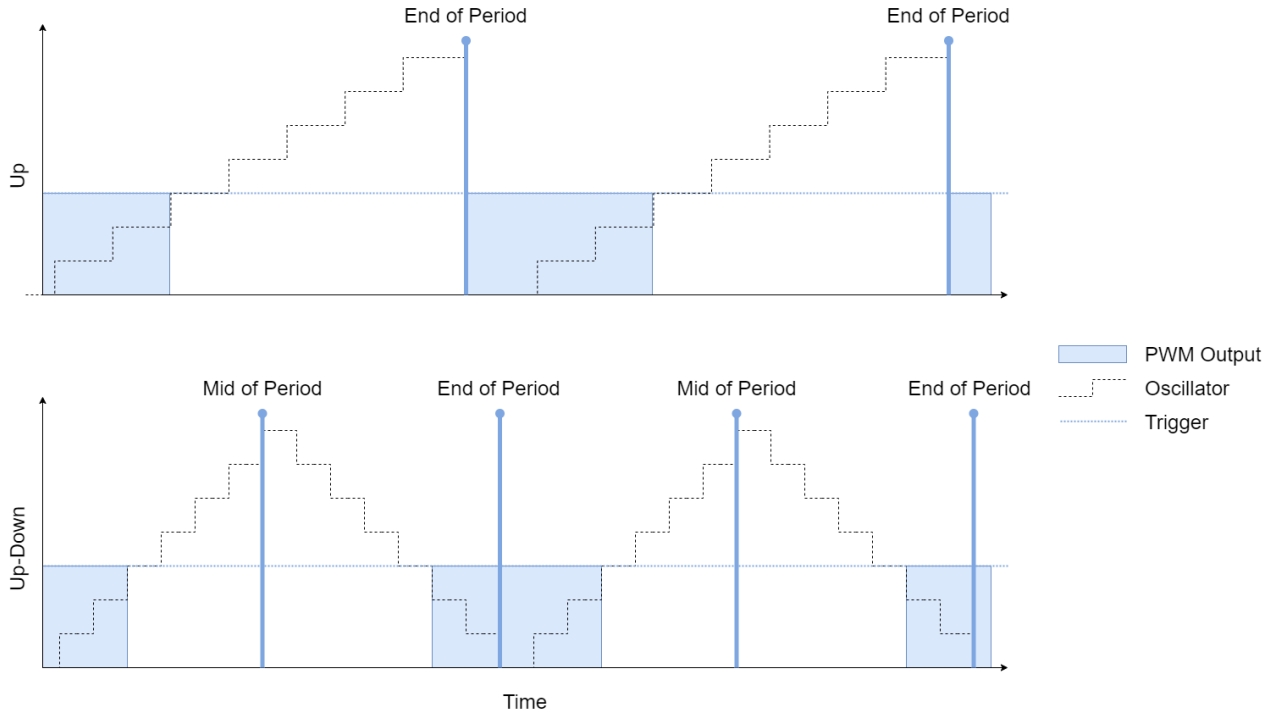
Specify the types of events on which to generate events. When the **Type**value is set to:

- `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.

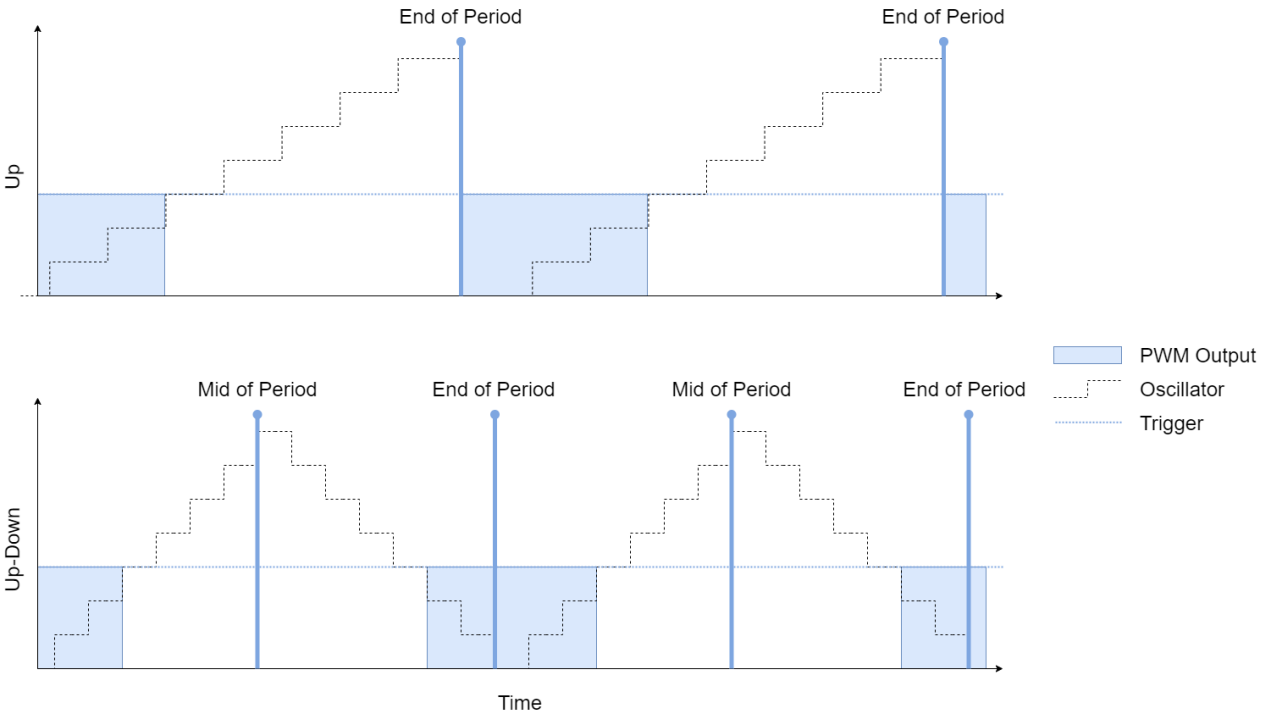
Example: 4

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

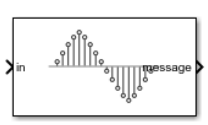
“Get Started with Multiprocessor Blocks on MCUs”
“Integrate MCU Scheduling and Peripherals in Motor Control Application”

External Websites

https://en.wikipedia.org/wiki/Pulse-width_modulation

Digital IO Interface

Simulate digital input and output pins on processor



Libraries:
SoC Blockset / Peripherals

Description

The Digital IO Interface block simulates the behavior of a GPIO processor pin on a hardware board. You can configure the block as either an input digital block or an output digital block.

When you configure this block as an input, the block captures the crossing event of an input signal on the pin. When the crossing event occurs, the block emits the logical state of the pin as a signal and sends an event to a Task Manager block. A task can then read the logical value using an IO Data Sink block.

When you configure this block as an output, the block emits the logical signal from a connected IO Data Source block.

Ports

Input

msg — SoC message data
scalar

This port receives the logical signal from the **msg** port of a connected IO Data Source block.

Dependencies

To enable this port, set the **IO direction** parameter to `Output`.

Data Types: `SoCData`

in — Input digital signal
scalar

This port specifies the input digital voltage signal to convert into a digital logical measurement.

Dependencies

To enable this port, set the **IO direction** parameter to `Input`.

Data Types: `single` | `double`

Output

out — Output digital signal
scalar

This port outputs the logical signal value provided from the **msg** input port.

Dependencies

To enable this port, set the **IO direction** parameter to `Output`.

Data Types: `single` | `double`

event — Task event signal
scalar

This port sends a message at each digital IO threshold event. This output connects to the input of the Task Manager block to execute the associated event-driven task after executing the IO event.

Dependencies

To enable this port, set the **IO direction** parameter to `Input` and select the **Enable event** parameter.

Data Types: `rteEvent`

message — SoC message data
scalar

This port sends the Digital IO Interface input signal data as a message to the **msg** input port of the IO Data Sink block.

Dependencies

To enable this port, set the **IO direction** parameter to `Input`.

Data Types: `SoCData`

Parameters

IO direction — Configure block as digital input or output
`Input` (default) | `Output`

Select a value for the Digital IO Interface block to act as a digital input or a digital output.

Low input voltage — Low input voltage threshold
`0.2` (default) | nonnegative scalar

Specify the lower threshold voltage of the input signal. When the signal value crosses below the threshold, the block registers this crossing as an input change, generating an event and new message with a digital value of `0`.

Example: `0.3`

High input voltage — High input voltage threshold
`2.5` (default) | nonnegative scalar

Specify the upper threshold voltage of the input signal. When the signal value crosses above the threshold, the block registers this crossing as an input change, generating an event and new message with a digital value of `1`.

Example: `0.3`

Enable event — Option to enable event generation
`on` (default) | `off`

Select this parameter for the Digital IO Interface block to generate an event following a digital transition event. Selecting this parameter also enables the **event** output port, which you can connect to a Task Manager to simulate asynchronous digital read operation.

Condition — Condition on when to trigger event
`Rising edge` (default) | `Falling edge` | `Both`

Select the condition for when to generate the digital input event.

- `Rising edge` — Generate an event when the input signal on the **in** input port is greater than the **High input voltage** parameter value.
- `Falling edge` — Generate an event when the input signal on the **in** input port is less than the **Low input voltage** parameter value.
- `Both` — Generate an event on both conditions of the `Rising edge` and `Falling edge` options.

Sample time — Sample time
`-1` (default) | positive scalar

Specify the sample time (in seconds) to define the rate at which to output the message containing the digital input data.

Version History

Introduced in R2021b

See Also

IO Data Sink | IO Data Source | Task Manager

External Websites
https://en.wikipedia.org/wiki/General-purpose_input/output

Audio Capture Interface

Simulate capture of sample from audio stream on hardware

Libraries:
SoC Blockset / Peripherals

Description

The Audio Capture Interface block simulates the behavior of an audio input device. The block outputs audio samples as a message to a connected Audio Capture block that simulates audio driver code and connects to your algorithm. The block can also emit an event that, when connected to a Task Manager block, can trigger a task containing the Audio Capture block.

Ports

Input

data — Data frame from captured audio
M-element vector | *M*-by-*C* matrix

To provide a single audio channel, specify this data as an *M*-element audio data frame, where *M* defines the number of samples of audio data per frame. To provide multiple audio channels provided, specify data as an *M*-by-*C* matrix, where *C* defines the number of channels.

Dependencies

To enable this port, set the **Input** parameter to `From input port`.

Data Types: `int8` | `int16` | `int32`

Output

event — Task event signal
scalar

This port sends a message after a frame of audio samples is captured for all channels. This output connects to the input of the Task Manager block to execute the associated event-driven task.

Dependencies

To enable this port, select the **Show event port** parameter.

Data Types: `rteEvent`

msg — SoC message data
scalar

This port outputs data messages containing audio data to a connected Audio Capture block.

Data Types: `SoCData`

Parameters

Input — Source of simulated audio data
`From dialog` (default) | `From input port` | `From timeseries object`

Specify the source of the simulated audio data.

Value — Audio data to be output
`int16([1 2 3 4 5 6 7 8])` (default) | *CN*-length vector

Specify the audio data as a *CN*-length vector representing the audio frames for all channels, where *N* is the number of samples per frame and *C* is the number of audio channels. The samples for each channel are contiguous.

Object name — Timeseries object
`[ ]` (default) | `MATLAB workspace variable`

Specify the audio data as a timeseries object defined in the MATLAB workspace.

Sample time — Sample time in seconds
`-1` (default) | positive scalar

Enter the sample time defining the rate at which to output the message containing audio data.

Show event port — Option to enable task event ports
`off` (default) | `on`

Select this parameter to enable an event port that, when connected to the Task Manager block, can execute event-driven tasks.

Version History

Introduced in R2021a

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”.

Embedded Coder does not generate code for this block. In the generated code, the advanced Linux sound architecture (ALSA) driver framework performs audio capture on the device. The related Audio Capture block represents the ALSA driver. For more information on the ALSA driver framework, see the Advanced Linux Sound Architecture website.

See Also

Hardware Mapping | Audio Capture | Audio Playback Interface | Audio Playback

External Websites
Advanced Linux Sound Architecture

Audio Playback Interface

Simulate audio output device on a hardware board

Libraries:
SoC Blockset / Peripherals

Description

The Audio Playback Interface block simulates the behavior of an audio output device. The block receives audio samples as messages from a connected Audio Playback block that simulates audio driver code connected to your algorithm.

Ports

Input

msg — SoC message data
vector

This port receives data messages containing audio data from a connected Audio Playback block.

Data Types: `SoCData`

Output

data — Audio data frame
M-element vector | *M*-by-*C* matrix

When the block receives a single audio channel, the audio data frame is an *M*-element vector from a simulated hardware audio output. The **Samples per frame** parameter defines the number of samples, *M*, of audio data. When the block receives multiple audio channels, the audio data frame is an *M*-by-*C* matrix, where *C* is specified by the **Number of channels** parameter.

Dependencies

To enable this port, set the **Output** parameter to `To output port`.

Data Types: `int8` | `int16` | `int32`

Parameters

Output — Output type
`To terminator` (default) | `To output port`

Specify if the block acts as a terminator, similar to the Terminator block, or produces data to an output port.

Number of channels — Number of data channels
2 (default) | positive integer

Specify the number of audio channels, *C*, sent to the audio device. This number must match the **Number of channels** parameter in the Audio Playback block.

Samples per frame — Size of data vector read from audio device
`4410` (default) | positive scalar integer

Specify the number of samples per frame, *M*, of audio data.

Sample time — Sample time in seconds
`-1` (default) | positive scalar

Enter the sample time to be used by the timer-driven task subsystem when you clear the **Enable event-based execution** parameter.

Version History

Introduced in R2021a

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”.

Embedded Coder does not generate code for this block. In the generated code, the advanced Linux sound architecture (ALSA) driver framework performs audio output on the device. The related Audio Playback block represents the ALSA driver. For more information on the ALSA driver framework, see the Advanced Linux Sound Architecture website.

See Also

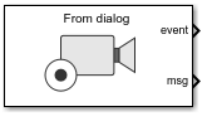
Audio Capture | Audio Capture Interface | Audio Playback

External Websites

Advanced Linux Sound Architecture

Video Capture Interface

Simulate capture of images from video stream on hardware



Libraries:
SoC Blockset / Peripherals

Description

The Video Capture Interface block simulates the behavior of a video input device. The block emits the captured image in message for a connected Video Capture block that can be used in an algorithm. The block can also emit an event that, when connected a Task Manager block, can trigger a task containing the Video Capture block.

Ports

Input

data — Image data
3*MN*-length vector

Specify a 3*MN*-length vector as the unwrapped images matrices.

Dependencies

To enable this port, set the **Input** parameter to `From input port`.

Data Types: `uint8`

Output

event — Task event signal
scalar

This port sends a message at each new image event. This output connects to the input of the Task Manager block to execute the associated event-driven task after receiving the new image.

Dependencies

To enable this port, select the **Show event port** parameter.

Data Types: `rteEvent`

msg — SoC message data
scalar

This port outputs data messages containing image data to a connected Video Capture block.

Data Types: `SoCData`

Parameters

Input — Source of simulated video data
`From dialog` (default) | `From input port` | `From timeseries object`

Specify the source of the simulated image data.

Value — Image to be imported
`uint8(floor(rand(1,3*160*120)*255))` (default) | 3*MN*-element vector

Specify the simulated image as a vector. The vector gets transformed into three *M*-by-*N* matrices representing the color channels of the image source.

Object name — Timeseries object name
[] (default) | MATLAB workspace variable

Specify the video source as a timeseries object in the MATLAB workspace.

Sample time — Sample time in seconds
`-1` (default) | positive scalar

Enter the sample time defining the rate at which to output the message containing video data.

Show event port — Option to enable task event ports
`off` (default) | `on`

Select this parameter to enable an event port that, when connected to the Task Manager block, can execute event-driven tasks.

Version History

Introduced in R2021a

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”.

Embedded Coder does not generate code for this block. In the generated code,the V4L2 driver framework performs the video capture on the device. The related Video Capture block represents the V4L2 driver. For more information on the V4L2 driver framework, see Video4Linux.

See Also

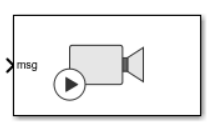
Video Display | Video Capture | Video Display Interface | Video Stream FIFO | Video Stream Connector | Video Test Sink | Video Test Source

External Websites

Video4Linux

Video Display Interface

Simulate display of images to video screen on device



Libraries:
SoC Blockset / Peripherals

Description

The Video Display Interface block simulates the behavior of a video output device. This block receives image samples as messages from a connected Video Display block that simulates the video driver code connected to your algorithm.

Ports

Input

msg — SoC message data
vector

This port receives the image data as a message from the **msg** port of a connected Video Display block.

Data Types: `SoCData`

Output

data — Image data
3**height*width*-element vector

The port outputs the unwrapped image matrix data as three *heightwidth*-element vectors where *height* and *width* are the dimensions of the image matrix defined by the `Image Size` parameter. The `3` is the multiplier for the RGB channels.

Dependencies

To enable this port, set the **Output** parameter to `To output port`.

Data Types: `uint8`

Parameters

Output — Output type
`To terminator` (default) | `To output port`

Specify if the block acts as a terminator, similar to the Terminator block, or produces data to an output port.

Image size — Image size
`160x120` (default) | `320x240` | `640x480` | `800x600` | `custom`

Specify the height and width dimensions of the image emitted as a vector by the `data` port of this block. Specify `custom` to set custom image dimensions.

Image size ([width, height]) — Image size
`[320, 240]` (default) | two-element vector of positive integers

Specify custom height and width dimensions of the image matrix emitted by the data port of this block.

Dependencies

To enable this parameter, set the **Image size** parameter to `custom`.

Pixel format — Format of the pixel data
`RGB` (default) | `YCbCr 4:2:2`

Specify the image data output encoding as `RGB` or `YCbCr 4:2:2` triplets.

Dependencies

To enable this parameter, set the **Image size** parameter to `custom`.

Sample time — Sample time in seconds
`-1` (default) | positive scalar

Enter the sample time defining the rate at which to receive the message containing video data.

Version History

Introduced in R2021a

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”.

Embedded Coder does not generate code for this block. In the generated code,the V4L2 driver framework performs the video display on the device. The related Video Display block represents the V4L2 driver. For more information on the V4L2 driver framework, see Video4Linux.

See Also

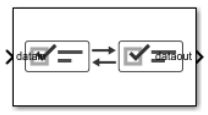
Video Display | Video Capture | Video Capture Interface | Video Stream FIFO | Video Stream Connector | Video Test Sink | Video Test Source

External Websites

Video4Linux

Interprocess Data Channel

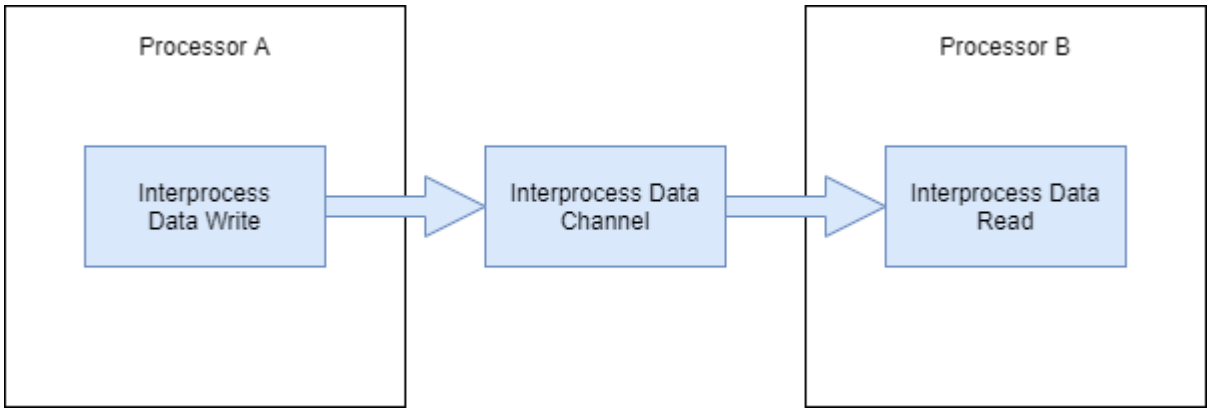
Model interprocessor data channel between two processors



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”
“Interprocess Data Communication via Dedicated Hardware Peripheral”

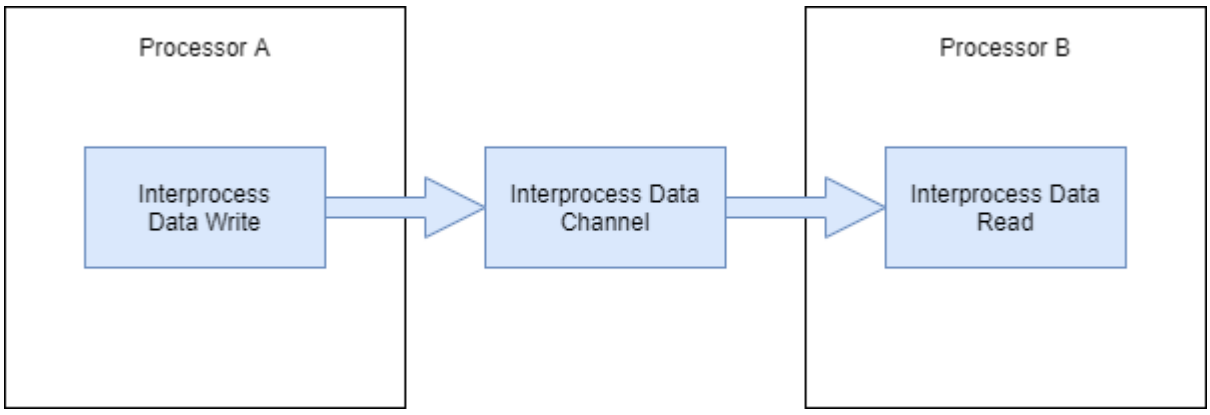
Interprocess Data Read

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.

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 and Interprocess Data Write 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

“Multiprocessor Execution”
“Interprocess Data Communication via Dedicated Hardware Peripheral”

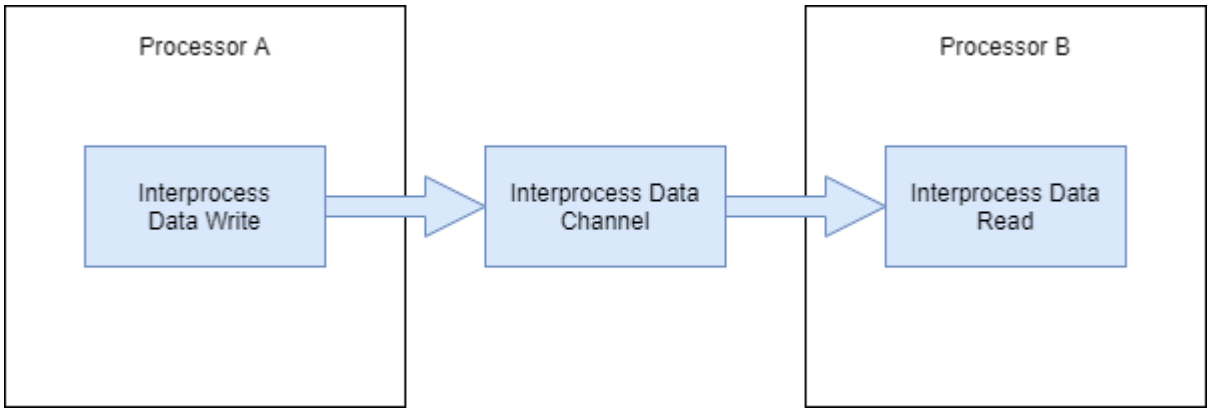
Interprocess Data Write

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.

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 and Interprocess Data Write blocks of the model.

Version History

Introduced in R2020b

See Also

Interprocess Data Read | Interprocess Data Channel

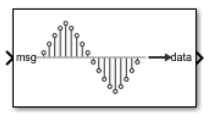
Topics

“Multiprocessor Execution”

“Interprocess Data Communication via Dedicated Hardware Peripheral”

ADC Read

Read ADC data values from ADC Interface block



Libraries:
SoC Blockset / Processor I/O

Description

The ADC Read block converts the message received from the ADC Interface block to a signal that can be used by an algorithm. The data type of the output signal is the same as the data type in the contained data massage.

Ports

Input

msg — Data message from register
scalar

This message port receives ADC value messages from a connected ADC Interface block. The messages process when the Task Manager block triggers a task containing the ADC Read block. For more information on messages, see “Messages”.

Data Types: `SoCData`

Output

data — Output signal
scalar

This port emits a measurement from the ADC Interface block.

Data Types: `double` | `single` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Parameters

Data type — Data type of ADC measurements received
`double` (default) | `single` | `uint8` | `int8` | `int16` | `int32` | `uint16` | `uint32`

Select the data type of the input data. Match this data type with data type of the ADC hardware.

Sample time — Sample time
`-1` (default) | nonnegative scalar

Specify how often the scheduler runs this block. If this value is `-1` (default), the scheduler assigns the sample time for the block.

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

When deployed to a supported hardware board, this generated code for this block reads the ADC registers in the MCU for the specified ADC module. For more information on configuring the register settings, see **Hardware Mapping** and Hardware Mapping Peripherals for Texas Instruments C2000 Processors Properties.

Note Supported hardware boards include the `TI Delfino F2837xD` and `TI Delfino F2837xD LaunchPad` from the SoC Blockset™ Support Package for Texas Instruments® C2000™ Processors.

See Also

ADC Interface | PWM Write | PWM Interface

Topics

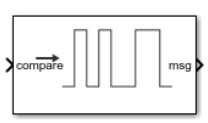
“Get Started with Multiprocessor Blocks on MCUs”
“Integrate MCU Scheduling and Peripherals in Motor Control Application”

External Websites

https://en.wikipedia.org/wiki/Analog-to-digital_converter

PWM Write

Send pulse width modulation (PWM) signal configuration to PWM Interface block



Libraries:
SoC Blockset / Processor I/O

Description

The PWM Write block sets the duty cycle for a PWM peripheral. In simulation, the block passes through the duty cycle input to drive the PWM Interface block that simulates the PWM switching signals produced by the hardware. When deployed to hardware, the PWM Write block writes to the appropriate PWM drivers on the hardware.

Ports

Input

Compare — Comparator trigger value
positive scalar | *N*-length positive vector

Specify the comparator values that trigger the change in pulse-width modulation waveform. When a *N* is 1, the comparator value defines the duty cycle of the PWM waveform. Using 2 comparators and the settings provided in the PWM Interface block, more fine grained control of the output waveform can be achieved.

Data Types: `single` | `double`

Period — Period of PWM waveform
positive scalar

Specify the total period of the PWM waveform.

Dependencies

This parameter is only available when the **Show period as input** parameter is enabled.

Data Types: `single` | `double`

Phase — Phase offset of the waveform
scalar from `0` to PWM period

Specify the phase of the PWM waveform relative period of waveform. The phase is represented as a scalar between `0` and PWM period.

Dependencies

This parameter is only available when the **Show phase as input** parameter is enabled.

Data Types: `single` | `double`

Output

msg — PWM duty cycle values
scalar

This message port sends duty cycle values as messages to a connected PWM Interface block. For more information on messages, see “Messages”.

Note This output is used only during simulation and is ignored in code generation and external mode simulation.

Data Types: `SoCData`

Parameters

Show period as input — Show period input port
`off` (default) | `on`

Show a **Period** input port on the block. This port accepts a dynamic period value for the PWM waveform, allowing the period of waveform to be modified during execution.

Show phase as input — Show phase input port
`off` (default) | `on`

Show a **Phase** input port on the block. This port accepts a dynamic phase value for the PWM waveform, allowing the phase offset of the waveform to be modified during execution.

Version History

Introduced in R2020b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

When deployed to a supported hardware board, this generated code for this block writes to the PWM registers in the MCU for the specified PWM module. For more information on configuring the register settings, see Hardware Mapping and Hardware Mapping Peripherals for Texas Instruments C2000 Processors Properties.

Note Supported hardware boards include the `TI Delfino F2837xD` and `TI Delfino F2837xD LaunchPad` from the SoC Blockset Support Package for Texas Instruments C2000 Processors.

See Also

PWM Interface

Topics

“Get Started with Multiprocessor Blocks on MCUs”

“Integrate MCU Scheduling and Peripherals in Motor Control Application”

External Websites
https://en.wikipedia.org/wiki/Pulse-width_modulation

Register Read

Read data from a register region on the specified IP core

Libraries:
SoC Blockset / Processor I/O

Description

The Register Read block reads data from a register region on the specified IP core. In simulation, a timer-driven or event-driven task subsystem contains the Register Read block. The data signals from the Register Read block connect to the Register Channel block managing those registers and their transactions.

When developing or analyzing the software side of an SoC application, the Register Read block can be connected to an IO Data Source block. In this configuration, the IO Data Source block provides either previously recorded or artificial data, enabling a more directed simulation of the software and processor side of the application, without need to explicitly model the hardware and memory interactions.

Ports

Input

msg — Data message from register
scalar

This message port receives data messages from a connected Register Channel or IO Data Source block. The messages process when the Task Manager block triggers task containing the Register Read block. For more information on messages, see “Messages”.

Data Types: `SoCData`

Output

data — Output signal
vector

This port emits the data vector read from the specified registers in the Register Channel starting at **Offset address** from the base address of the IP core.

Data Types: `single` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Parameters

Device name — Path and file name of IP core device
`/dev/mwipcore` (default) | character array

Enter the path and file name of the IP core device.

Offset address — Offset from the base address of the IP core to the register
`hex2dec('0100')` (default) | positive integer

Enter the offset from the base address of the IP core to the register. The block reads data from this register. Use the `hex2dec` function when you specify the offset address using a hexadecimal number expressed as a character vector. The offset address can be selected using the Memory Mapper tool.

Output data type — Data type used by IP core
uint32 (default) | single | int8 | uint8 | int16 | int32 | uint32 | boolean | fixed-point

Enter the data type used by the IP core.

Output vector size — Size of data vector from IP core
`1` (default) | positive integer

Enter the size of the data vector read from the IP core device.

Sample time — Sample time
`0.1` (default) | positive number

Enter the sample time in seconds. Either the connected Register Channel or IO Data Source blocks get polled at this rate when this block is used in a timer-driven task.

Version History

Introduced in R2019a

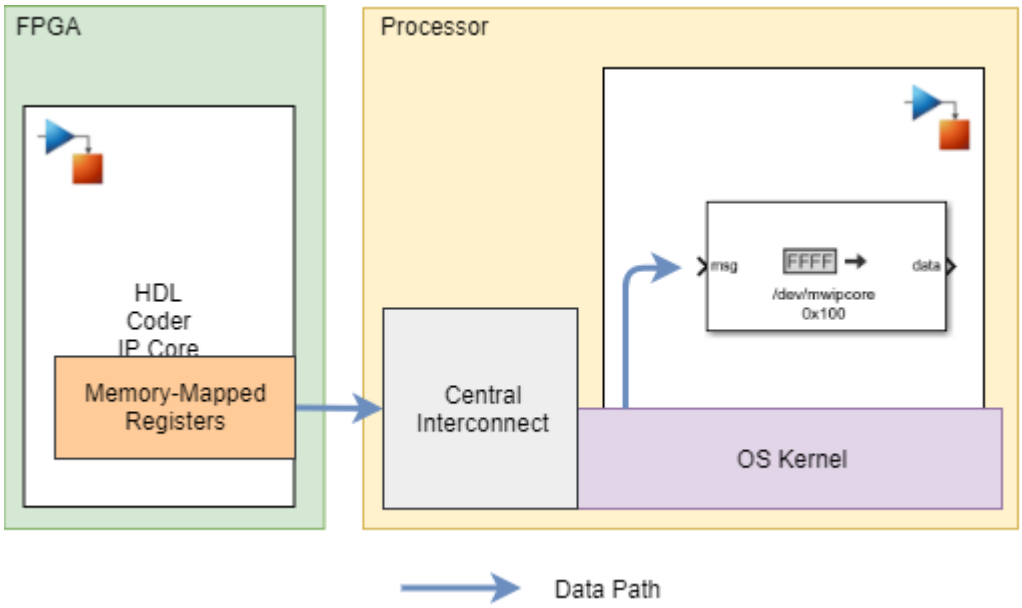
Extended Capabilities

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 Builder implements the Register Read block with FPGA and processor IPs that use the AXI4 interface protocol. The AXI4 interface protocol allows the processor algorithm to read vector data from a contiguous group of registers on the FPGA. Use this block for simple, low-throughput memory-mapped communication, such as reading from control and status registers. This diagram shows a generalized representation of the generated code implementation.



See Also

Register Channel

Register Write

Write data to a register region on the specified IP core

Libraries:
SoC Blockset / Processor I/O

Description

The Register Write block writes data from your processor algorithm to a register region on the specified IP core. In simulation, a timer-driven or event-driven task subsystem contains the Register Write block. The data signals from the Register Write block connect to the Register Channel block managing those registers and their transactions.

When developing or analyzing the software side of an SoC application, the Register Write block can be connected to an IO Data Sink block. In this configuration, the IO Data Sink block provides either previously recorded or artificial data, enabling a more directed simulation of the software and processor side of the application, without need to explicitly model the hardware and memory interactions.

Ports

Input

data — Data input
vector

This port receives the data vector to write to the registers on the IP core starting at **Offset address** from the base address of the IP core.

Data Types: `single` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean` | `fixed point`

Output

msg — Output register data message
scalar

This message port sends the output register data, as a message, to the connected Register Channel or IO Data Sink block. For more information on messages, see “Messages”.

Data Types: `SoCData`

Parameters

Output sink — Direct output to port or base workspace
`To output port` (default) | `Base workspace` | `IP core register`

Select `To output port` to write data to the output port, `msg`. Select `Base workspace` to write data to a variable in the base workspace. When writing to the base workspace, the block updates the value

of a `Simulink.Parameter` object with name set by `Simulink.Parameter object name` parameter in the base workspace. Select `IP core register` to write to an IP Core Register Read block with the same `Register name` parameter.

Note Placing the Register Write block inside a Initialize Function block subsystem, writes to a `Simulink.Parameter` object at the start of simulation. A register, represented as a Constant block, in an FPGA reference model can be initialized at the start of simulation with the value of the `Simulink.Parameter` object. This method of writing to FPGA registers requires a constant value throughout the simulation but can reduce the simulation time required by your SoC model.

Simulink.Parameter object name — Name of Simulink.Parameter object
`A` (default) | character vector

Name of `Simulink.Parameter` object to be created in the `Base workspace`.

Example: `A`

Dependencies

To enable this parameter, set `Output sink` to `Base workspace`.

Register name — Name of a register in IP Core
`RegA` (default) | character vector

Name of register defined in an IP Core Register Read block located in the FPGA reference model.

Example: `RegA`

Dependencies

To enable this parameter, set `Output sink` to `IP core register`.

Device name — Path and file name of IP core device
`/dev/mwipcore` (default) | character array

Enter the path and file name of the IP core device.

Offset address — Offset from the base address of the IP core to the register
`hex2dec('0100')` (default) | positive integer

Enter the offset from the base address of the IP core to the register. The block writes data to this register. Use the `hex2dec` function when you specify the offset address using a hexadecimal number expressed as a character vector. The offset address can be selected using the Memory Mapper tool.

Version History

Introduced in R2019a

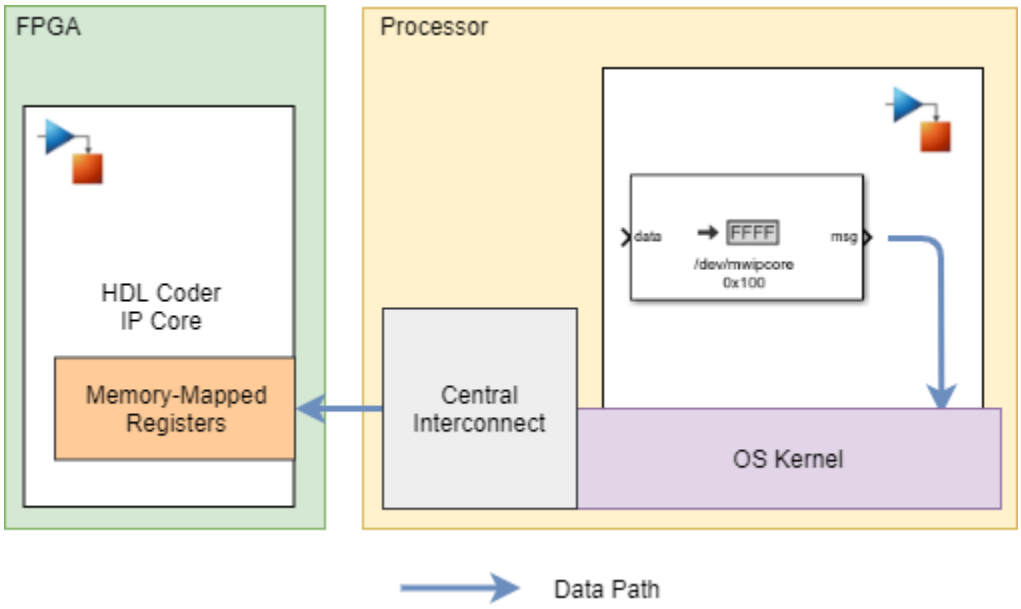
Extended Capabilities

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 Builder implements the Register Write block with FPGA and processor IPs that use the AXI4 interface protocol. The AXI4 interface protocol allows the processor to write vector data from the processor to a contiguous group of registers on the FPGA. Use this block for simple, low-throughput memory-mapped communication, such as writing to control and status registers. This diagram shows a generalized representation of the generated code implementation.



See Also

Register Channel | Register Read

Stream Read

Stream data from shared memory to processor algorithms

Libraries:
SoC Blockset / Processor I/O

Description

The Stream Read block streams data from shared memory in the memory channel to your processor algorithm. In simulation, a timer-driven or event-driven task subsystem contains the Stream Read block. The data signals from the Memory Channel block connect to the Stream Read block. Following a write to the shared memory, the Memory Channel notifies the Task Manager block of the write event. The Task Manager block then triggers the event-driven subsystem containing the Stream Read block and associated algorithm.

When developing or analyzing the software side of an SoC application, the Stream Read block can be connected to an IO Data Source block. In this configuration, the IO Data Source block provides either previously recorded or artificial data, enabling a more directed simulation of the software and processor side of the application, without need to explicitly model the hardware and memory interactions.

Ports

Output

data — Data frame from shared memory
vector

This port emits a data frame read from shared a region of shared memory defined in the Memory Channel block.

Data Types: `uint16` | `uint32` | `uint64` | `fixdt(0,128,0)`

valid — Valid frame data
scalar

A flag indicating a valid data frame read from the memory channel.

Data Types: `Boolean`

done — Notification message of completed read
scalar

This message port sends notification, as a message, to the connected Memory Channel or IO Data Source block that the read completed. For more information on messages, see “Messages”.

Data Types: `Boolean`

Input

msg — Data message from memory
scalar

This message port receives data messages from the connected Memory Channel or IO Data Source block. The messages process when the Task Manager block triggers the task containing the Stream Read block. For more information on messages, see “Messages”.

Data Types: `SoCData`

Parameters

Main

Device name — Name of IP core device
`ip:s2mm0` (default) | colon-separated list of IP core name and channel

Enter the name and channel of the IP core on the FPGA as a colon separated list.

Output data type — Data type of IP core
`uint32` (default) | `uint16` | `uint64` | `fixdt(0,128,0)`

Enter the data type used by the memory channel.

Samples per frame — Size of data vector read from IP core
`1024` (default) | positive scalar integer

Enter the size of the data vector read from the memory channel.

Number of buffers — Number of data buffers
`16` (default) | positive integer

Enter the number of data frame buffers in physical memory. This number should match the **Number of buffers** parameter in the Memory Channel block or IO Data Source block.

Enable event-based execution — Enable event-driven task execution
`on` (default) | `off`

To use this block in event-driven task subsystems, select this parameter. To use this block in timer-driven task subsystems, clear this parameter.

When **Enable event-based execution** is selected, this block reads from the Memory Channel each time a a full buffer is available in the shared memory region. When **Enable event-based execution** is cleared, the block reads the data in the shared memory region at each sample time.

Sample time — Sample time in seconds
`-1` (default) | positive scalar

Enter the sample time used by the timer-driven task subsystem when the **Enable event-based execution** is cleared.

Version History

Introduced in R2019a

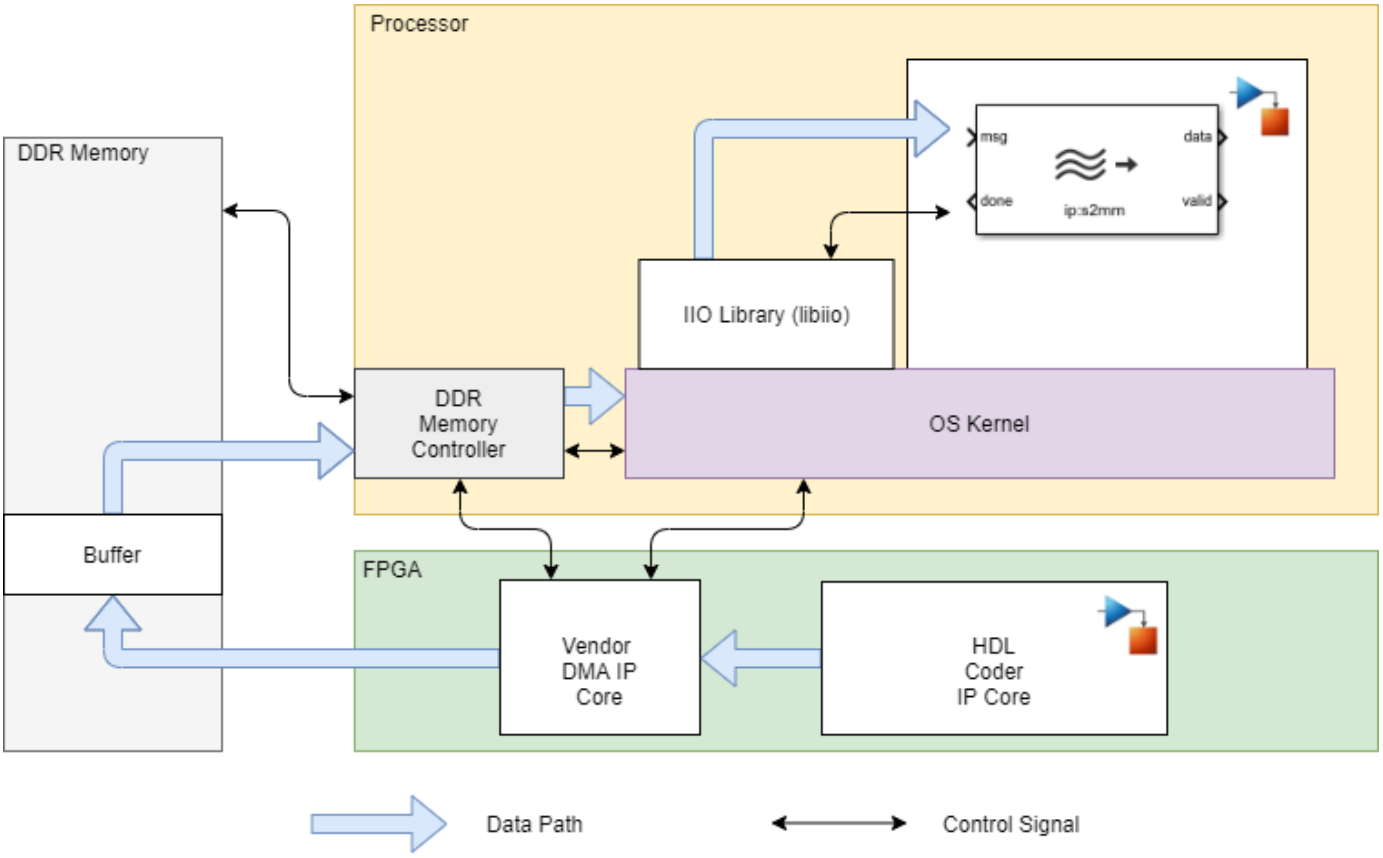
Extended Capabilities

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”.

The **SoC Builder** tool implements the Stream Read, Stream Write, Memory Channel, and Task Manager blocks with FPGA and processor IPs that use the AXI4-stream communication protocol. The AXI4-stream protocol uses a direct memory access (DMA) to read a data vector to a shared region on the external memory. This protocol enables high-speed streaming of data between the FPGA and the embedded processor through external memory. This diagram shows a generalized representation of the generated code implementation.



See Also

Task Manager | Memory Channel | Stream Write

Topics

“Event-Driven Tasks”

Stream Write

Stream data from processor algorithms to shared memory

Libraries:
SoC Blockset / Processor I/O

Description

The Stream Write block streams data from your processor algorithm to shared memory in the Memory Channel block. The Stream Write block has an internal counter that keeps track of the number of empty buffers in the shared memory. After a successful read from memory, the memory sends a done signal to the Stream Write block. Then, the block increments the counter, asserting that a buffer is available in the memory. A write transaction is successful if at least one buffer is available for writing. The Stream Write block sends a status of `True` back to the software. You can use this status signal to perform actions such as counting dropped frames or issuing rewrite requests.

In simulation, a timer-driven or event-driven task subsystem contains the Stream Write block. The data signals from the software algorithm connect to the Stream Write block. The write transaction is issued as a message to the Memory channel block. After a read operation from shared memory, the Memory Channel block notifies the Stream Write block of the read event via the `done` signal.

Ports

Input

data — Data frame from software algorithm
vector

This port receives a data frame from the software algorithm. The block then streams the data as a message to a region of shared memory defined in the Memory Channel block.

Data Types: `uint16` | `uint32` | `uint64` | `fixdt(0,128,0)`

done — Notification of available buffer in memory
`scalar`

This message port receives a notification from the connected Memory Channel or IO Data Sink block. The notification indicates that a read transaction completed and that a buffer in memory is available for writing.

Data Types: `Boolean`

Output

msg — Data message to memory
scalar

When buffer space is available in the memory, this message port emits data messages to the connected Memory Channel or IO Data Sink block. For more information on messages, see “Messages”.

Data Types: `SoCData`

status — Status of completed write transaction
`scalar`

This port sends a true status(1) to the processor after a write transaction to memory occurred. Use this status to count dropped frames.

Data Types: `Boolean`

Parameters

Device name — Name of IP core device
`ip:MM2S` (default) | colon-separated list of IP core name and channel

The device name parameter is generated by the **SoC Builder** tool. Enter the name and channel of the IP core on the FPGA as a colon-separated list.

Number of buffers — Number of data buffers
3 (default) | positive integer

Enter the number of data frame buffers in the physical memory. This number must match the **Number of buffers** parameter in the Memory Channel or IO Data Sink block.

Enable event-based execution — Option to enable event-driven task execution
`on` (default) | `off`

- Select this parameter to use this block in event-driven task subsystems. In this case, the block writes to the Memory Channel block each time an empty buffer is available in the shared memory region.
- Clear this parameter to use this block in timer-driven task subsystems. In this case, the block writes the data in the shared memory region at each sample time.

Version History

Introduced in R2020b

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”.

The **SoC Builder** tool implements the Stream Write, Stream Read, Memory Channel, and Task Manager blocks with FPGA and processor IPs that use the AXI4-stream communication protocol. The AXI4-stream protocol uses a direct memory access (DMA) to read a data vector to a shared region on

the external memory. This protocol enables high-speed streaming of data between the FPGA and the embedded processor through external memory.

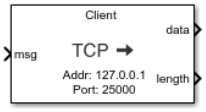
See Also

Blocks
Task Manager | Memory Channel | Stream Read

Topics
“Event-Driven Tasks”

TCP Read

Receive TCP/IP packets from remote host over TCP/IP network



Libraries:
SoC Blockset / Processor I/O

Description

The TCP Read block receives a stream of TCP/IP packets from a remote host over a TCP/IP (Transmission Control Protocol/Internet Protocol) network.

Ports

Input

msg — Stream of TCP/IP packets received from the remote host
scalar

This message port receives TCP/IP packets, as messages, from a connected IO Data Source block. The messages process when the Task Manager block triggers task containing the TCP Read block. For more information on messages, see “Messages”.

Note This input is used only during simulation. and does nothing in code generation and external mode simulation.

Data Types: `SoCData`

Output

data — TCP/IP packet received from remote host
numeric vector

Output TCP/IP packets received from remote host, returned as a numeric vector. The size and data type of this output is same as the size and data type of the input message.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

length — Length of output TCP/IP packet
nonnegative scalar

Length of output TCP/IP packets returned on the output **data** port.

Data Types: `uint32`

Parameters

Network role — Set block as client or server
`Client` (default) | `Server`

To configure this block as a TCP/IP client or server, set this parameter to `Client` or `Server`, respectively.

When you set this parameter to `Client`, you must provide the remote IP address and remote IP port number of the TCP/IP server from which you want to receive TCP/IP packets. Specify this information by using the **Remote address** and **Remote port** parameters.

When you set this parameter to `Server`, you must provide the local IP port number, which acts as the listening port of the TCP/IP server running in the hardware. Specify this information using the **Local port** parameter. When you set this parameter to `Server`, you can only connect to one client at a time.

Remote address — IP address of remote server from which TCP/IP packets are received
`127.0.0.1` (default) | dotted-quad expression

Specify the IP address of remote server from which you want to receive TCP/IP packets.

Dependencies

To enable this parameter, set the **Network role** parameter to `Client`.

Remote port — IP port on remote server from which TCP/IP packets are received
`25000` (default) | integer from 1 to 65535

Specify the port number of the remote server from which you want to receive TCP/IP packets.

Dependencies

To enable this parameter, set the **Network role** parameter to `Client`.

Local port — IP port of host on which data is received
`-1` (default) | integer from 1 to 65,535

Specify the port number of the application on which you want to receive the TCP/IP packets when the **Network role** is set to `Client`. The default value `-1` assigns any random available port as local port when you set the **Network role** parameter to `Client`.

This local port acts as the listening port on the TCP/IP server when the **Network role** is set to `Server`. Specify a value from 1 to 65535 when you set **Network role** parameter to `Server`. Specify this local port number as the remote port number in the sending host from which you want to receive TCP/IP packets.

Data type — Data type of TCP/IP packets received
`uint8` (default) | `single` | `double` | `int8` | `int16` | `int32` | `uint16` | `uint32`

Select the data type of the input data. Match this data type with data type of TCP/IP packets sent from the remote host.

Maximum data length (elements) — Maximum length of output TCP/IP packet
`1` (default) | positive scalar

Specify the maximum number of data elements that the output **data** port can produce at every time step.

Enable event-based execution — Enable event-based task execution
off (default) | on

To generate event-driven code, select this parameter. To generate timer-driven code, clear this parameter.

When **Enable event-based execution** is selected, the block reads TCP/IP packets from the socket buffer whenever any TCP/IP packet is received in the socket buffer irrespective of the sample time. When **Enable event-based execution** is cleared, the block reads available TCP/IP packets from the socket buffer at each sample time. To set the size of the TCP/IP packet that the block can read from the socket buffer, specify the size in the **Receive buffer size** parameter.

Sample time — Sample time
`-1` (default) | nonnegative scalar

Specify how often the scheduler runs this block. If this value is `-1` (default), the scheduler assigns the sample time for the block.

Version History

Introduced in R2019a

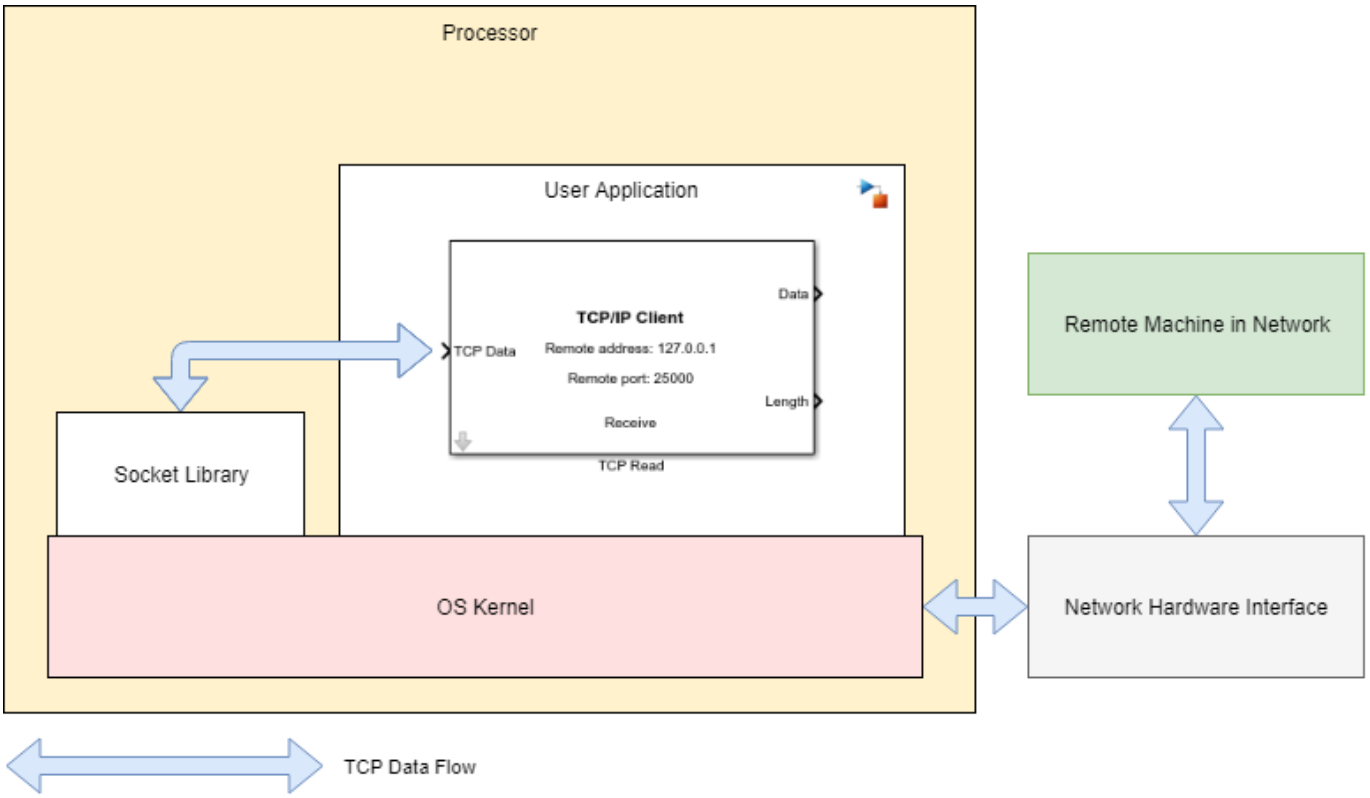
Extended Capabilities

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”.

Embedded Coder generates event-driven or timer-driven code for this block based on the **Enable event-based execution** parameter selection. This diagram shows a generalized representation of the generated code implementation.



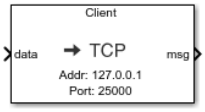
Note Timing measurements from generated code might vary within the execution of a task instance compared to the timing of tasks in simulation. You can configure your model to use data caching in task signals to reach improved agreement between the simulation and generated code. For more information, see Value and Caching of Task Subsystem Signals.

See Also

TCP Write | IO Data Source | Task Manager

TCP Write

Send TCP/IP packets to remote host over TCP/IP network



Libraries:
SoC Blockset / Processor I/O

Description

The TCP Write block sends TCP/IP packets to a remote host over a TCP/IP (Transmission Control Protocol/Internet Protocol) network.

Ports

Input

data — Input data
numeric vector

Input data, specified as a numeric vector. The block sends this data over a TCP/IP network to the remote host.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output

msg — Stream of TCP/IP packets sent to the remote host
scalar

This message port sends TCP/IP packets, as messages, to a connected IO Data Sink block. For more information on messages, see “Messages”.

Note This output is used only during simulation. and does nothing in code generation and external mode simulation.

Data Types: `SoCData`

Parameters

Network role — Set the block as server or client
`Client` (default) | `Server`

To configure this block as a TCP/IP client or server, set this parameter to `Client` or `Server`, respectively.

When you set this parameter to `Client`, you must provide the remote IP address and remote IP port number of the TCP/IP server to which you want to send TCP/IP packets. Specify this information by using the **Remote address** and **Remote port** parameters.

When you set this parameter to `Server`, you must provide the local IP port number, which acts as the listening port of the TCP/IP server running in the hardware. Specify this information using the **Local port** parameter.

Remote address — IP address of remote server to which TCP/IP packets are sent
`127.0.0.1` (default) | dotted-quad expression

Specify the IP address of the remote server to which you want to send TCP/IP packets.

Dependencies

To enable this parameter, set the **Network role** parameter to `Client`.

Remote port — IP port of remote server to which TCP/IP packets are sent
`25000` (default) | integer from 1 to 65,535

Specify the port number of the remote server to which you want to send TCP/IP packets.

Dependencies

To enable this parameter, set the **Network role** parameter to `Client`.

Local port — IP port on sending host from which TCP/IP packets are sent
`-1` (default) | integer from 1 to 65535

When the **Network role** parameter is set to `Client`, specify the IP port number of the application from which you want to send TCP/IP packets. The default value `-1`, sets this IP port number to a random available port number and uses that port to send the packets.

When the **Network role** parameter is set to `Server`, this local port acts as the listing port of the TCP/IP server running in the hardware. In this case, specify a value from 1 to 65,535 for this parameter.

Byte order — Byte order
`LittleEndian` (default) | `BigEndian`

Byte order of the TCP/IP packets, specified as one of these values:

- `LittleEndian` — Sets the byte order of TCP/IP packets to little endian.
- `BigEndian` — Sets the byte order of TCP/IP packets to big endian.

Version History

Introduced in R2019a

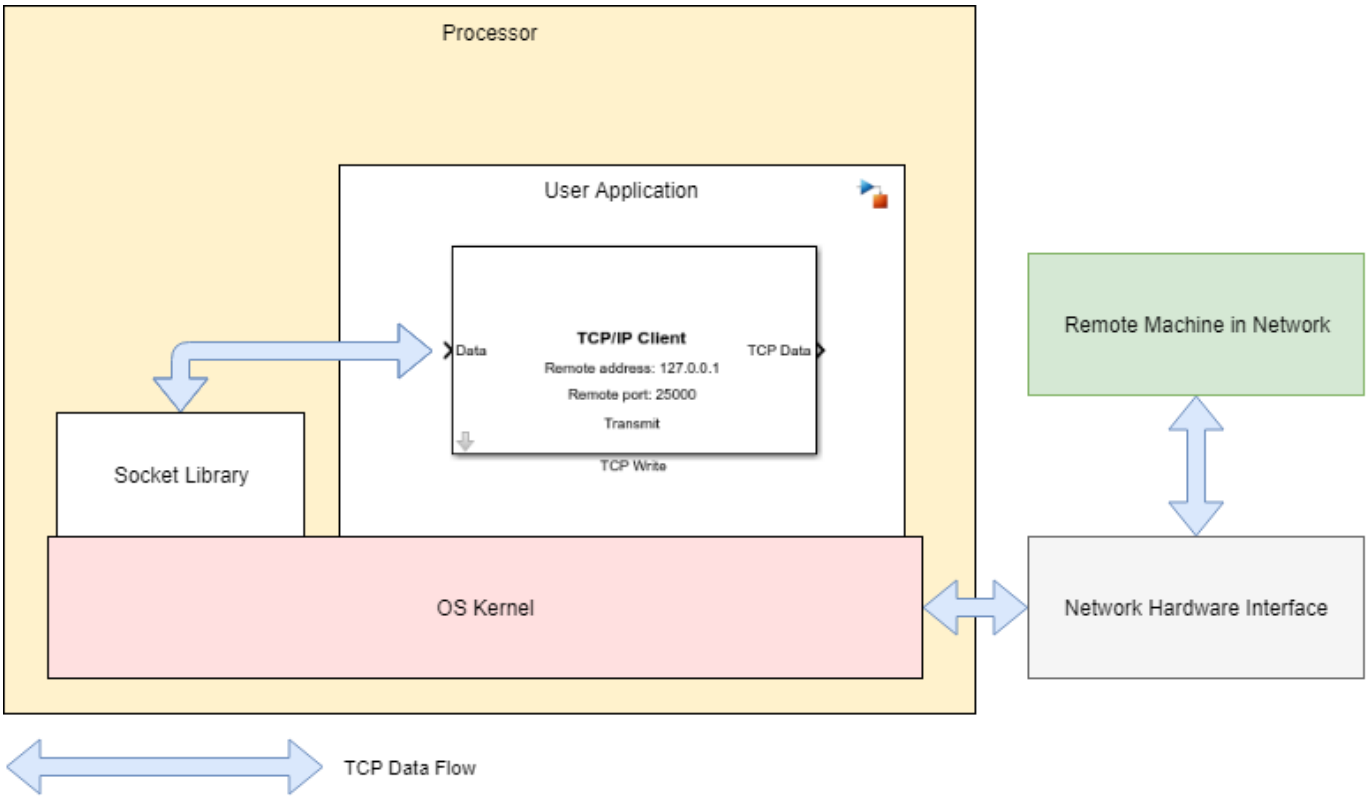
Extended Capabilities

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”.

Embedded Coder generates event driven code for this block. This diagram shows a generalized representation of the generated code implementation.



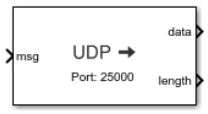
Note Timing measurements from generated code might vary within the execution of a task instance compared to the timing of tasks in simulation. You can configure your model to use data caching in task signals to reach improved agreement between the simulation and generated code. For more information, see Value and Caching of Task Subsystem Signals.

See Also

TCP Read | IO Data Sink | Task Manager

UDP Read

Receive UDP packets from remote host



Libraries:
SoC Blockset / Processor I/O
SoC Blockset / Host I/O

Description

The UDP Read block receives UDP (User Datagram Protocol) packets from a remote host on the application on target. The remote host is the computer or hardware from which you want to receive UDP packets. The block reads UDP packets from UDP socket buffer and returns the UDP packets as a one-dimensional array.

Ports

Input

msg — UDP packet
numeric vector

This message port receives UDP packets, as messages, from a connected IO Data Source block. The messages process when the Task Manager block triggers task containing the UDP Read block. For more information on messages, see “Messages”.

Note This input is used only during simulation. and does nothing in code generation and external mode simulation.

Data Types: `SoCData`

Output

data — Output UDP packet
numeric vector

Output UDP packet, received from a remote host, returned as a numeric vector.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

length — Length of received UDP packet
nonnegative scalar

Length of output UDP packet returned on the output **data** port.

Data Types: `uint32`

Parameters

Local port — IP port number of local host
`25000` (default) | integer from 1 to 65,535

Specify the port number of the application on target in which you want to receive data. Match the local IP port number with the remote IP port number of the remote host.

Data type — Data type of received data
uint8 (default) | single | double | int8 | int16 | int32 | uint16 | uint32

Select the type of data the block receives from the sending host. Match the data type with data type of input data.

Maximum data length (elements) — Maximum length of output UDP packet
1 (default) | positive integer

Specify the maximum number of data elements that the output **data** port can produce at every time step.

Receive buffer size (bytes) — Number of data bytes in received data
`65535` (default) | integer from 1 to 65,535

Specify the maximum number of data bytes that the block can receive at each time step.

Enable event-based execution — Enable or disable event-based task execution
`off` (default) | `on`

To generate event-driven code, select this parameter. To generate timer-driven code, clear this parameter.

When **Enable event-based execution** is selected, the block reads data from the socket buffer whenever any UDP data is received in the socket buffer irrespective of the sample time. When **Enable event-based execution** is cleared, the block reads available UDP data from the socket buffer at each sample time. To set the size of the data that the block can read from the socket buffer, specify the size in the **Receive buffer size** parameter.

Sample time — Sample time
`-1` (default) | nonnegative scalar

Specify how often the scheduler runs this block. If this value is `-1` (default), the scheduler assigns the sample time for the block.

Version History

Introduced in R2019a

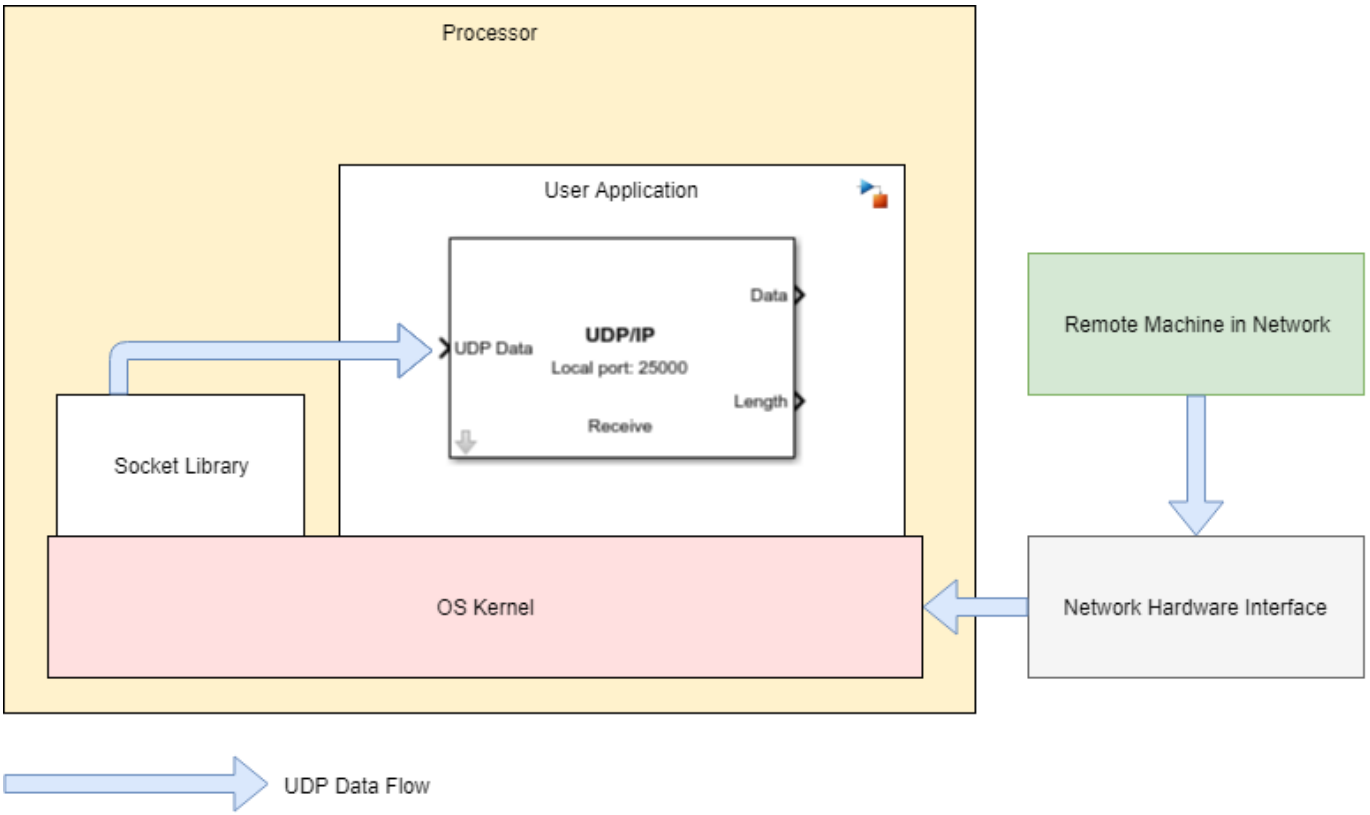
Extended Capabilities

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”.

Embedded Coder generates event-driven or timer-driven code for this block, based on the **Enable event-based execution** parameter selection. This diagram shows a generalized representation of the generated code implementation.



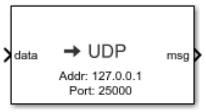
Note Timing measurements from generated code might vary within the execution of a task instance compared to the timing of tasks in simulation. You can configure your model to use data caching in task signals to reach improved agreement between the simulation and generated code. For more information, see Value and Caching of Task Subsystem Signals.

See Also

IO Data Source | UDP Write | Task Manager

UDP Write

Send UDP packets to remote host



Libraries:
SoC Blockset / Processor I/O
SoC Blockset / Host I/O

Description

The UDP Write block sends UDP (User Datagram Protocol) packets from the application on target to a remote host. The remote host is the computer or hardware to which you want to send UDP packets.

Ports

Input

data — Input signal
numeric vector

Input data, specified as a numeric vector. The block sends this data as UDP packet to the remote host. To set the byte order in which you want to send this UDP data, set the **Byte order** parameter. The block converts this input data to the specified byte order type.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32`

Output

msg — UDP packet sent to remote host
numeric vector

This message port sends UDP packets, as messages, to a connected IO Data Sink block. For more information on messages, see “Messages”.

Note This output is used only during simulation. and does nothing in code generation and external mode simulation.

Data Types: `SoCData`

Parameters

Remote IP address (255.255.255.255 for broadcast) — IP address of remote host to which data is sent
`127.0.0.1` (default) | dotted-quad expression

Specify the remote IP address of the host to which you want to send UDP packets.

Remote port — IP port of remote host to which data is sent
`25000` (default) | integer from 1 to 65,535

Specify the port number of the host to which you want to send UDP packets.

Local port — IP port number of application on target from which data is sent
`-1` (default) | integer from 1 to 65,535

Specify the port number of the application on the target from which you want to send the UDP packets. The default value `-1`, sets the local port number to a random available port number and uses that port to send the UDP packets.

Byte order — Byte order
`LittleEndian` (default) | `BigEndian`

Byte order of the UDP packets, specified as one of these values:

- `LittleEndian` — Sets the byte order of UDP packets to little endian.
- `BigEndian` — Sets the byte order of UDP packets to big endian.

Version History

Introduced in R2019a

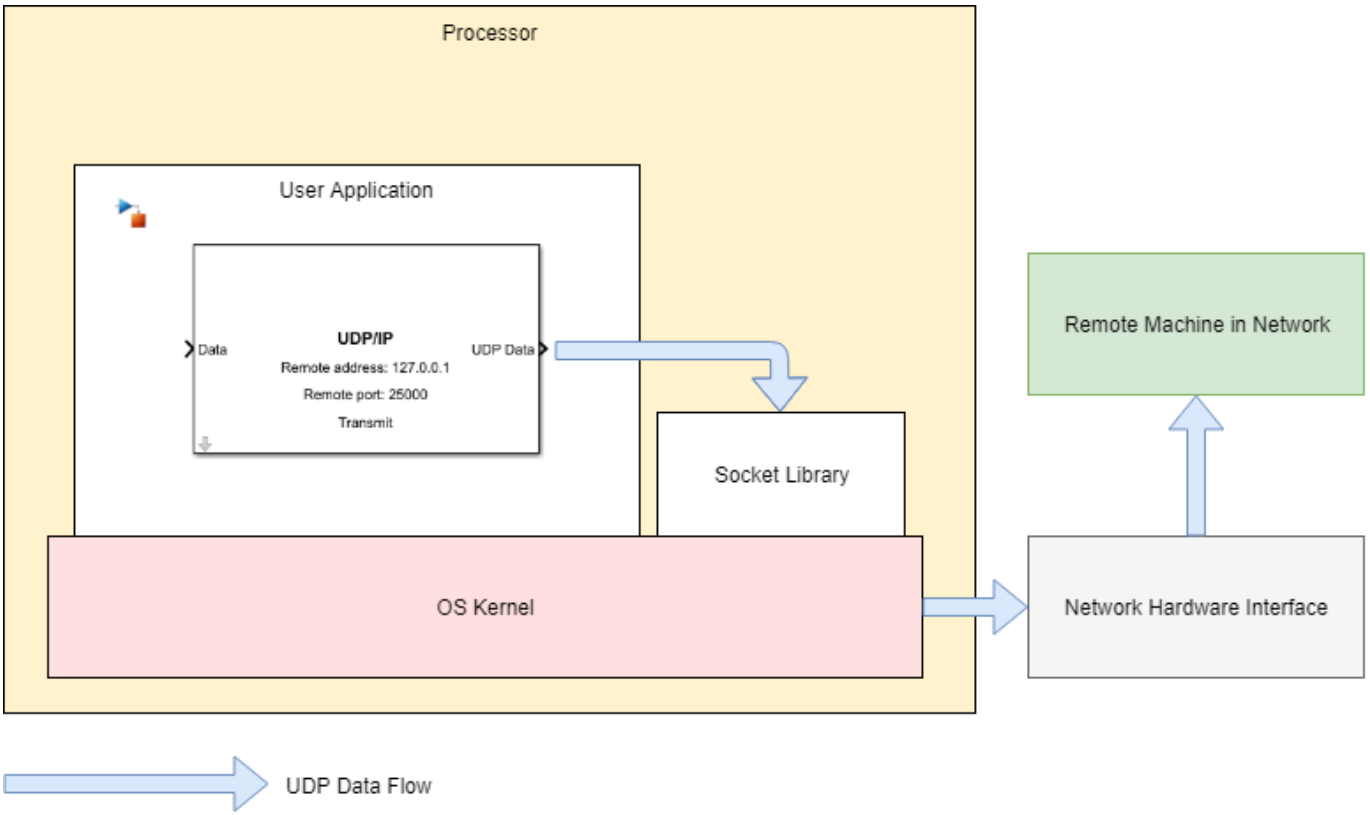
Extended Capabilities

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”.

Embedded Coder generates event-driven code for this block. This diagram shows a generalized representation of the generated code implementation.



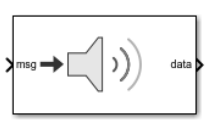
Note Timing measurements from generated code might vary within the execution of a task instance compared to the timing of tasks in simulation. You can configure your model to use data caching in task signals to reach improved agreement between the simulation and generated code. For more information, see Value and Caching of Task Subsystem Signals.

See Also

IO Data Sink | UDP Read | Task Manager

Audio Capture

Capture audio samples from an input audio device and send to an algorithm



Libraries:
SoC Blockset / Peripherals

Description

The Audio Capture block simulates behavior of the driver code that captures samples of an audio stream on an hardware board. The input of this block must be connected to an Audio Capture Interface block that simulates the physical audio device on a hardware board. Place this block inside a task and connect to an algorithm to process a frame of audio samples each time the task executes. For more information on tasks, see “Timer-Driven Task” and “Event-Driven Tasks”.

Ports

Input

msg — Data message from audio capture interface
scalar

This message port receives data messages from a connected Audio Capture Interface block. The messages process when the Task Manager block triggers the task containing the Audio Capture block. For more information on messages, see “Messages”.

Data Types: `SoCData`

Output

data — Data frame from captured audio
M-element vector | *M*-by-*C* matrix

When block receives a single audio channel, data is an *M*-element audio data frame received from a simulated hardware audio source. The **Samples per frame** parameter defines the number of samples, *M*, of audio data. When the block receives multiple audio channels, the audio data is an *M*-by-*C* matrix, where *C* is specified by the **Number of channels** parameter.

Data Types: `int8` | `int16` | `int32`

Parameters

Data type — Data type of audio device
`int16` (default) | `int8` | `int32`

Specify the data type for the audio capture device.

Number of channels — Number of data channels
2 (default) | positive integer

Specify the number of audio channels, *C*, received from the audio device. This number should match the **Number of channels** parameter in the Audio Capture Interface block.

Samples per frame — Size of data vector read from audio device
`4410` (default) | positive scalar integer

Specify the number samples per frame, *M*, of audio data received.

Sample time — Sample time in seconds
`-1` (default) | positive scalar

If used in a timer-driven task, enter the sample time of the task defined in the Task Manager block. If used in the event-driven task, enter `-1`.

Version History

Introduced in R2021a

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. For more information on generating code for SoC designs, see “Generate SoC Design”.
- To generate and execute C code for your SoC models, this block requires Embedded Coder and SoC Blockset Support Package for Embedded Linux Devices.
- To use this block, you must set **Hardware Board** to `Embedded Linux Board`. You can then follow the instructions to configure the board. For more information, see “Create an Embedded Linux Device Configuration” (SoC Blockset Support Package for Embedded Linux Devices).

When deployed to a supported hardware board, this block uses the V4L2 driver framework to capture images from video capture devices, such as a camera. To specify the video capture device and event triggers, use the Hardware Mapping tool. For more information on the V4L2 driver framework, see Video4Linux.

See Also

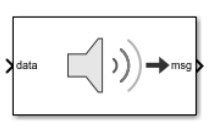
Hardware Mapping | Audio Capture Interface | Audio Playback Interface | Audio Playback

External Websites

Advanced Linux Sound Architecture

Audio Playback

Playback audio samples from an algorithm to an output audio device



Libraries:
SoC Blockset / Peripherals

Description

The Audio Playback block simulates behavior of the driver code that plays audio to an audio output, such as a speaker, on a hardware board. The output of this block must be connected to an Audio Playback Interface block that simulates a physical audio device on the hardware board. Place this block inside a task and connect to an algorithm to output a frame of audio samples each time the task executes. For more information on tasks, see “Timer-Driven Task” and “Event-Driven Tasks”.

Ports

Input

data — Audio data frame
M-element vector | *M*-by-*C* matrix

Audio data frame to be sent to an audio playback device, specified as an *M*-element for single channel audio or as an *M*-by-*C* matrix for multi-channel audio. *C* is specified by the **Number of channels** parameter.

Data Types: `int8` | `int16` | `int32`

Output

msg — Data message from audio capture interface
scalar

This port outputs data messages containing audio data to a connected Audio Playback Interface block. For more information on messages, see “Messages”.

Data Types: `SoCData`

Parameters

Number of channels — Number of channels
2 (default) | positive scalar

Specify the number of audio channels, *C*, in each audio data sample.

Version History

Introduced in R2021a

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. For more information on generating code for SoC designs, see “Generate SoC Design”.
- To generate and execute C code for your SoC models, this block requires Embedded Coder and SoC Blockset Support Package for Embedded Linux Devices.
- To use this block, you must set **Hardware Board** to `Embedded Linux Board`. You can then follow the instructions to configure the board. For more information, see “Create an Embedded Linux Device Configuration” (SoC Blockset Support Package for Embedded Linux Devices).

When deployed to a supported hardware board, this block uses the V4L2 driver framework to capture images from video capture devices, such as a camera. To specify the video capture device and event triggers, use the Hardware Mapping tool. For more information on the V4L2 driver framework, see Video4Linux.

See Also

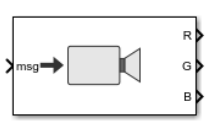
Hardware Mapping | Audio Capture Interface | Audio Playback Interface | Audio Capture

External Websites

Advanced Linux Sound Architecture

Video Capture

Capture video from input video device and send to algorithm



Libraries:
SoC Blockset / Peripherals

Description

The Video Capture block simulates behavior of the driver code that captures images from a video device, such as a camera, on your hardware board. The input of this block must be connected to an Video Capture Interface block that simulates a physical video source device on the hardware board. Place this block inside a task and connect it to an algorithm to process an image each time the task executes. For more information on tasks, see “Timer-Driven Task” and “Event-Driven Tasks”.

Ports

Input

msg — Data message from video capture interface
scalar

This message port receives data messages from a connected Video Capture Interface block. The messages process when the Task Manager block triggers task containing the Video Capture block. For more information on messages, see “Messages”.

Data Types: `SoCData`

Output

`Color` — Color component of image
height-by-*width* matrix

This port outputs a *height*-by-*width* matrix for that ***`Color`*** component, where the dimensions are the size of the image. The **Image size** parameter specifies the *height* and *width* dimensions.

Dependencies

The **Pixel format** parameter sets the color component format as either `RGB` or `YCbCr 4:2:2`.

Data Types: `uint8`

Parameters

Image size — Image size
`160x120` (default) | `320x240` | `640x480` | `800x600` | `custom`

Specify the *height* and *width* dimensions of the image matrix emitted by the color channel ports of this block. Specify `custom` to set custom image dimensions.

Image size ([width, height]) — Image size
`[320, 240]` (default) | 2-element vector of positive integers

Specify custom *height*-by-*width* dimensions of the image matrix emitted by the color channel ports.

Dependencies

To enable this parameter, set the **Image size** parameter to `custom`.

Pixel format — Format of the pixel data
`RGB` (default) | `YCbCr 4:2:2`

Specify the image data encoding as `RGB` or `YCbCr 4:2:2` triplets.

Sample time — Sample time in seconds
`-1` (default) | positive scalar

If used in a timer-driven task, enter the sample time of the task defined in the Task Manager block. If used in the event-driven task, enter `-1`.

Version History

Introduced in R2021a

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. For more information on generating code for SoC designs, see “Generate SoC Design”.
- To generate and execute C code for your SoC models, this block requires Embedded Coder and SoC Blockset Support Package for Embedded Linux Devices.
- To use this block, you must set **Hardware Board** to `Embedded Linux Board`. You can then follow the instructions to configure the board. For more information, see “Create an Embedded Linux Device Configuration” (SoC Blockset Support Package for Embedded Linux Devices).

When deployed to a supported hardware board, this block uses the V4L2 driver framework to capture images from video capture devices, such as a camera. To specify the video capture device and event triggers, use the Hardware Mapping tool. For more information on the V4L2 driver framework, see Video4Linux.

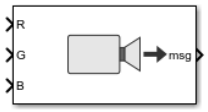
See Also

Hardware Mapping | Video Capture Interface | Video Display | Video Display Interface | Video Stream FIFO | Video Stream Connector | Video Test Sink | Video Test Source

External Websites
Video4Linux

Video Display

Display image samples from an algorithm to an output video device



Libraries:
SoC Blockset / Peripherals

Description

The Video Display block simulates behavior of the driver code that displays images to a video output, such as an LCD screen, on a hardware board. The output of this block must be connected to a Video Display Interface block that simulates a physical display screen on the hardware board. Place this block inside a task and connect to an algorithm to output a image each time the task executes. For more information on tasks, see “Timer-Driven Task” and “Event-Driven Tasks”.

Ports

Input

Color — Color component of image
height-by-*width* size matrix

Specify a *height*-by-*width* matrix, where the dimensions are the size of the image.

Dependencies

Use the **Pixel format** parameter to specify the color component format as either `RGB` or `YCbCr 4:2:2`.

Data Types: `uint8`

Output

msg — Data message to the video display interface
scalar

This port outputs data messages containing image data to a connected Video Display Interface block. For more information on messages, see “Messages”.

Data Types: `SoCData`

Parameters

Pixel format — Format of pixel data
`RGB` (default) | `YCbCr 4:2:2`

Specify the input image data encoding as `RGB` or `YCbCr 4:2:2` triplets.

Version History

Introduced in R2021a

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. For more information on generating code for SoC designs, see “Generate SoC Design”.
- To generate and execute C code for your SoC models, this block requires Embedded Coder and SoC Blockset Support Package for Embedded Linux Devices.
- To use this block, you must set **Hardware Board** to `Embedded Linux Board`. You can then follow the instructions to configure the board. For more information, see “Create an Embedded Linux Device Configuration” (SoC Blockset Support Package for Embedded Linux Devices).

When deployed to a supported hardware board, this block uses the V4L2 driver framework to capture images from video capture devices, such as a camera. To specify the video capture device and event triggers, use the Hardware Mapping tool. For more information on the V4L2 driver framework, see Video4Linux.

See Also

Hardware Mapping | Video Capture Interface | Video Capture | Video Display Interface | Video Stream FIFO | Video Stream Connector | Video Test Sink | Video Test Source

External Websites

Video4Linux

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”.

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".

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" and "Event-Driven Tasks", 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".

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".

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”.

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”.

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”.

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”.

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”.

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”.

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”.

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”.

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”.

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”.

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”.

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”
“What is Task Execution?”
“Task Duration”

Proxy Task

A placeholder for a task in your application

Libraries:
SoC Blockset / Processor Task Execution

Description

The Proxy Task block simulates the effect of a timer-driven task in your SoC application without an explicit implementation. This block can be used as a placeholder for timer-driven tasks to be developed in the future, or implemented simultaneously by another developer.

When added to your processor reference Model block, it causes the processor model, when set to schedule rates by ports, to display a periodic event port with a sample time equal to `Sample Time` parameter. Connect the periodic event port to a timer-driven task output port on the Task Manager block.

Ports

Input

Port_1 — Function-call from `Task Manager` block
scalar

A function-call signal that triggers the Proxy Task block when operating as an event-driven task.

Dependencies

To enable this port, set the `Type` property to `Event-driven`.

Data Types: `function-call`

Parameters

Type — Type of task
`Timer-driven` (default) | `Event-driven`

Select the type of task as either Timer-driven or Event-driven.

Sample Time — Sample time
1 (default) | positive scalar

Sample time of the block.

Note

- The `Sample Time` of the Proxy Task block must match the sample time of the corresponding timer-driven task from the Task Manager block.

- The `Sample Time` of the Proxy Task block should be unique within the model. When other blocks in the model use the same sample time, the duration of the task defined by Proxy Task block cannot be guaranteed in code generation.

Dependencies

To enable this parameter, set the `Type` property to `Timer-driven`.

Version History

Introduced in R2019b

See Also

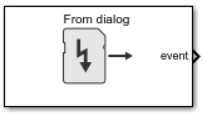
Task Manager | Testbench Task

Topics

“Timer-Driven Task”

Event Source

Simulate and playback recorded task events



Libraries:
SoC Blockset / I/O Data Source and Sink

Description

The Event Source block, connected to the Task Manager block, enables you to simulate tasks events in your Simulink model. Task timing data can be provided from the block mask, an external file, or from an input port driven by other model signals.

Ports

Input

data — Input data
numeric vector

Input data, specified as a numeric vector. The block converts this data into an event that corresponds to the rate of the input data. You can use this event to drive an event-driven task in the Task Manager block.

Dependencies

To enable this port, set the **Input** parameter to `From input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

Output

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.

Data Types: `rteEvent`

Parameters

Input — Source of input data
`From dialog` (default) | `From input port` | `From timeseries object`

Set the input data source for the block by selecting one of these values.

- `From dialog` — Input a one-dimensional array of data by using a function. Specify this function for the **Value** parameter.

- `From input port` — Input data and signals using input ports on the block.
- `From timeseries object` — Input data and time values using a timeseries object that you created in MATLAB. For more information see “Time Series Objects and Collections”.

Sample time — Time interval of sampling
`-1` (default) | nonnegative scalar

Specify a discrete time interval, in seconds, at which the block outputs data.

Dependencies

To enable this parameter, set the **Input** parameter to `From dialog`.

Object name — Name of timeseries object
`[ ]` (default) | timeseries object

Specify a `timeseries` object. This `timeseries` object provides the input data for the block. For more information about time series objects, see “Time Series Objects and Collections”.

Dependencies

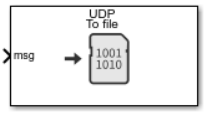
To enable this parameter, set the **Input** parameter to `From timeseries object`.

Version History

Introduced in R2020b

IO Data Sink

Record, output, or terminate input message



Libraries:
SoC Blockset / I/O Data Source and Sink

Description

The IO Data Sink block records, outputs, or terminates the received input message signal. The input of this block connects to the output of the TCP Write, UDP Write, or Register Write block. This block enables you to save the received input data to a file that you can play back using the IO Data Source block in the model. You can also terminate the signal or output the signals through an output port which can be fed as an input to IO Data Source block.

Ports

Input

msg — SoC message data
numeric vector

This port receives the data vector from the **msg** port of the processor io blocks, which includes Stream WriteTCP Write, UDP Write, or Register Write blocks.

Data Types: `SoCData`

Output

data — Output data
numeric vector

Output data, returned as a numeric vector. The block converts the received input message into a data signal.

Dependencies

To enable this port, set the **Output** parameter to `To output port`.

Data Types: `uint32` | `double` | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `int64` | `uint64` | `Boolean` | `fixedpoint`

length — Length of output data
nonnegative scalar

Length of output data, returned as a nonnegative scalar.

Dependencies

To enable this port, set the **Output** parameter to `To output port`.

Data Types: `double`

valid — Indication of valid data
Boolean scalar

Control signal that indicates whether the output data is valid. When this value is `1` (true), the value on the output **data** port is valid.

Dependencies

To enable this port, set the **Output** parameter to `To output port`.

Data Types: `Boolean`

done — Completion of data streaming
Boolean scalar

When **done** is `1`, the block has no more stream output data to return in the **data** port. When **done** is `0`, the block has more stream data to return in the **data** port.

Dependencies

To enable this port, set the **Device type** parameter to `Stream`.

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.

Dependencies

To enable this port, set the **Show port** parameter to `Data and Event`.

Data Types: `rteEvent`

Parameters

Output — Sink of output data from block
`To file` (default) | `To output port` | `To terminator`

Set the sink of output data from the block by selecting one of these values.

- `To file` — Save output data to a file.
- `To output port` — Output data and signals by using output ports on the block.
- `To terminator` — Terminate the received input signal.

Device type — Device type selection
`UDP` (default) | `TCP` | `Register` | `Stream`

Select a device type to enable the corresponding input data port.

- `UDP` — Enable the **msg** input port to receive UDP data as message from a **msg** port of UDP Write block.

- `TCP` — Enable the **msg** input port to receive TCP data as message from a **msg** port of TCP Write block.
- `Register` — Enable the **msg** input port to receive Register data as message from a **msg** port of Register Write block.
- `Stream` — Enable the **msg** input port to receive Stream data as message from a **msg** port of Stream Write block.

Show port — Enable output ports
`Data` (default) | `Data and Event`

Select one of these values to enable the corresponding output ports towards the writing source.

- `Data` — Enable only the **msg** input port.
- `Data and event` — Enable the **msg** input and **event** output ports.

Number of buffers — Number of data buffers
8 (default) | nonnegative scalar

Specify the number of data elements to store in the data queue. This parameter must match the **Number of buffers** parameter specified in the Memory Channel block.

Dependencies

To enable this parameter, set the **Device type** parameter to `Stream`.

Sample time — Time interval of sampling
`-1` (default) | nonnegative numeric scalar

Specify a discrete time interval, in seconds, at which the block outputs data. The default value `-1` inherits the sample time from the solver used for simulating the model.

Dataset name — Name of data file
no default | file path

Specify the full path to where you want to save the file on the host PC. This block saves the output data as a TGZ file. You can import this file into the model by using the IO Data Source block.

Dependencies

To enable this parameter, set the **Output** parameter to `To file`.

Source name — Name of dataset
no default

Specify a name for the output data source in which to save the data in the dataset file.

Dependencies

To enable this parameter, set the **Output** parameter to `To file`.

Data type — Data type of output data
`uint32` (default) | `double` | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `int64` | `uint64` | `boolean` | `fixedpoint`

Select the data type of the output data. This value must match the data type of the input data.

Dependencies

To enable this parameter, set the **Output** parameter to `To file` or `To output port`.

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”.

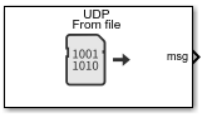
Embedded Coder does not generate code for this block. In the generated code, the device I/O connects directly to the TCP Write, UDP Write, or Register Write block.

See Also

IO Data Source | UDP Write | TCP Write | Stream Write | Register Write | Task Manager

IO Data Source

Play back recorded data



Libraries:
SoC Blockset / I/O Data Source and Sink

Description

The IO Data Source block enables you to import recorded hardware IO data and play it back in your Simulink model. The block converts the input data into a message signal that you can give as input to the TCP Read, UDP Read, Stream Read, or Register Read blocks, depending on the device type you choose. The playback of hardware IO data in your Simulink model helps you develop models with better accuracy than models developed by using randomly generated data during simulation.

When you develop models that use real hardware IO data during deployment, you can choose to use randomly generated synthetic data as hardware IO data in simulation. As physical hardware data accounts for various effects like data loss, time delay, and so on. If you use synthetic data as hardware IO data in simulation for such models, it leads to unexpected results when you deploy it in the hardware board. To evaluate and verify such models, using real hardware IO data during simulation is recommended. For more information on how to record hardware IO data and save it to your host computer, see the `DataRecorder` object.

Note If you have a IO Data Source block with **Input** set to `From file`, associated with a `Timer-driven` Task Manager block in your model and you plan to use a fixed-step solver, then enter a step size value lesser than the value set for the **Period** parameter in the Task Manager block. For example, suppose the value of **Period** specified in the Task Manager block is `0.1`, then choose a fixed-step size less than `0.1`.

Ports

Input

data — Input data
numeric vector

Input data, specified as a numeric vector. The block converts this data into a bus signal of the specific device type specified by the **Device type** parameter. Match the data type of this input data with the data type you select in the **Data type** parameter. The output bus signal consists of data values, length of data, and valid status of data.

Dependencies

To enable this port, set the **Input** parameter to `From input port`.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `int64` | `uint8` | `uint16` | `uint32` | `uint64` | `Boolean` | `fixed point`

length — Length of input data
nonnegative scalar

Length of input data, specified as a nonnegative scalar.

Dependencies

To enable this port, set the **Input source** parameter to `From input port`.

Data Types: `uint32`

valid — Valid data signal
Boolean scalar

When **valid** is `1`, the block captures the input data from the **data** and **length** ports. When **valid** is `0`, the block considers the input data as invalid and does not capture it.

Dependencies

To enable this port, set the **Input source** parameter to `From input port`.

Data Types: `Boolean`

done — Notification of freed buffer in memory
`False` (default) | `True`

This message port receives a notification from the connected Memory Channel or IO Data Sink block that a read transaction completed, and that a buffer in memory is available for writing.

Dependencies

To enable this port, set the **Device type** parameter to `Stream`.

Data Types: `Boolean`

Output

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.

Dependencies

To enable this port, set the **Show port** parameter to `Event` or `Data and event`.

Data Types: `rteEvent`

msg — SoC message data
numeric vector

This port sends the data vector as a message to the **msg** input port of processor I/O blocks, which includes Register Read, Stream Read, UDP Read, and TCP Read blocks.

Dependencies

To enable this port, set the **Show port** parameter to `Data` or `Data and Event`.

Data Types: `SoCData`

Parameters

Input — Source of input data
`From file` (default) | `From dialog` | `From input port` | `From timeseries object`

Set the input data source for the block by selecting one of these values.

- `From file` — Read data from a recorded data file at the same time interval at which it was recorded on the hardware board.
- `From dialog` — Input a one-dimensional array of data by using a function. Specify this function for the **Value** parameter.
- `From input port` — Input data and signals using input ports on the block.
- `From timeseries object` — Input data and time values using a timeseries object that you created in MATLAB. For more information see “Time Series Objects and Collections”.

Value — Value of source data
`uint32(1:1024)` (default) | function to generate input data

Specify a MATLAB function that creates a row vector of numeric data. This row vector is captured as the input data for the block.

Dependencies

To enable this parameter, set the **Input** parameter to `From dialog`.

Data type — Data type of input data
`uint32` (default) | `double` | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `int64` | `uint64` | `boolean` | `fixed point`

Select the data type of the input data to be received by the **data** port.

Dependencies

To enable this parameter, set the **Input** parameter to `From file`.

Device type — Device type of input data source
`UDP` (default) | `TCP` | `Register` | `Stream`

Select a device type to enable the corresponding output data port.

- `UDP` — Enables the **msg** output port to output UDP data as a message.
- `TCP` — Enables the **msg** output port to output TCP data as a message.
- `Register` — Enables the **msg** output port to output Register data as a message.
- `Stream` — Enables the **msg** output port to output Stream data as a message.

Dependencies

To enable this parameter, set the **Input** parameter to `From input port` or `From dialog`.

Sample time — Time interval of sampling
`-1` (default) | nonnegative scalar

Specify a discrete time interval, in seconds, at which the block outputs data.

Dependencies

To enable this parameter, set the **Input** parameter to `From dialog`.

Dimensions — Samples per frame
`1024` (default) | nonnegative scalar

Specify the size of the input data. The block reads this number of samples per frame during reading and playback in simulation.

Dependencies

To enable this parameter, set the **Input** parameter to `From file`.

Dataset name — Name of recorded file
no default | file path

Specify the full path to a recorded data file on the host PC or browse and select a file on the host PC. This block supports only TGZ files created by using the SoC Blockset data recording API.

Dependencies

To enable this parameter, set the **Input** parameter to `From file`.

Source name — Name of dataset
no default

Specify the dataset source name you want to use as the input source available within the recorded data specified in the **Dataset name** parameter. You can either type the name in the **Source name** box or click **Select** to select the name from the list of sources available in the recorded data file.

Dependencies

To enable this parameter, set the **Input** parameter to `From file`.

Number of buffers — Number of data buffers
`1024` (default) | nonnegative scalar

Specify the number of data elements to store in the input data queue.

Dependencies

To enable this parameter, set the **Device type** parameter to `Stream`.

Show port — Enable output ports
`Data` (default) | `Event` | `Data and event`

Select one of these values to enable the corresponding output ports.

- `Data` — Enable only the **msg** output port.
- `Event` — Enable only the **event** output port.
- `Data and event` — Enable the **msg** and **event** output ports.

Object name — Name of timeseries object
`[]` (default) | timeseries object

Specify a `timeseries` object. This `timeseries` object provides the input data for the block. For more information about time series objects, see “Time Series Objects and Collections”.

Dependencies

To enable this parameter, set the **Input** parameter to `From timeseries object`.

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”.

Embedded Coder does not generate code for this block. In the generated code, the device I/O connects directly to the TCP Read, UDP Read, Stream Read, or Register Read block.

See Also

IO Data Sink | Task Manager | TCP Read | UDP Read | Stream Read | Register Read | `DataRecorder`

Testbench Task

An external timer-driven task load on your SoC processor application

Libraries:
SoC Blockset / Processor Testbench

Description

The Testbench Task block simulates a timer-driven task load external to your software application. Using Testbench Task blocks, you can simulate the impact of your software application in the presence of other processes that compete for execution time of the processor.

Note As part of a processor test bench, the Testbench Task block must be placed in the top-level of your SoC model.

Ports

Input

Timer Task Function Call — Function-call input port from a timer-driven task
scalar

This port accepts a function-call event signal from a timer-driven task event port of the Task Manager block.

Version History

Introduced in R2019b

Extended Capabilities

C/C++ Code Generation

Generate C and C++ code using Simulink® Coder™.

Embedded Coder does not generate code for this block.

See Also

Task Manager | Proxy Task

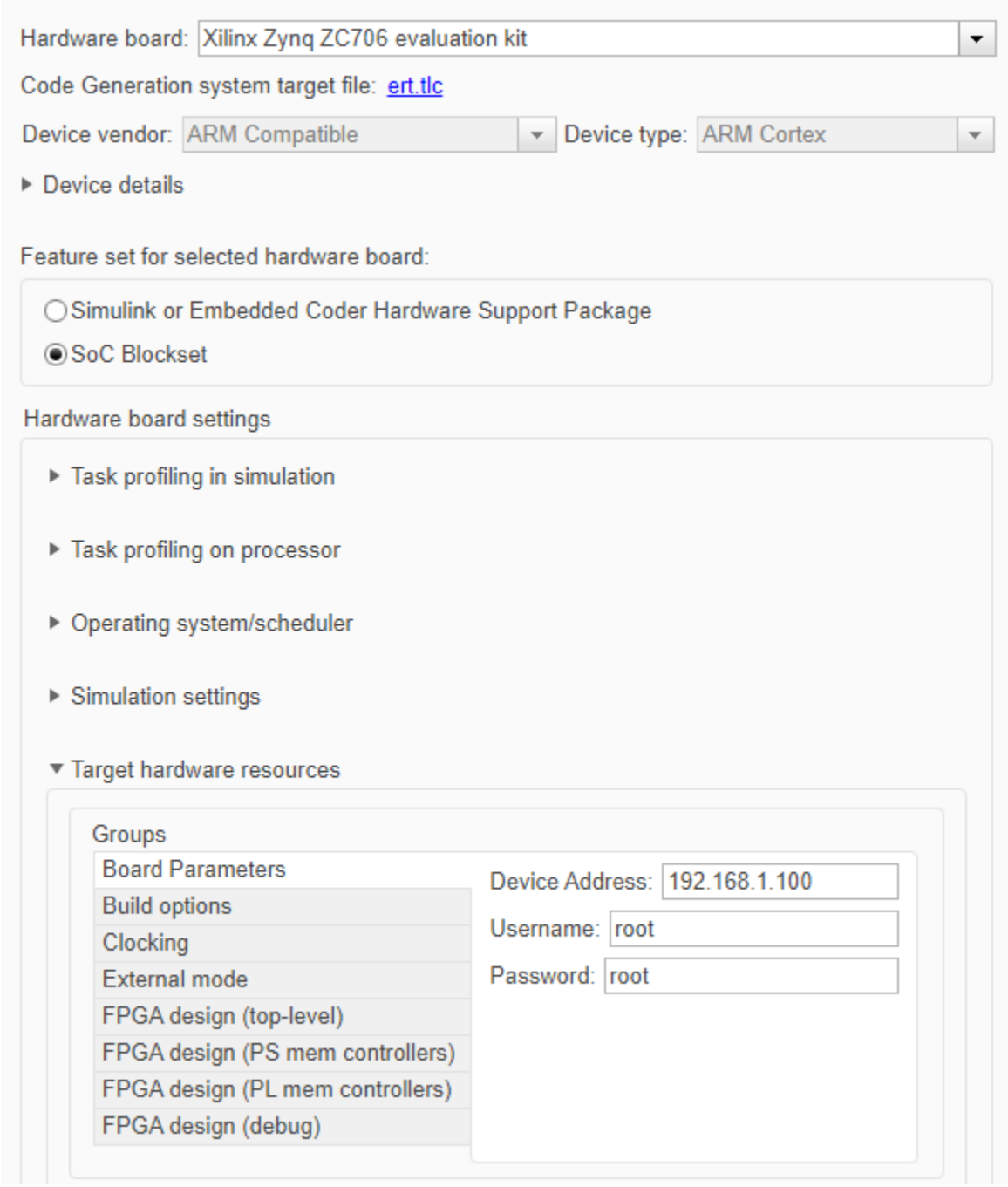
Topics

“Timer-Driven Task”

Configuration Parameters

Hardware Implementation Pane

Hardware Implementation Pane Overview



Hardware board settings

Parameter	Description	Default Value
“Processing Unit” on page 2-8	Processor for model reference block in the SoC model.	`None`

Design Mapping

Parameter	Description	Default Value
“Design Mapping” on page 2-9	Open the hardware mapping tool.	not applicable

Task profiling in simulation

Parameter	Description	Default Value
“Show in SDI” on page 2-10	Show the task execution data collected in simulation in the **Simulation Data Inspector** application.	`on`
“Save to file” on page 2-10	Save the task execution data to a file.	`on`
“Overwrite file” on page 2-10	Overwrite the last task execution data file.	`off`

Task profiling on processor

Parameter	Description	Default Value
“Show in SDI” on page 2-11	Show the task execution data collected on hardware in the **Simulation Data Inspector** application.	`off`
“Save to file” on page 2-11	Save the task execution data to a file.	`off`
“Overwrite file” on page 2-11	Overwrite the last task execution data file.	`off`
“Instrumentation” on page 2-11	Choose to perform code instrumentation or Kernel instrumentation.	`Code`
“Profiling duration” on page 2-11	Choose whether to perform Kernel profiling for an unlimited or limited time duration.	`Unlimited`

Operating system/scheduler

Parameter	Description	Default Value
“Operating System/Scheduler” on page 2-13	Specify the Kernel latency of the OS in simulation of a task.	`0`

Simulation Settings

Parameter	Description	Default Value
“Set random number generator seed” on page 2-14	Set the random number generator seed.	`off`
“Seed Value” on page 2-14	Specify the seed value for the simulation of task duration deviation.	`default`
“Cache input data at task start” on page 2-14	Cache the input data at the start of a task.	`off`

Board Parameters

Parameter	Description	Default Value
Device Address	Network address of hardware board or device.	`192.168.1.10`
Username	Login *`username`* on hardware board or device.	`root`
Password	Login password on hardware board or device.	`root`

Processor

Parameter	Description	Default Value
“Number of cores” on page 2-15	Set the number of CPU cores in the processor.	`1`

Board Options

Parameter	Description	Default Value
“Build Action” on page 2-18	Defines how **SoC Builder** tool responds when you build your model.	`Build, load, and run`

Clocking

Parameter	Description	Default Value
CPU Clock (MHz) on page 2-17	The CPU clock frequency in MHz.	`1000`

External Mode

Parameter	Description	Default Value
“Communication Interface” on page 2-19	Transport layer used to exchange data between the development computer and hardware.	`TCP/IP`
“Run external mode in a background thread” on page 2-19	Execute the external mode engine in the generated code in a background task.	`disabled`
“Port” on page 2-19	IP address port on hardware board.	`17725`
“Verbose” on page 2-20	Enable view of the external mode execution progress and updates in the Diagnostic Viewer.	`disabled`

FPGA design (top-level)

Parameter	Description	Default Value
“View/Edit Memory Map” on page 2-21	Choose whether to perform global synthesis or per IP core synthesis.	`Out of Context per IP`
“Include 'MATLAB as AXI Master' IP for host-based interaction” on page 2-21	Use host-based scripts with an integrated JTAG master on the target platform.	`on`
“Include processing system” on page 2-21	For processor-based platforms, include the processing system.	`on`
“Interrupt latency (s)” on page 2-21	The latency from hardware asserting an interrupt to the start of the interrupt service routine.	`0.00001`
“Register configuration clock frequency (MHz)” on page 2-21	The system configuration clock drives the configuration register interfaces for the vendor IP cores in the system.	`50`
“IP core clock frequency (MHz)” on page 2-21	The clock for all Simulink based generated HDL IP cores.	`100`

FPGA design (PS mem controllers)

The default values for these parameters vary by board.

Parameter	Description	Default Value
“Controller clock frequency (MHz)” on page 2-23	Frequency of datapath between PS memory interconnect and PS memory controller.	`200`
“Controller data width (bits)” on page 2-23	Bit width of datapath between PS memory interconnect and PS memory controller.	`64`
“Bandwidth derating (%)” on page 2-23	For every 100 clocks, will hold off all transaction execution for this number of clocks.	`2.3`
“First write transfer latency (clocks)” on page 2-23	Number of clock cycles between write request and start of transfer.	`4`
“Last write transfer latency (clocks)” on page 2-24	Number of clock cycles between the end of write transfer and completion of transaction.	`4`
“First read transfer latency (clocks)” on page 2-24	Number of clock cycles between read request and start of transfer.	`5`
“Last read transfer latency (clocks)” on page 2-24	Number of clock cycles between the end of read transfer and completion of transaction.	`1`

FPGA design (PL mem controllers)

The default values for these parameters vary by board.

Parameter	Description	Default Value
“Controller clock frequency (MHz)” on page 2-23	Frequency of datapath between PL memory interconnect and PL memory controller.	`200`
“Controller data width (bits)” on page 2-23	Bit width of datapath between PL memory interconnect and PL memory controller.	`64`
“Bandwidth derating (%)” on page 2-23	For every 100 clocks, will hold off all transaction execution for this number of clocks.	`2.3`
“First write transfer latency (clocks)” on page 2-23	Number of clock cycles between write request and start of transfer.	`4`

Parameter	Description	Default Value
“Last write transfer latency (clocks)” on page 2-24	Number of clock cycles between the end of write transfer and completion of transaction.	4
“First read transfer latency (clocks)” on page 2-24	Number of clock cycles between read request and start of transfer.	5
“Last read transfer latency (clocks)” on page 2-24	Number of clock cycles between the end of read transfer and completion of transaction.	1

FPGA design (mem channels)

Parameter	Description	Default Value
“Interconnect clock frequency (MHz)” on page 2-25	Frequency of the master datapath to the interconnect controller in MHz.	200
“Interconnect data width (bits)” on page 2-25	Data width of master datapath to interconnect controller in bits.	64
“Interconnect FIFO depth (num bursts)” on page 2-25	Maximum number of bursts that can be buffered before data is dropped.	12
“Interconnect almost-full depth” on page 2-25	When the almost full depth is reached, the attached channel protocol interface block asserts back pressure to the data source.	8

FPGA design (debug)

Parameter	Description	Default Value
“Memory channel diagnostic level” on page 2-26	The internal operation of the memory channel can be instrumented for debug or diagnostic analysis.	Basic diagnostic signals
“Include AXI interconnect monitor” on page 2-26	Gather performance metrics of the memory interconnect such as data throughput, latency, and number of bursts executed.	off
“Trace capture depth” on page 2-26	Maximum number of Trace entries to be logged in trace mode	1024

Hardware Board Settings

Processing Unit

Choose the processor in the SoC or MCU onto which to deploy the model block in the SoC model. The top level SoC model is set to `None`.

Settings

Default: `None`

See Also

“Multiprocessor Execution” | “Run Multiprocessor Models in External Mode”

Design Mapping

View/Edit Hardware Mapping

Open the **Hardware Mapping** tool to map the simulation parameters of interface blocks to deployed configuration values of peripherals and to map tasks in the model to available hardware interrupt sources for the selected hardware board.

See Also

“Hardware Implementation Pane” on page 2-2

Task Profiling in Simulation

Show in SDI

Show task execution data collected in simulation in the Simulation Data Inspector (SDI). For more information on visualizing tasks in SDI, see “Task Visualization in Simulation Data Inspector”.

Settings

Default: `off`

Save to file

Save task execution data to a file. For more information on recording task execution data, see “Recording Tasks for Use in Simulation”.

Settings

Default: `off`

Overwrite file

Overwrite last task execution data file. For more information on recording task execution data, see “Recording Tasks for Use in Simulation”.

Settings

Default: `off`

Task Profiling on Processor

Show in SDI

Show the task execution data collected on a processor in the Simulation Data Inspector (SDI) application. For more information on visualizing tasks in the SDI application, see “Task Visualization in Simulation Data Inspector”.

Default: `off`

Save to file

Save task execution data to a file. For more information on recording task execution data, see “Recording Tasks for Use in Simulation”.

Default: `off`

Overwrite file

Overwrite the last task execution data file. For more information on recording task execution data, see “Recording Tasks for Use in Simulation”.

Default: `off`

Instrumentation

Choose the instrumentation method using which task execution data has to be collected from processor to display on SDI. For more information, see “Kernel Instrumentation Profiler” and “Code Instrumentation Profiler”.

Default: `Code`

Profiling duration

Choose the profiling duration for Kernel profiling.

If you select `Kernel` to specify Kernel profiling, set **Profiling duration** to `Unlimited` or `Limited`.

- `Unlimited` — This option performs Kernel profiling on the hardware and streams it to the host PC for an unlimited time duration. Kernel profiling for an unlimited time duration on hardware with low free disk storage or a model with high task rates can result in packet loss of profiling data streamed from the hardware. This packet loss depends on the free memory available on the host PC on which you run MATLAB.
- `Limited` — This option performs Kernel profiling on the hardware and streams it to host PC for a limited time duration. Kernel profiling for a limited time duration on hardware does not result in

packet loss of profiling data streamed from hardware. The maximum time duration for limited Kernel profiling depends on the RAM and free disk storage available on the hardware board on which Kernel profiling is performed.

Default: `Unlimited`

Operating System/Scheduler

Kernel latency

Sets the simulated delay in the start of a task expected by the kernel latency of the operating system (OS). For more information on kernel latency, see “Effect Kernel Latency on Task Execution”.

Settings

Default: `default`

Simulation Settings

Set random number generator seed

Enable explicit specification of random number seed for task or memory simulation.

Settings

Default: `off`

Seed Value

Random number generator seed for the simulation of task duration deviation of the Task Manager block.

Settings

Default: `default`

Cache input data at task start

Cache the data from signals at the start of task execution. Otherwise, evaluate with the signal data at the end of the task execution.

See Also

Task Manager

Processor

Number of cores

Set the number of CPU cores in the processor.

Settings

Default: 1, positive scalar

Board Parameters

Device Address

Enter the IP address or host name of the hardware board.

When you use the Support Package Installer to update the firmware on the board, the Support Package Installer automatically gets the value of the IP address from the board and applies it to this parameter.

If you swap boards, or change the IP address of the board, get the value of the new IP address and enter it here.

Settings

Default: `192.168.1.10`

Username

Enter the root user name for Linux running on the hardware board.

When you use the Support Package Installer to update the hardware board firmware, the Support Package Installer automatically applies the value you entered there to this parameter.

Settings

Default: `root`

Password

Enter the root password for Linux running on the hardware board.

When you use the Support Package Installer to update the hardware board firmware, the Support Package Installer automatically applies the value you entered there to this parameter.

Settings

Default: `root`

See Also

“Hardware Implementation Pane” (SoC Blockset Support Package for Xilinx Devices)

Clocking

CPU Clock (MHz)

The frequency of the CPU core clock in MHz.

Settings

Default: 1000

Build Action

Defines how **SoC Builder** tool builds your model.

Settings

Default: `Build, load, and run`

`Build, load, and run`

With this option, launching **SoC Builder**:

1. Generates code from the model.
2. Compiles and links the code into an executable with libraries.
3. Loads the executable and libraries into the hardware board.
4. Runs the executable in the hardware board.

`Build`

With this option, launching **SoC Builder**:

1. Generates code from the model.
2. Compiles and links the code into an executable with libraries.

This option does not load and run the executable on the hardware board.

See Also

SoC Builder

External Mode

Communication Interface

Select the transport layer that external mode uses to exchange data between the host computer and the target hardware.

Settings

Default:`TCP/IP`, `XCP on TCP/IP`

Run external mode in a background thread

Force the external mode task in the generated code to execute in a background thread.

When external mode runs in the model thread, external mode executes after each execution step of the model and collects data at the base rate of the model. When model code consumes most of the thread execution time in each time step, external mode execution overruns into the next time step. This overrun delays the start of the next model execution step and degrades the real-time behavior of the deployed model.

You can configure external mode to run in a background thread. When external mode runs in a background thread, it executes in the time between the end of model code of one time step and the start of the next time step. By not blocking the model step, external mode can be used in systems that require real-time execution. This configuration enables direct observation of the deployed model on the hardware board as it would behave in standalone operation.

When model code consumes most of the execution time for each time step, external mode in the background thread starves for execution time. Without sufficient time to collect and transmit data from the hardware board to the host computer, data packets drop. This case results in gaps in the data logging.

To help avoid dropped data packets in deployed models where real-time execution takes priority over data logging, configure external mode to operate as a background task.

Note Enabling the `Run external mode in a background thread` parameter is not recommended for models that use a very small time step or that might encounter task overruns. These situations can cause Simulink to become unresponsive.

Settings

Default:`disabled`

Port

Enter the port for the IP address of the hardware board.

Settings

Default: 17725

Verbose

To view the external mode execution progress and updates in the Diagnostic Viewer or in the MATLAB command window, select this check box.

Settings

Default: `disabled`

FPGA design (top-level)

View/Edit Memory Map

Click to view and edit the FPGA memory map.

Include 'MATLAB as AXI Master' IP for host-based interaction

Use host-based scripts with an integrated JTAG master on the target platform to initialize configuration registers and memory regions in the generated design. You can also use it to interact with the design while running in order to read back diagnostic information. The JTAG master can be used instead of or in addition to an embedded processor on the target platform.

Settings

Default: `on`, `off`

Include processing system

For processor-based platforms, include the processing system. The processing system must be included when using Embedded Coder to generate embedded software.

Settings

Default: `off`, `on`

Interrupt latency (s)

The latency from hardware asserting an interrupt to the start of the interrupt service routine.

Settings

Default: `0.00001`

Register configuration clock frequency (MHz)

The system configuration clock drives the configuration register interfaces for the vendor IP cores in the system. User-authored Simulink IP cores will utilize the parameter below for its configuration register bus.

Settings

Default: `50`

IP core clock frequency (MHz)

The clock for all Simulink-based generated HDL IP cores. A single clock drives all IP and is used for both datapath and configuration register logic.

Settings

Default: `100`

FPGA design (mem controllers)

Memory controller parameters. The default values for these parameters vary by board.

Controller clock frequency (MHz)

Frequency of datapath between memory interconnect and memory controller.

The clock rate used to drive transactions to the external memory. The controller clock frequency determines the overall system bandwidth for external memory that must be shared among all the masters in the model.

Settings

Default: `200`

Controller data width (bits)

Set the width, in bits, of the datapath between the memory controller and the memory interconnect.

Settings

Default: `64`

Bandwidth derating (%)

Model memory transaction inefficiencies specified by a derating percentage value. For every 100 clocks, memory transaction execution is paused for the number of clocks equal to **Bandwidth derating**. To set this parameter, measure the maximum bandwidth on your board and reflect the bandwidth derating from your board in this parameter. See an example in “Analyze Memory Bandwidth Using Traffic Generators”.

Settings

Default: `2.3`

First write transfer latency (clocks)

Specify the delay, in clock cycles, between a write request and the start of a transfer.

This delay is the number of clock cycles between making a request to the memory controller and until it returns a response. It is reflected in the **Logic Analyzer** waveforms as the time that the memory controller state remains as `BurstAccepted`. For more information about viewing waveforms in simulation, see “Buffer and Burst Waveforms”.

To set this value, measure the clock cycles between the burst-request and start of transfer on your board. For instructions for extracting this information from a hardware execution, see “Configuring and Querying the AXI Interconnect Monitor”.

Settings

Default: 4

Last write transfer latency (clocks)

Specify the delay in clock cycles between the end of a memory transfer and the end of a write transaction.

To set this value, measure the clock cycles between the end of the burst and the completion of the transaction on your board. For instructions for extracting this information from a hardware execution, see “Configuring and Querying the AXI Interconnect Monitor”.

Settings

Default: 4

First read transfer latency (clocks)

Specify the delay, in clock cycles, between a read request and the start of a transfer.

This delay is the number of clock cycles between making a request to the memory controller and until it returns a response. It is reflected in the **Logic Analyzer** waveforms as the time that the memory controller state remains as `BurstAccepted`. For more information about viewing waveforms in simulation, see “Buffer and Burst Waveforms”.

To set this value, measure the clock cycles between the burst-request and start of transfer on your board. For instructions for extracting this information from a hardware execution, see “Configuring and Querying the AXI Interconnect Monitor”.

Settings

Default: 5

Last read transfer latency (clocks)

Specify the delay in clock cycles between the end of a memory transfer and the end of a read transaction.

To set this value, measure the clock cycles between the end of the burst and the completion of the transaction on your board. For instructions for extracting this information from a hardware execution, see “Configuring and Querying the AXI Interconnect Monitor”.

Settings

Default: 1

FPGA design (mem channels)

Interconnect clock frequency (MHz)

Frequency of the master datapath to the interconnect controller in MHz.

Settings

Default: `200`

Interconnect data width (bits)

Data width of master datapath to interconnect controller in bits.

Settings

Default: 64

Interconnect FIFO depth (num bursts)

Specify depth of data FIFO, in units of bursts. When the writer has no buffers to write to, the FIFO can absorb data until a buffer becomes available. This value is the maximum number of bursts that can be buffered before data gets dropped.

Settings

Default: 12

Interconnect almost-full depth

Specify a number that asserts a backpressure signal from the channel to the data source. To avoid dropping data, set a high watermark, allowing the data producer enough time to react to backpressure. This number must be smaller than the FIFO depth.

Settings

Default: 8

FPGA design (debug)

Memory channel diagnostic level

The internal operation of the memory channel can be instrumented for debug or diagnostic analysis. When enabled a `diag` output port will be added to the block.

Settings

Default: `Basic diagnostic signals`, `No debug`

Include AXI interconnect monitor

Gather performance diagnostics of the AXI memory interconnect such as data throughput, latency, and number of bursts executed. You can use the AXI master or a processing system on the target to gather the information. When using an AXI master, a host-based script can plot the data using MATLAB. These figures can then be compared against the simulation results.

Settings

Default: `off`

Trace capture depth

Maximum number of Trace entries to be logged in trace mode, choose the depth in powers of 2.

Settings

Default: `1024`

Hardware Mapping Peripherals for Xilinx Processors Properties

Select the configurations for peripherals in the model deployed to the hardware board

Description

The peripheral configurations for devices listed in the **Hardware Mapping** tool for Xilinx processors appear in the selected block in the **Browser** > **Peripherals**. This table shows the association between the driver block, simulation interface block, and **Hardware Mapping** tool hardware configuration.

Driver Block	Interface Block	Hardware Configuration
Video Capture	Video Capture Interface	"Video Capture" on page 2-0
Video Display	Video Display Interface	"Video Display" on page 2-0
Audio Capture	Audio Capture Interface	"Audio Capture" on page 2-0
Audio Playback	Audio Playback Interface	"Audio Playback" on page 2-0

Use the information provided in the properties group to choose the appropriate configuration.

Properties

Video Capture

`Device name` — VLS4 device mapping
`/dev/video0` (default) | hardware path of video device

This parameter specifies the VLS4 video device to use in the generated code as a Linux hardware path.

Video Display

`Display title` — Title of video display
`My Display` (default) | string

This parameter specifies the title of the video viewer shown on the screen of a connected monitor.

Audio Capture

`Device name` — ALSA device mapping
`hw:2,0` (default) | `hw:`*X*`,`*Y*

This parameter specifies the ALSA hardware card, *X*, and device, *Y*, mapping on the embedded Linux device.

`Audio sampling frequency` — Sampling frequency of audio device
44100 (default) | positive scalar integer

This parameter specifies the audio sampling frequency of the device managed by the ALSA driver. The selected value must be supported by the embedded Linux peripheral device.

Audio Playback

`Device name` — ALSA device mapping
`hw:2,0` (default) | `hw:`*X*`,`*Y*

This parameter specifies the ALSA hardware card, *X*, and device, *Y*, mapping on the embedded Linux device.

`Audio sampling frequency` — Sampling frequency of audio device
44100 (default) | positive scalar integer

This parameter specifies the audio sampling frequency of the device managed by the ALSA driver. The selected value must be supported by the peripheral device and the ALSA driver on your embedded Linux device.

Version History

Introduced in R2022b

See Also

Video Capture | Video Capture Interface | Video Display | Video Display Interface | Audio Capture | Audio Capture Interface | Audio Playback | Audio Playback Interface

Hardware Mapping Peripherals for Intel Processors Properties

Select the configurations for the software tasks and peripherals in the model deployed to the hardware board

Description

The peripheral configurations for devices listed in the **Hardware Mapping** tool for Intel processors.

Driver Block	Interface Block	Hardware Configuration
Video Capture	Video Capture Interface	“Video Capture” on page 2-0
Video Display	Video Display Interface	“Video Display” on page 2-0
Audio Capture	Audio Capture Interface	“Audio Capture” on page 2-0
Audio Playback	Audio Playback Interface	“Audio Playback” on page 2-0

Properties

Video Capture

`Device name` — VLS4 device mapping
`/dev/video0` (default) | hardware path of video device

This parameter specifies the VLS4 video device to use in the generated code as a Linux hardware path.

Video Display

`Display title` — Title of video display
`My Display` (default) | string

This parameter specifies the title of the video viewer shown on the screen of a connected monitor.

Audio Capture

`Device name` — ALSA device mapping
`hw:2,0` (default) | `hw:`*X*`,`*Y*

This parameter specifies the ALSA hardware card, *X*, and device, *Y*, mapping on the embedded Linux device.

`Audio sampling frequency` — Sampling frequency of audio device
44100 (default) | positive scalar integer

This parameter specifies the audio sampling frequency of the device managed by the ALSA driver. The selected value must be supported by the embedded Linux peripheral device.

Audio Playback

`Device name` — ALSA device mapping
`hw:2,0` (default) | `hw:X,Y`

This parameter specifies the ALSA hardware card, *X*, and device, *Y*, mapping on the embedded Linux device.

`Audio sampling frequency` — Sampling frequency of audio device
44100 (default) | positive scalar integer

This parameter specifies the audio sampling frequency of the device managed by the ALSA driver. The selected value must be supported by the peripheral device and the ALSA driver on your embedded Linux device.

Version History

Introduced in R2022b

Hardware Mapping Peripherals for Texas Instruments C2000 Processors Properties

Select the configurations for the peripherals in the model deployed to the hardware board

Description

The peripheral configurations for devices listed in the **Hardware Mapping** tool for Texas Instruments C2000 processors appear in the selected block in the **Browser > Peripherals**. This table shows the association between the driver block, simulation interface block, and **Hardware Mapping** tool hardware configuration.

Driver Block	Interface Block	Hardware Configuration
ADC Read	ADC Interface	"ADC" on page 2-0
PWM Write	PWM Interface	"PWM" on page 2-0

Properties

ADC

`Module` — Hardware ADC Module
`A` (default) | `B` | `C` | `D`

Select the ADC module `A` through `D` on the hardware board.

`Start of conversion` — Start of conversion trigger
`SOC0` (default) | `SOC0` | ... | `SOC15`

Identify the start-of-conversion trigger by number.

`Resolution` — Resolution of digital conversion
`12-bit (Single-ended input)` (default) | `16-bit (Differential inputs)`

Select the resolution of the digital conversion output.

`Conversion channel` — Input channel to apply ADC
`Internal` (default) | `Undefined` | `Interrupt name`

Select the input channel to which this ADC conversion applies.

`SOCx Acqusition window (cycles)` — Length of ADC acquisition period
positive scalar integer

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.

`SOCx Trigger source` — SoC trigger source
`Software` | `Timer` *`x`* `TINT`*`x`*`n` | `GPIO ADCEXTSOC` | `ePWM`*`x`* `ADCSOCA`

Select the event source that triggers the start of the conversion.

`ADCINT will trigger SOCx` — Use `ADCINT` interrupt to trigger start of conversion
`No ADCINT` (default) | `ADCINT1` | `ADCINT2`

At the end of conversion, use the `ADCINT1` or `ADCINT2` interrupt to trigger a start of conversion. 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.

`Enable interrupt at EOC` — Enable post interrupts when the ADC triggers end of conversion pulses
`false` (default) | `true`

Enable post interrupts when the ADC triggers EOC pulses. When you select this option, the dialog box displays the **Interrupt selection** and **Interrupt continuous mode** options.

`Interrupt selection` — ADC interrupt selection
`ADCINT1` (default) | `ADCINT2` | `ADCINT3` | `ADCINT4`

Select which `ADCINT#` interrupt the ADC posts to after triggering an EOC pulse.

`Interrupt continuous mode` — Generate new EOC signal overriding previous interrupt flag status
`false` (default) | `true`

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

`PWM`

`PWM Module` — Indicates which `ePWM` module to use
`ePWM1` (default) | `ePWM2` | ... | `ePWM`*x*

Select the appropriate `ePWM` module, `ePWM`*x*, where `x` is a positive integer.

`High speed clock divider` — High speed time base clock prescaler divider `HSPCLKDIV`
`1` (default) | `2` | `4` | `6` | `8` | `10` | `12` | `14`

Set the high speed time base clock prescaler divider, `HSPCLKDIV`.

`Timerbase clock divider` — Time base clock `TBCLK` prescaler divider corresponding to `CLKDIV`
`1` (default) | `2` | `4` | `8` | `16` | `32` | `64` | `128`

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 = PWM clock/(HSPCLKDIV * CLKDIV)`.

For example, the default values of both `CLKDIV` and `HSPCLKDIV` are 1, and the default frequency of PWM clock is 200 MHz, so: `TBCLK` in Hz = 200 MHz/(1 * 1) = 200 MHz `TBCLK` in seconds = 1/`TBCLK` in Hz = 1/200 MHz = 0.005 µs.

`Period (clock cycles)` — Period of `ePWM` counter
`1` (default) | `2` | `4` | `8` | `16` | `32` | `64` | `128`

Set the period of the `ePWM` counter waveform.

The timer period is in clock cycles:

Count Mode	Calculation	Example
`Up or down`	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 is `TCTR = (TBPRD + 1) * TBCLK`, where `TCTR` 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 `TCTR` = 50 µs. For the default action settings on the `ePWM`*x* tab, the `ePWM` period = 50 µs.
`Up-down`	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 is `TCTR = 2 * TBPRD * TBCLK`, where `TCTR` 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 `TCTR` = 100 µs. For the default action settings on the `ePWM`*x* tab, the `ePWM` period = 100 µs.

The initial duty cycle of the waveform from the time the PWM peripheral starts operation until the ePWM input port receives a new value for the duty cycle is Timer period / 2.

`Initialize CMP`*x*` count (clock cycles)` — Initialize the CMP*x* count
`0` (default) | positive integer

Set the initial count value of the comparator in clock cycles.

`Enable phase offset` — Enable the timer phase offset
false (default) | true

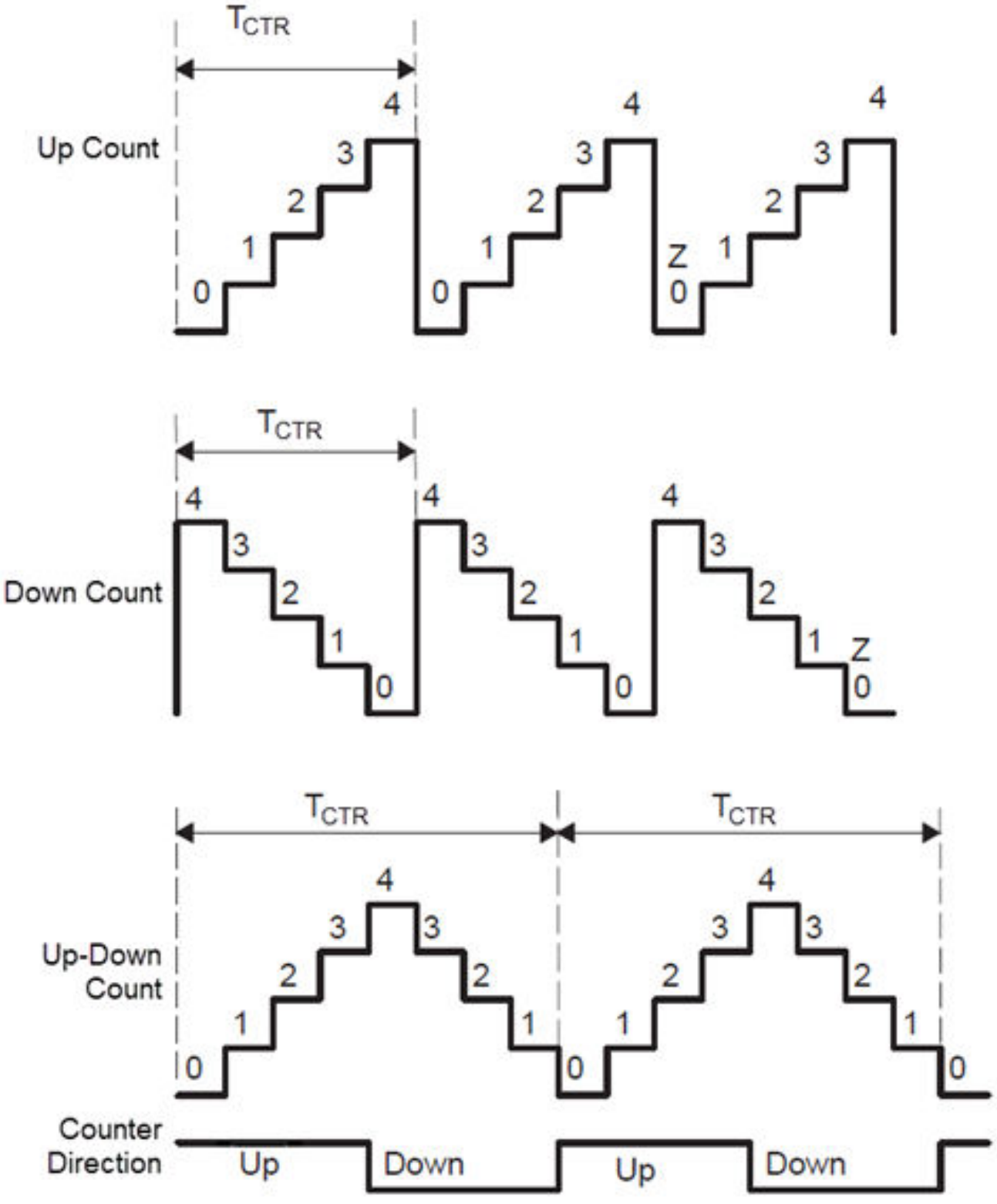
Enables to provide a timer phase offset value.

`Timer phase offset` — Timer phase offset
`0` (default) | integer between `0` and `65535`

The specified offset value is loaded in the time base counter on a synchronization event. Enter the phase offset value, `TBPHS`, in `TBCLK` cycles from 0 to 65535.

`Count mode` — Indicates counting mode of ePWM counter
`Up-Down` (default) | `Down` | `Up`

Specify the counting mode of the PWM internal counter. This figure shows three counting waveforms.


TCTR
Up Count
0
1
2
3
4
Z
Down Count
Up-Down Count
Counter Direction
Up
Down

`Action on counter=zero` — Behavior of action qualifier (AQ) submodule at zero count
`Do nothing` (default) | `Clear` | `Set` | `Toggle`

This group determines 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 `ePWMA` circuit. The `ePWMB` always generates a complement signal of `ePWMA`.

`Action on counter=period` — Behavior of action qualifier (AQ) submodule at period count
`Do nothing` (default) | `Clear` | `Set` | `Toggle`

This group determines 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 `ePWMA` circuit. The `ePWMB` always generates a complement signal of `ePWMA`.

`Action on counter=CMP`*`x`*` on `*`direction`*` count` — Behavior of Action Qualifier (AQ) submodule for the comparator (CMP) on for the given direction count
`Clear` (default) | `Do nothing` | `Set` | `Toggle`

This group determines 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 `ePWMA` circuit. The `ePWMB` always generates a complement signal of `ePWMA`.

`Enable shadow mode` — Enable the shadow mode
`Disable` (default) | `Enable`

When shadow mode is not enabled, the `CMPA` register refreshes immediately. Provide different reload mode for `CMPA` register.

`Reload CMP`*`x`*` register` — 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)` | `Freeze`

The time when the counter period resets based on the following condition:

- `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.

`ADC Start of conversion for ePWM module` — Trigger condition for an ADC start of the conversion event
`Counter equals to zero (CTR=Zero)` (default) | `Counter equals to period (CTR=PRD)` | `Counter equals to Zero or period (CTR=Zero or CTR=PRD)` | `Disable` | `Counter is `*`direction`*` and equal to CMP`*`x`*

This parameter specifies the counter match condition that triggers an ADC start of the conversion event. The choices are:

- `Counter equals to zero (CTR=Zero)` – Triggers an ADC start of the conversion event when the `ePWM` counter reaches 0.
- `Counter equals to period (CTR=PRD)` – Triggers an ADC start of the conversion event when the `ePWM` counter reaches the period value.
- `Counter equals to Zero or period (CTR=Zero or CTR=PRD)` – Triggers an ADC start of the conversion event when the time base counter, `TBCTR`, reaches zero or when the time base counter reaches the period, `TBCTR = TBPRD`.
- `Disable` – Disable ADC start of conversion event.
- `Counter is` *`direction`* `and equal to CMP`*`x`* – Triggers an ADC start of the conversion event when the counter equals the specified comparator and the counter *`direction`* is either `incrementing` or `decrementing`.

`ePWM interrupt` — Generate ISR for ePWM

`Disable` (default) | `Counter equals to zero (CTR=Zero)` | `Counter equals to period (CTR=PRD)` | `Counter equals to Zero or period (CTR=Zero or CTR=PRD)` | `Counter is` *`direction`* `and equal to CMP`*`x`*

This parameter registers that an interrupt occurs for the specified event and generates interrupt service routine (ISR) code to be used by the Task Manager. The choices are:

- `Counter equals to zero (CTR=Zero)` – Generates an ISR for when the `ePWM` counter reaches 0.
- `Counter equals to period (CTR=PRD)` – Generates an ISR for when the `ePWM` counter reaches the period value.
- `Counter equals to Zero or period (CTR=Zero or CTR=PRD)` – Generates an ISR for when the time base counter, `TBCTR`, reaches zero or when the time base counter reaches the period, `TBCTR = TBPRD`.
- `Disable` – Disable ISR generation.
- `Counter is` *`direction`* `and equal to CMP`*`x`* – Generates an ISR for when the counter equals the specified comparator and the counter *`direction`* is either `incrementing` or `decrementing`.

`Dead band (cycles)` — Enables the phase offset

`0` (default) | integer between `0` and `65535`

This parameter specifies the deadband delay for rising edge and falling edge in time-base clock cycles.

Version History

Introduced in R2022b

Hardware Mapping Peripherals for Embedded Linux Processors Properties

Select the configurations for peripherals in the model deployed to the hardware board

Description

The peripheral configurations for devices listed in the **Hardware Mapping** tool for embedded Linux processors appear in the selected block in the **Browser** > **Peripherals**. This table shows the association between the driver block, simulation interface block, and **Hardware Mapping** tool hardware configuration.

Driver Block	Interface Block	Hardware Configuration
Video Capture	Video Capture Interface	“Video Capture” on page 2-0
Video Display	Video Display Interface	“Video Display” on page 2-0
Audio Capture	Audio Capture Interface	“Audio Capture” on page 2-0
Audio Playback	Audio Playback Interface	“Audio Playback” on page 2-0

Use the information provided in the properties group to choose the appropriate configuration.

Properties

Video Capture

`Device name` — VLS4 device mapping
`/dev/video0` (default) | hardware path of video device

This parameter specifies the VLS4 video device to use in the generated code as a Linux hardware path.

Video Display

`Display title` — Title of video display
`My Display` (default) | string

This parameter specifies the title of the video viewer shown on the screen of a connected monitor.

Audio Capture

`Device name` — ALSA device mapping
`hw:2,0` (default) | `hw:`*X*`,`*Y*

This parameter specifies the ALSA hardware card, *X*, and device, *Y*, mapping on the embedded Linux device.

`Audio sampling frequency` — Sampling frequency of audio device
44100 (default) | positive scalar integer

This parameter specifies the audio sampling frequency of the device managed by the ALSA driver. The selected value must be supported by the embedded Linux peripheral device.

Audio Playback

`Device name` — ALSA device mapping
`hw:2,0` (default) | `hw:`*X*`,`*Y*

This parameter specifies the ALSA hardware card, *X*, and device, *Y*, mapping on the embedded Linux device.

`Audio sampling frequency` — Sampling frequency of audio device
44100 (default) | positive scalar integer

This parameter specifies the audio sampling frequency of the device managed by the ALSA driver. The selected value must be supported by the peripheral device and the ALSA driver on your embedded Linux device.

Version History

Introduced in R2022b

See Also

Video Capture | Video Capture Interface | Video Display | Video Display Interface | Audio Capture | Audio Capture Interface | Audio Playback | Audio Playback Interface

Functions

getData

Get data from file reader

Syntax

```
rd = getData(fr,sourceName)
```

Description

`rd = getData(fr,sourceName)` returns the data recorded from the specified source in the file reader. The `fr` input is an `socFileReader` object. The `sourceName` is the source name specified when saving the file by using the `save` object function.

Examples

Read Data from File Reader

Create a file reader to read data from the specified TGZ-compressed file.

```
fr = socFileReader('UDPDataReceived.tgz');
```

Get the data of a specified source from the file using the `getData` function.

```
rd = getData(fr,'UDPDataReceived-Port25000');
```

Input Arguments

`fr` — File reader

`socFileReader` object

File reader, returned as an `socFileReader` object.

`sourceName` — Name of recorded data source

character vector

Name of a recorded data source in `fr`, specified as a character vector. The function returns the recorded data of this specified source.

Output Arguments

`rd` — Data from recorded source

`timeseries` object

Data from recorded source, returned as a `timeseries` object.

Data Types: `timeseries`

Version History

Introduced in R2019a

See Also

`record` | `DataRecorder` | `save`

soc.recorder

Create data recording session

Syntax

```
dr = soc.recorder(hw)
```

Description

`dr = soc.recorder(hw)` creates a data recording session, `dr` on the SoC hardware board connected through `hw`. The `hw` input is a connection to an SoC hardware board, established using the `socHardwareBoard` function. The data recording session is a `DataRecorder` object. `hw` is an `soc.internal.zynq` or `soc.internal.intelsoc` object created using the `socHardwareBoard` function.

Examples

Record Data From SoC Hardware Board

Create a connection from MATLAB to the specified SoC hardware board using the IP address, username, and password of the board.

```
hw = socHardwareBoard('Xilinx Zynq ZC706 evaluation kit','hostname','192.168.1.18','username','ro
```

Create a data recording session on the SoC hardware board by using the `hw` object. The resulting `DataRecorder` object represents the data recording session on the SoC hardware board.

```
dr = soc.recorder(hw)
```

```
dr =

  DataRecorder with properties:

         HardwareName: 'Xilinx Zynq ZC706 evaluation kit'
              Sources: {}
            Recording: false
```

List the input sources added to the data recording session.

```
dr.Sources(hw)
```

```
ans =

  1×0 empty cell array
```

By default, `soc.recorder` objects have no added input sources. To add an input source to the data recording session, first create an input source object by using the `soc.iosource` function. For this example, create an User Datagram Protocol (UDP) source object.

```
udpSrc = soc.iosource(hw,'UDP Receive')
```

```
udpSrc =

  soc.iosource.UDPRead with properties:

   Main
              LocalPort: 25000
             DataLength: 1
               DataType: 'uint8'
      ReceiveBufferSize: -1
           BlockingTime: 0
    OutputVarSizeSignal: false
             SampleTime: 0.1000
          HideEventLines: true

  Show all properties
```

Add this UDP source object to the data recording session by using the `addSource` object function.

```
addSource(dr,udpSrc,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
dr.Sources
```

```
ans =

  1×1 cell array

    {'UDPDataOnPort25000'}
```

Call the setup function to initialize all hardware peripheral input sources added to the data recording session, and start the data recording process.

```
setup(dr)
```

Record data for 60 seconds on the SoC hardware board.

```
record(dr, 60);
```

Check the status of the data recording session by using the `isRecording` object function. The recording status when data recording is in progress is `1`.

```
recordingStatus = isRecording(dr)
```

```
recordingStatus =

  logical

   1
```

The recording status when data recording is complete is `0`.

```
isRecording(dr)
```

```
recordingStatus =

  logical

   0
```

Save recorded data to a TGZ-compressed file.

```
save(dr,'UDPDataReceived','UDP Data Testing',{'Recorded On Zynq Board'})
```

This function saves the recorded data as the file `UDPDataReceived.tgz` in your working folder of the host PC. You can read this file by using an `socFileReader` object in MATLAB or an IO Data Source block in your Simulink model.

Remove the added source from the data recording session by using the `removeSource` object function.

```
removeSource(dr,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
ans =

  1×0 empty cell array
```

Input Arguments

`hw` — Hardware object

`socHardwareBoard` object

Hardware object, specified as a `socHardwareBoard` object that represents the connection to the SoC hardware board.

Output Arguments

`dr` — Data recording session specified for SoC hardware board

`soc.recorder` object

Data recording session for specified SoC hardware board, specified as a `DataRecorder` object.

Version History

Introduced in R2019a

See Also

`DataRecorder`

Topics

“Record I/O Data from SoC Device”

setup

Set up hardware for data recording

Syntax

```
setup(dr)
```

Description

`setup(dr)` sets up any input sources on the SoC hardware board represented by `dr` to record data. `dr` is a data recording session on SoC hardware board created using `DataRecorder`. You must have added at least one source to `dr`, using the `addSource` function.

Examples

Record Data From SoC Hardware Board

Create a connection from MATLAB to the specified SoC hardware board using the IP address, username, and password of the board.

```
hw = socHardwareBoard('Xilinx Zynq ZC706 evaluation kit','hostname','192.168.1.18','username','ro
```

Create a data recording session on the SoC hardware board by using the `hw` object. The resulting `DataRecorder` object represents the data recording session on the SoC hardware board.

```
dr = soc.recorder(hw)
```

```
dr =

  DataRecorder with properties:

         HardwareName: 'Xilinx Zynq ZC706 evaluation kit'
              Sources: {}
            Recording: false
```

List the input sources added to the data recording session.

```
dr.Sources(hw)
```

```
ans =

  1×0 empty cell array
```

By default, `soc.recorder` objects have no added input sources. To add an input source to the data recording session, first create an input source object by using the `soc.iosource` function. For this example, create an User Datagram Protocol (UDP) source object.

```
udpSrc = soc.iosource(hw,'UDP Receive')
```

```
udpSrc =
```

```
  soc.iosource.UDPRead with properties:

   Main
              LocalPort: 25000
             DataLength: 1
               DataType: 'uint8'
      ReceiveBufferSize: -1
           BlockingTime: 0
    OutputVarSizeSignal: false
             SampleTime: 0.1000
         HideEventLines: true

  Show all properties
```

Add this UDP source object to the data recording session by using the `addSource` object function.

```
addSource(dr,udpSrc,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
dr.Sources
```

```
ans =

  1×1 cell array

    {'UDPDataOnPort25000'}
```

Call the setup function to initialize all hardware peripheral input sources added to the data recording session, and start the data recording process.

```
setup(dr)
```

Record data for 60 seconds on the SoC hardware board.

```
record(dr, 60);
```

Check the status of the data recording session by using the `isRecording` object function. The recording status when data recording is in progress is `1`.

```
recordingStatus = isRecording(dr)
```

```
recordingStatus =

  logical

   1
```

The recording status when data recording is complete is `0`.

```
isRecording(dr)
```

```
recordingStatus =

  logical

   0
```

Save recorded data to a TGZ-compressed file.

```
save(dr,'UDPDataReceived','UDP Data Testing',{'Recorded On Zynq Board'})
```

This function saves the recorded data as the file `UDPDataReceived.tgz` in your working folder of the host PC. You can read this file by using an `socFileReader` object in MATLAB or an IO Data Source block in your Simulink model.

Remove the added source from the data recording session by using the `removeSource` object function.

```
removeSource(dr,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
ans =

  1×0 empty cell array
```

Input Arguments

`dr` — Data recording session specified for SoC hardware board

`soc.recorder` object

Data recording session for specified SoC hardware board, specified as a `DataRecorder` object.

Version History

Introduced in R2019a

See Also

`DataRecorder` | `record` | `addSource` | `removeSource`

addSource

Add a input source to a data recording session

Syntax

`addSource(dr,src,sourceName)`

Description

`addSource(dr,src,sourceName)` adds the specified hardware input source, `src` to data recording session, `dr`. `dr` is a data recording session on SoC hardware board created using `DataRecorder`.

Examples

Record Data From SoC Hardware Board

Create a connection from MATLAB to the specified SoC hardware board using the IP address, username, and password of the board.

```
hw = socHardwareBoard('Xilinx Zynq ZC706 evaluation kit','hostname','192.168.1.18','username','r
```

Create a data recording session on the SoC hardware board by using the `hw` object. The resulting `DataRecorder` object represents the data recording session on the SoC hardware board.

```
dr = soc.recorder(hw)
```

```
dr =

  DataRecorder with properties:

         HardwareName: 'Xilinx Zynq ZC706 evaluation kit'
              Sources: {}
            Recording: false
```

List the input sources added to the data recording session.

```
dr.Sources(hw)
```

```
ans =

  1×0 empty cell array
```

By default, `soc.recorder` objects have no added input sources. To add an input source to the data recording session, first create an input source object by using the `soc.iosource` function. For this example, create an User Datagram Protocol (UDP) source object.

```
udpSrc = soc.iosource(hw,'UDP Receive')
```

```
udpSrc =
```

```
  soc.iosource.UDPRead with properties:

   Main
              LocalPort: 25000
             DataLength: 1
               DataType: 'uint8'
      ReceiveBufferSize: -1
           BlockingTime: 0
    OutputVarSizeSignal: false
             SampleTime: 0.1000
         HideEventLines: true

  Show all properties
```

Add this UDP source object to the data recording session by using the `addSource` object function.

```
addSource(dr,udpSrc,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
dr.Sources
```

```
ans =

  1×1 cell array

    {'UDPDataOnPort25000'}
```

Call the setup function to initialize all hardware peripheral input sources added to the data recording session, and start the data recording process.

```
setup(dr)
```

Record data for 60 seconds on the SoC hardware board.

```
record(dr, 60);
```

Check the status of the data recording session by using the `isRecording` object function. The recording status when data recording is in progress is `1`.

```
recordingStatus = isRecording(dr)
```

```
recordingStatus =

  logical

   1
```

The recording status when data recording is complete is `0`.

```
isRecording(dr)
```

```
recordingStatus =

  logical

   0
```

Save recorded data to a TGZ-compressed file.

```
save(dr,'UDPDataReceived','UDP Data Testing',{'Recorded On Zynq Board'})
```

This function saves the recorded data as the file `UDPDataReceived.tgz` in your working folder of the host PC. You can read this file by using an `socFileReader` object in MATLAB or an IO Data Source block in your Simulink model.

Remove the added source from the data recording session by using the `removeSource` object function.

```
removeSource(dr,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
ans =

  1×0 empty cell array
```

Input Arguments

`dr` — Data recording session specified for SoC hardware board

`soc.recorder` object

Data recording session for specified SoC hardware board, specified as a `DataRecorder` object.

`src` — Source object for specified input source

`soc.iosource` object

Source object for specified input source, specified as an `soc.iosource` object.

`sourceName` — Name of input source in the data recording session

character vector

Name of input source in the data recording session, specified as a character vector. The function uses this name as the source name when the specified input source is recorded and saved on a dataset file.

Note Setting `sourceName` to `'all'` errors as the `sourceName` `'all'` is used to remove all input sources added to a data recording session using the `removeSource` function.

Version History

Introduced in R2019a

See Also

`removeSource` | `DataRecorder` | `soc.iosource`

removeSource

Remove input source from data recording session

Syntax

```
removeSource(dr,sourceName)
```

Description

`removeSource(dr,sourceName)` removes an already added input source from a data recording session, `dr`.

Examples

Record Data From SoC Hardware Board

Create a connection from MATLAB to the specified SoC hardware board using the IP address, username, and password of the board.

```
hw = socHardwareBoard('Xilinx Zynq ZC706 evaluation kit','hostname','192.168.1.18','username','r
```

Create a data recording session on the SoC hardware board by using the `hw` object. The resulting `DataRecorder` object represents the data recording session on the SoC hardware board.

```
dr = soc.recorder(hw)
```

```
dr =

  DataRecorder with properties:

         HardwareName: 'Xilinx Zynq ZC706 evaluation kit'
              Sources: {}
            Recording: false
```

List the input sources added to the data recording session.

```
dr.Sources(hw)
```

```
ans =

  1×0 empty cell array
```

By default, `soc.recorder` objects have no added input sources. To add an input source to the data recording session, first create an input source object by using the `soc.iosource` function. For this example, create an User Datagram Protocol (UDP) source object.

```
udpSrc = soc.iosource(hw,'UDP Receive')
```

```
udpSrc =
```

```
  soc.iosource.UDPRead with properties:

   Main
              LocalPort: 25000
             DataLength: 1
               DataType: 'uint8'
      ReceiveBufferSize: -1
           BlockingTime: 0
    OutputVarSizeSignal: false
             SampleTime: 0.1000
         HideEventLines: true

  Show all properties
```

Add this UDP source object to the data recording session by using the `addSource` object function.

```
addSource(dr,udpSrc,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
dr.Sources
```

```
ans =

  1×1 cell array

    {'UDPDataOnPort25000'}
```

Call the setup function to initialize all hardware peripheral input sources added to the data recording session, and start the data recording process.

```
setup(dr)
```

Record data for 60 seconds on the SoC hardware board.

```
record(dr, 60);
```

Check the status of the data recording session by using the `isRecording` object function. The recording status when data recording is in progress is `1`.

```
recordingStatus = isRecording(dr)
```

```
recordingStatus =

  logical

   1
```

The recording status when data recording is complete is `0`.

```
isRecording(dr)
```

```
recordingStatus =

  logical

   0
```

Save recorded data to a TGZ-compressed file.

```
save(dr,'UDPDataReceived','UDP Data Testing',{'Recorded On Zynq Board'})
```

This function saves the recorded data as the file `UDPDataReceived.tgz` in your working folder of the host PC. You can read this file by using an `socFileReader` object in MATLAB or an IO Data Source block in your Simulink model.

Remove the added source from the data recording session by using the `removeSource` object function.

```
removeSource(dr,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
ans =

  1×0 empty cell array
```

Input Arguments

`dr` — Data recording session specified for SoC hardware board

`soc.recorder` object

Data recording session for specified SoC hardware board, specified as a `DataRecorder` object.

`sourceName` — Name of specified input source in data recording session

character vector

Name of specified input source in the data recording session, specified as a character vector.

Note You can specify `sourceName` as `'all'` to remove all input sources added to a data recording session.

Version History

Introduced in R2019a

See Also

`soc.iosource` | `DataRecorder`

record

Record data from hardware using data recorder object

Syntax

```
record(dr,duration)
```

Description

`record(dr,duration)` records hardware input data on the SoC hardware board represented by `dr`, for the specified duration. `dr` is a data recording session on SoC hardware board created using `DataRecorder`.

Examples

Record Data From SoC Hardware Board

Create a connection from MATLAB to the specified SoC hardware board using the IP address, username, and password of the board.

```
hw = socHardwareBoard('Xilinx Zynq ZC706 evaluation kit','hostname','192.168.1.18','username','ro
```

Create a data recording session on the SoC hardware board by using the `hw` object. The resulting `DataRecorder` object represents the data recording session on the SoC hardware board.

```
dr = soc.recorder(hw)
```

```
dr =

  DataRecorder with properties:

         HardwareName: 'Xilinx Zynq ZC706 evaluation kit'
              Sources: {}
            Recording: false
```

List the input sources added to the data recording session.

```
dr.Sources(hw)
```

```
ans =

  1×0 empty cell array
```

By default, `soc.recorder` objects have no added input sources. To add an input source to the data recording session, first create an input source object by using the `soc.iosource` function. For this example, create an User Datagram Protocol (UDP) source object.

```
udpSrc = soc.iosource(hw,'UDP Receive')
```

```
udpSrc =
```

```
  soc.iosource.UDPRead with properties:

   Main
              LocalPort: 25000
             DataLength: 1
               DataType: 'uint8'
      ReceiveBufferSize: -1
           BlockingTime: 0
    OutputVarSizeSignal: false
             SampleTime: 0.1000
         HideEventLines: true

  Show all properties
```

Add this UDP source object to the data recording session by using the `addSource` object function.

```
addSource(dr,udpSrc,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
dr.Sources
```

```
ans =

  1×1 cell array

    {'UDPDataOnPort25000'}
```

Call the setup function to initialize all hardware peripheral input sources added to the data recording session, and start the data recording process.

```
setup(dr)
```

Record data for 60 seconds on the SoC hardware board.

```
record(dr, 60);
```

Check the status of the data recording session by using the `isRecording` object function. The recording status when data recording is in progress is `1`.

```
recordingStatus = isRecording(dr)
```

```
recordingStatus =

  logical

   1
```

The recording status when data recording is complete is `0`.

```
isRecording(dr)
```

```
recordingStatus =

  logical

   0
```

Save recorded data to a TGZ-compressed file.

```
save(dr,'UDPDataReceived','UDP Data Testing',{'Recorded On Zynq Board'})
```

This function saves the recorded data as the file `UDPDataReceived.tgz` in your working folder of the host PC. You can read this file by using an `socFileReader` object in MATLAB or an IO Data Source block in your Simulink model.

Remove the added source from the data recording session by using the `removeSource` object function.

```
removeSource(dr,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
ans =

  1×0 empty cell array
```

Input Arguments

`dr` — Data recording session specified for SoC hardware board

`soc.recorder` object

Data recording session for specified SoC hardware board, specified as a `DataRecorder` object.

`duration` — Duration of recording session

positive scalar

Duration of recording session, specified as a positive scalar in seconds. Data is recorded on the hardware board for the specified duration of time. You can check the status of the data recording session by calling the `isRecording` object function.

Data Types: `double`

Version History

Introduced in R2019a

See Also

`DataRecorder` | `isRecording` | `setup`

isRecording

Get data recording status

Syntax

```
recordingStatus = isRecording(dr)
```

Description

`recordingStatus = isRecording(dr)` returns the status of the data recording process on the SoC hardware board represented by `dr`. `dr` is a data recording session on SoC hardware board created using `DataRecorder`.

Examples

Record Data From SoC Hardware Board

Create a connection from MATLAB to the specified SoC hardware board using the IP address, username, and password of the board.

```
hw = socHardwareBoard('Xilinx Zynq ZC706 evaluation kit','hostname','192.168.1.18','username','ro
```

Create a data recording session on the SoC hardware board by using the `hw` object. The resulting `DataRecorder` object represents the data recording session on the SoC hardware board.

```
dr = soc.recorder(hw)
```

```
dr =

  DataRecorder with properties:

         HardwareName: 'Xilinx Zynq ZC706 evaluation kit'
              Sources: {}
            Recording: false
```

List the input sources added to the data recording session.

```
dr.Sources(hw)
```

```
ans =

  1×0 empty cell array
```

By default, `soc.recorder` objects have no added input sources. To add an input source to the data recording session, first create an input source object by using the `soc.iosource` function. For this example, create an User Datagram Protocol (UDP) source object.

```
udpSrc = soc.iosource(hw,'UDP Receive')
```

```
udpSrc =
```

```
  soc.iosource.UDPRead with properties:

   Main
              LocalPort: 25000
             DataLength: 1
               DataType: 'uint8'
      ReceiveBufferSize: -1
           BlockingTime: 0
    OutputVarSizeSignal: false
             SampleTime: 0.1000
         HideEventLines: true

  Show all properties
```

Add this UDP source object to the data recording session by using the `addSource` object function.

```
addSource(dr,udpSrc,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
dr.Sources
```

```
ans =

  1×1 cell array

    {'UDPDataOnPort25000'}
```

Call the setup function to initialize all hardware peripheral input sources added to the data recording session, and start the data recording process.

```
setup(dr)
```

Record data for 60 seconds on the SoC hardware board.

```
record(dr, 60);
```

Check the status of the data recording session by using the `isRecording` object function. The recording status when data recording is in progress is `1`.

```
recordingStatus = isRecording(dr)
```

```
recordingStatus =

  logical

   1
```

The recording status when data recording is complete is `0`.

```
isRecording(dr)
```

```
recordingStatus =

  logical

   0
```

Save recorded data to a TGZ-compressed file.

```
save(dr,'UDPDataReceived','UDP Data Testing',{'Recorded On Zynq Board'})
```

This function saves the recorded data as the file `UDPDataReceived.tgz` in your working folder of the host PC. You can read this file by using an `socFileReader` object in MATLAB or an IO Data Source block in your Simulink model.

Remove the added source from the data recording session by using the `removeSource` object function.

```
removeSource(dr,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
ans =

  1×0 empty cell array
```

Input Arguments

`dr` — Data recording session specified for SoC hardware board

`soc.recorder` object

Data recording session for specified SoC hardware board, specified as a `DataRecorder` object.

Output Arguments

`recordingStatus` — Status of data recording session

`false` (`0`) | `true` (`1`)

Status of data recording session, returned as logical value of `false` (`0`) or `true` (`1`). This value is `1` when data recording is in progress and `0` when data recording is complete.

Version History

Introduced in R2019a

See Also

`DataRecorder` | `record`

save

Save recorded data from SoC hardware board to file on host PC

Syntax

`save(dr,filename,description,tags)`

Description

`save(dr,filename,description,tags)` saves the recorded data from the SoC hardware board associated with `DataRecorder` object `dr` to a TGZ-compressed file, `filename`, on the host PC.

Examples

Record Data From SoC Hardware Board

Create a connection from MATLAB to the specified SoC hardware board using the IP address, username, and password of the board.

```
hw = socHardwareBoard('Xilinx Zynq ZC706 evaluation kit','hostname','192.168.1.18','username','r
```

Create a data recording session on the SoC hardware board by using the `hw` object. The resulting `DataRecorder` object represents the data recording session on the SoC hardware board.

```
dr = soc.recorder(hw)
```

```
dr =

  DataRecorder with properties:

         HardwareName: 'Xilinx Zynq ZC706 evaluation kit'
              Sources: {}
            Recording: false
```

List the input sources added to the data recording session.

```
dr.Sources(hw)
```

```
ans =

  1×0 empty cell array
```

By default, `soc.recorder` objects have no added input sources. To add an input source to the data recording session, first create an input source object by using the `soc.iosource` function. For this example, create an User Datagram Protocol (UDP) source object.

```
udpSrc = soc.iosource(hw,'UDP Receive')
```

```
udpSrc =
```

```
  soc.iosource.UDPRead with properties:

   Main
              LocalPort: 25000
             DataLength: 1
               DataType: 'uint8'
      ReceiveBufferSize: -1
           BlockingTime: 0
    OutputVarSizeSignal: false
             SampleTime: 0.1000
         HideEventLines: true

  Show all properties
```

Add this UDP source object to the data recording session by using the `addSource` object function.

```
addSource(dr,udpSrc,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
dr.Sources
```

```
ans =

  1×1 cell array

    {'UDPDataOnPort25000'}
```

Call the setup function to initialize all hardware peripheral input sources added to the data recording session, and start the data recording process.

```
setup(dr)
```

Record data for 60 seconds on the SoC hardware board.

```
record(dr, 60);
```

Check the status of the data recording session by using the `isRecording` object function. The recording status when data recording is in progress is `1`.

```
recordingStatus = isRecording(dr)
```

```
recordingStatus =

  logical

   1
```

The recording status when data recording is complete is `0`.

```
isRecording(dr)
```

```
recordingStatus =

  logical

   0
```

Save recorded data to a TGZ-compressed file.

```
save(dr,'UDPDataReceived','UDP Data Testing',{'Recorded On Zynq Board'})
```

This function saves the recorded data as the file `UDPDataReceived.tgz` in your working folder of the host PC. You can read this file by using an `socFileReader` object in MATLAB or an IO Data Source block in your Simulink model.

Remove the added source from the data recording session by using the `removeSource` object function.

```
removeSource(dr,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
ans =

  1×0 empty cell array
```

Input Arguments

`dr` — Data recording session specified for SoC hardware board

`soc.recorder` object

Data recording session for specified SoC hardware board, specified as a `DataRecorder` object.

`filename` — File name

character vector

File name by which you want to save the recorded data from the SoC hardware board on your host PC, specified as a character vector.

`description` — Description added to file

character vector

Description added to file, specified as character vector. Add a description to the file helps you identify the data file when you read it using an `socFileReader` object. This input is optional.

Data Types: `char`

`tags` — Tags for data set

`cell array`

Tags for the data set, specified as cell array of character vectors. Adding tags to the file helps you identify the different input sources when you read the file using an `socFileReader` object. This input is optional.

Version History

Introduced in R2019a

See Also

`DataRecorder` | `socFileReader`

socTaskTimes

Plot histogram of the task durations from a recorded Simulation Data Inspector run

Syntax

```
taskData = socTaskTimes(modelName,runName)
taskData = socTaskTimes( ___ ,suppressPlot)
```

Description

`taskData = socTaskTimes(modelName,runName)` creates an array of structures, one element for each task. Each structure contains the task name, task start times, task durations, and mean and standard deviations of the task durations. The function also plots the histogram of task durations for each task.

`taskData = socTaskTimes( ___ ,suppressPlot)` to suppress the plot generated.

Input Arguments

`modelName` — Name of the Simulink model

string (default) | character array

Name of the Simulink model associated with run containing tasks.

Data Types: `char` | `string`

`runName` — Name of the Simulation Data Inspector run

string (default) | character array

Name of Simulation Data Inspector run containing a task.

Data Types: `char` | `string`

`suppressPlot` — Name of the Simulink model

`"SuppressPlot"` (default)

Suppress the automatic generation of task duration plots.

Data Types: `char` | `string`

Output Arguments

`taskData` — Task timing data and statistics

structure

Task timing and duration statistics, returned as a structure with the fields:

Version History

Introduced in R2019a

See Also

“Task Visualization in Simulation Data Inspector” | “Recording Tasks for Use in Simulation”

socHardwareUsage

Generate processor core usage statistics from a recorded Simulation Data Inspector run

Syntax

```
coreData = socHardwareUsage(modelname,runname)
coreData = socHardwareUsage(__,suppressPlot)
```

Description

`coreData = socHardwareUsage(modelname,runname)` generates task timing data and duration statistics, including the core name, the average core usage (%) and the average usage, for each task. The function also can plot the average usage of each core.

`coreData = socHardwareUsage(__,suppressPlot)` suppress the automatic generation of usage plots.

Input Arguments

`modelname` — Name of the Simulink model

string scalar | character array

Name of a Simulink model that is associated with a Simulation Data Inspector run, specified as a string scalar or character vector.

Data Types: `char` | `string`

`runname` — Name of the Simulation Data Inspector run

string scalar | character array

Name of a Simulation Data Inspector run containing a task, specified as a string scalar or character vector.

Data Types: `char` | `string`

`suppressPlot` — Suppress generation of usage plots

`"SuppressPlot"` (default)

Suppress the automatic generation of hardware usage plots.

Data Types: `char` | `string`

Output Arguments

`coreData` — Processor core usage statistics

structure

Task timing data and duration statistics, returned as a structure.

Version History

Introduced in R2021b

See Also

“Task Visualization in Simulation Data Inspector” | “Recording Tasks for Use in Simulation”

soclib

Open the SoC Blockset block library

Syntax

```
soclib
```

Description

`soclib` opens the SoC Blockset block library.

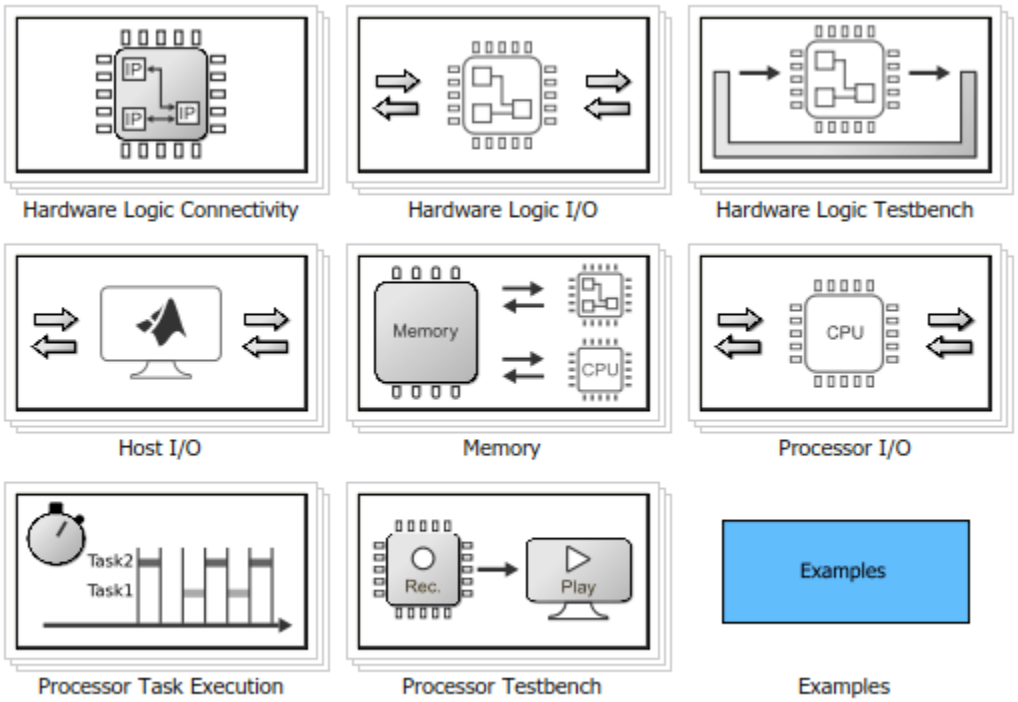
Examples

View the SoC Blockset Library

This example shows how to open and view the SoC Blockset library.

Run the following command to open the SoC Blockset library in Simulink:

```
soclib
```



Version History

Introduced in R2019a

See Also

“Get Started with SoC Blockset”

collectMemoryStatistics

Retrieve performance data from AXI interconnect monitor

Syntax

```
collectMemoryStatistics(profiler)
```

Description

`collectMemoryStatistics(profiler)` retrieves performance data from the AXI interconnect monitor IP running on your hardware board. The `profiler` object represents a connection to that IP. When the AXI interconnect monitor is configured in `'Profile'` mode, call this function in a loop to retrieve average transaction latency and counts of bursts and bytes while transactions are occurring. In `'Trace'` mode, call this function once after memory transactions are complete to retrieve detailed memory transaction event data.

Examples

Configure and Query AXI Interconnect Monitor

The AXI interconnect monitor (AIM) is an IP core that collects performance metrics for an AXI-based FPGA design. Create an `socIPCore` object to setup and configure the AIM IP, and use the `socMemoryProfiler` object to retrieve and display the data.

For an example of how to configure and query the AIM IP in your design using MATLAB as AXI Master, see “Analyze Memory Bandwidth Using Traffic Generators”. Specifically, review the `soc_memory_traffic_generator_axi_master.m` script that configures and monitors the design on the device.

The performance monitor can collect two types of data. Choose *Profile* mode to collect average transaction latency and counts of bytes and bursts. In this mode, you can launch a performance plot tool, and then configure the tool to plot bandwidth, burst count, and transaction latency. Choose *Trace* mode to collect detailed memory transaction event data and view the data as waveforms.

```
Mode = 'Profile'; % or 'Trace'
```

To obtain diagnostic performance metrics from your generated FPGA design, you must set up a JTAG connection to the device from MATLAB. Load a `.mat` file that contains structures derived from the board configuration parameters. This file was generated by the **SoC Builder** tool. These structures describe the memory interconnect and masters configuration such as buffer sizes and addresses. Use the `socHardwareBoard` object to set up the JTAG connection.

```
load('soc_memory_traffic_generator_zc706_aximaster.mat');
hwObj = socHardwareBoard('Xilinx Zynq ZC706 evaluation kit','Connect',false);
AXIMasterObj = socAXIMaster(hwObj);
```

Configure the AIM. The `socIPCore` object provides a function that performs this initialization. Then, create an `socMemoryProfiler` object to gather the metrics.

```
apmCoreObj = socIPCore(AXIMasterObj,perf_mon,'PerformanceMonitor','Mode',Mode);
initialize(apmCoreObj);
profilerObj = socMemoryProfiler(hwObj,apmCoreObj);
```

Retrieve performance metrics or signal data from a design running on the FPGA by using the `socMemoryProfiler` object functions.

For `'Profile'` mode, call the `collectMemoryStatistics` function in a loop.

```
NumRuns = 100;
for n = 1:NumRuns
    collectMemoryStatistics(profilerObj);
end
```

JTAG design setup time is long relative to FPGA transaction times, and if you have a small number of transactions in your design, they might have already completed by the time you query the monitor. In this case, the bandwidth plot shows only one sample, and the throughput calculation is not accurate. If this situation occurs, increase the total number of transactions the design executes.

For `'Trace'` mode, call the `collectMemoryStatistics` function once. This function stops the IP from writing transactions into the FIFO in the AXI interconnect monitor IP, although the transactions continue on the interconnect. Set the size of the transaction FIFO, **Trace capture depth**, in the configuration parameters of the model, under **Hardware Implementation** > **Target hardware resources** > **FPGA design (debug)**.

```
collectMemoryStatistics(profilerObj);
```

Visualize the performance data by using the `plotMemoryStatistics` function. In `'Profile'` mode, this function launches a performance plot tool, and you can configure the tool to plot bandwidth, burst count, and average transaction latency. In `'Trace'` mode, this function opens the **Logic Analyzer** tool to view burst transaction event data.

```
plotMemoryStatistics(profilerObj);
```

Input Arguments

`profiler` — Memory profiler object

`socMemoryProfiler` object

Memory profiler object, specified as an `socMemoryProfiler` object that provides access to the AXI memory interconnect IP running on the hardware board.

Version History

Introduced in R2019a

See Also

“Memory Performance Information from FPGA Execution”

Topics

“Analyze Memory Bandwidth Using Traffic Generators”

plotMemoryStatistics

Plot performance data obtained from AXI interconnect monitor

Syntax

```
plotMemoryStatistics(profiler)
```

Description

`plotMemoryStatistics(profiler)` generates visualizations of the performance data from the AXI interconnect monitor IP running on your hardware board. The `profiler` object represents a connection to that IP. When the AXI interconnect monitor is configured in `'Profile'` mode, this function launches a performance plot tool. You can configure the tool to plot bandwidth, burst count, and average transaction latency. In `'Trace'` mode, this function opens the **Logic Analyzer** to view detailed memory transaction event data.

Examples

Configure and Query AXI Interconnect Monitor

The AXI interconnect monitor (AIM) is an IP core that collects performance metrics for an AXI-based FPGA design. Create an `socIPCore` object to setup and configure the AIM IP, and use the `socMemoryProfiler` object to retrieve and display the data.

For an example of how to configure and query the AIM IP in your design using MATLAB as AXI Master, see “Analyze Memory Bandwidth Using Traffic Generators”. Specifically, review the `soc_memory_traffic_generator_axi_master.m` script that configures and monitors the design on the device.

The performance monitor can collect two types of data. Choose *Profile* mode to collect average transaction latency and counts of bytes and bursts. In this mode, you can launch a performance plot tool, and then configure the tool to plot bandwidth, burst count, and transaction latency. Choose *Trace* mode to collect detailed memory transaction event data and view the data as waveforms.

```
Mode = 'Profile'; % or 'Trace'
```

To obtain diagnostic performance metrics from your generated FPGA design, you must set up a JTAG connection to the device from MATLAB. Load a `.mat` file that contains structures derived from the board configuration parameters. This file was generated by the **SoC Builder** tool. These structures describe the memory interconnect and masters configuration such as buffer sizes and addresses. Use the `socHardwareBoard` object to set up the JTAG connection.

```
load('soc_memory_traffic_generator_zc706_aximaster.mat');
hwObj = socHardwareBoard('Xilinx Zynq ZC706 evaluation kit','Connect',false);
AXIMasterObj = socAXIMaster(hwObj);
```

Configure the AIM. The `socIPCore` object provides a function that performs this initialization. Then, create an `socMemoryProfiler` object to gather the metrics.

```
apmCoreObj = socIPCore(AXIMasterObj,perf_mon,'PerformanceMonitor','Mode',Mode);
initialize(apmCoreObj);
profilerObj = socMemoryProfiler(hwObj,apmCoreObj);
```

Retrieve performance metrics or signal data from a design running on the FPGA by using the `socMemoryProfiler` object functions.

For `'Profile'` mode, call the `collectMemoryStatistics` function in a loop.

```
NumRuns = 100;
for n = 1:NumRuns
    collectMemoryStatistics(profilerObj);
end
```

JTAG design setup time is long relative to FPGA transaction times, and if you have a small number of transactions in your design, they might have already completed by the time you query the monitor. In this case, the bandwidth plot shows only one sample, and the throughput calculation is not accurate. If this situation occurs, increase the total number of transactions the design executes.

For `'Trace'` mode, call the `collectMemoryStatistics` function once. This function stops the IP from writing transactions into the FIFO in the AXI interconnect monitor IP, although the transactions continue on the interconnect. Set the size of the transaction FIFO, **Trace capture depth**, in the configuration parameters of the model, under **Hardware Implementation** > **Target hardware resources** > **FPGA design (debug)**.

```
collectMemoryStatistics(profilerObj);
```

Visualize the performance data by using the `plotMemoryStatistics` function. In `'Profile'` mode, this function launches a performance plot tool, and you can configure the tool to plot bandwidth, burst count, and average transaction latency. In `'Trace'` mode, this function opens the **Logic Analyzer** tool to view burst transaction event data.

```
plotMemoryStatistics(profilerObj);
```

Input Arguments

`profiler` — Memory profiler object

`socMemoryProfiler` object

Memory profiler object, specified as an `socMemoryProfiler` object that provides access to the AXI memory interconnect IP running on the hardware board.

Version History

Introduced in R2019a

See Also

“Memory Performance Information from FPGA Execution”

Topics

“Analyze Memory Bandwidth Using Traffic Generators”

initialize

Initialize IP core corresponding to `socIPCore` object

Syntax

```
initialize(socIP)
```

Description

`initialize(socIP)` initializes the IP core corresponding to `socIP`, an `socIPCore` object.

Examples

Initialize Traffic Generator IP

Create an `socIPCore` object representing a traffic generator IP on an FPGA board. Then initialize it using the `initialize` function.

```
% Create IPCore object for traffic generator IP
trafficGeneratorObj = socIPCore(AXIMasterObj, atg,'TrafficGenerator');
% Initialize traffic generator IP
initialize(trafficGeneratorObj);
```

Input Arguments

socIP — Connection to IP core running on FPGA board

`socIPCore`

Connection to IP core running on FPGA board, specified as an `socIPCore` object.

Version History

Introduced in R2019a

See Also

`socIPCore`

start

Start IP core execution on hardware board

Syntax

```
start(socIP)
```

Description

`start(socIP)` starts the execution of the IP core represented by the `socIP` object.

This function is only applicable when `socIPCore` is an object representing `TrafficGenerator` or `VDMATrigger`.

Examples

Initialize and Start a Traffic Generator IP

Create an `socIPCore` object representing a traffic generator IP on an FPGA board. Then initialize the traffic generator using the `initialize` function.

```
% Create IPCore object for traffic generator IP
trafficGeneratorObj = socIPCore(AXIMasterObj, atg,'TrafficGenerator');
% Initialize traffic generator IP
initialize(trafficGeneratorObj);
```

Start the traffic generator IP execution on your FPGA board.

```
start(trafficGeneratorObj);
```

Input Arguments

socIP — Connection to IP core running on FPGA board

`socIPCore`

Connection to IP core running on FPGA board, specified as an `socIPCore` object.

Version History

Introduced in R2019a

See Also

`socIPCore`

readmemory

Read data from AXI4 memory-mapped locations

Syntax

```
data = readmemory(mem,addr,size)
data = readmemory(mem,addr,size,Name,Value)
```

Description

`data = readmemory(mem,addr,size)` reads `size` locations of data, starting from the address specified in `addr`, and incrementing the address for each word. By default, the output data type is `uint32`. `addr`, must refer to an AXI slave memory location controlled by the AXI manager IP on your hardware board. The `socAXIManager` object, `mem`, manages the connection between MATLAB and the AXI manager IP.

`data = readmemory(mem,addr,size,Name,Value)` reads `size` locations of data, starting from the address specified in `addr`, with additional options specified by one or more `Name,Value` pair arguments.

Examples

Access Memory on SoC Hardware Board from MATLAB

For this example, you must have a design running on a hardware board connected to the MATLAB host machine.

Create a MATLAB AXI manager object. The object connects with the hardware board and confirms that the IP is present. You can create the object with a vendor name or an `socHardwareBoard` object.

```
mem = socAXIManager('Xilinx');
```

Write and read one or more addresses with one command. By default, the functions auto-increment the address for each word of data. For instance, write ten addresses, then read the data back from a single location.

```
writememory(mem,140,[10:19])
rd_d = readmemory(mem,140,1)
```

```
rd_d =

  uint32

   10
```

Now, read the written data from ten locations.

```
rd_d = readmemory(mem,140,10)
```

```
rd_d =

  1×10 uint32 row vector

   10   11   12   13   14   15   16   17   18   19
```

Set the `BurstType` property to `'Fixed'` to turn off the auto-increment and access the same address multiple times. For instance, read the written data ten times from the same address.

```
rd_d = readmemory(mem,140,10,'BurstType','Fixed')
```

```
rd_d =

  1×10 uint32 row vector

   10   10   10   10   10   10   10   10   10   10
```

Write incrementing data ten times to the same address. The final value stored in address `140` is `29`.

```
writememory(mem,140,[20:29],'BurstType','Fixed')
rd_d = readmemory(mem,140,10)
```

```
rd_d =

  1×10 uint32 row vector

   29   11   12   13   14   15   16   17   18   19
```

Alternatively, specify the address as a hexadecimal string. To cast the read data to a data type other than `uint32`, use the `OutputDataType` property.

```
writememory(mem,'1c',[0:4:64])
rd_d = readmemory(mem,'1c',16,'OutputDataType',numerictype(0,6,4))
```

```
rd_d =

  Columns 1 through 10
         0    0.2500    0.5000    0.7500    1.0000    1.2500    1.5000    1.7500    2.0000    2.2
  Columns 11 through 16
    2.5000    2.7500    3.0000    3.2500    3.5000    3.7500

          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Unsigned
            WordLength: 6
        FractionLength: 4
```

When you are done accessing the board, release the JTAG connection.

```
release(mem)
```

Input Arguments

`mem` — JTAG connection to AXI manager IP running on hardware board

`socAXIManager` object

JTAG connection to AXI manager IP running on your hardware board, specified as an `socAXIManager` object.

`addr` — Starting address for read operation

integer | hexadecimal character vector

Starting address for read operation, specified as an integer or a hexadecimal character vector. The function casts the address to `uint32` data type. The address must refer to an AXI slave memory location controlled by the AXI manager IP on your hardware board.

Example: `'a4'`

`size` — Number of locations to read

integer

Number of memory locations to read, specified as an integer. By default, the function reads from a contiguous address block, incrementing the address for each operation. To turn off the address increment and read repeatedly from the same location, set the `BurstType` property to `'Fixed'`.

When you specify a large operation size, such as reading a block of DDR memory, the object automatically breaks the operation into multiple bursts, using the maximum supported burst size. The maximum supported burst size is 256 words.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `readmemory(mem,140,10,'BurstType','Fixed')`

`BurstType` — AXI4 burst type

`'Increment'` (default) | `'Fixed'`

AXI4 burst type, specified as the comma-separated pair consisting of `'BurstType'` and either `'Increment'` or `'Fixed'`. If this value is `'Increment'`, the AXI manager reads a vector of data from contiguous memory locations, starting with the specified address. If this value is `'Fixed'`, the AXI manager reads all data from the same address.

`OutputDataType` — Data type assigned to read data

`'uint32'` (default) | `'int8'` | `'int16'` | `'int32'` | `'uint8'` | `'uint16'` | `'single'` | `numerictype` object

Data type assigned to the read data, specified as `'uint32'`, `'int8'`, `'int16'`, `'int32'`, `'uint8'`, `'uint16'`, `'single'`, or a `numerictype` object.

Output Arguments

`data` — Read data

scalar | vector

Read data, returned as scalar or vector depending on the value you specified for `size`. The function casts the data to the data type specified by the `OutputDataType` property.

Version History

Introduced in R2019a

See Also

`writememory`

writememory

Write data to AXI4 memory-mapped locations

Syntax

```
writememory(mem,addr,data)
writememory(mem,addr,data,Name,Value)
```

Description

`writememory(mem,addr,data)` writes all words specified in `data`, starting from the address specified in `addr`, and then incrementing the address for each word. `addr`, must refer to an AXI slave memory location controlled by the AXI manager IP on your hardware board. The `socAXIManager` object, `mem`, manages the connection between MATLAB and the AXI manager IP.

`writememory(mem,addr,data,Name,Value)` writes all words specified in `data`, starting from the address specified in `addr`, with additional options specified by one or more `Name,Value` pair arguments.

Examples

Access Memory on SoC Hardware Board from MATLAB

For this example, you must have a design running on a hardware board connected to the MATLAB host machine.

Create a MATLAB AXI manager object. The object connects with the hardware board and confirms that the IP is present. You can create the object with a vendor name or an `socHardwareBoard` object.

```
mem = socAXIManager('Xilinx');
```

Write and read one or more addresses with one command. By default, the functions auto-increment the address for each word of data. For instance, write ten addresses, then read the data back from a single location.

```
writememory(mem,140,[10:19])
rd_d = readmemory(mem,140,1)
```

```
rd_d =

  uint32

   10
```

Now, read the written data from ten locations.

```
rd_d = readmemory(mem,140,10)
```

```
rd_d =
```

```
  1×10 uint32 row vector

   10   11   12   13   14   15   16   17   18   19
```

Set the `BurstType` property to `'Fixed'` to turn off the auto-increment and access the same address multiple times. For instance, read the written data ten times from the same address.

```
rd_d = readmemory(mem,140,10,'BurstType','Fixed')
```

```
rd_d =

  1×10 uint32 row vector

   10   10   10   10   10   10   10   10   10   10
```

Write incrementing data ten times to the same address. The final value stored in address `140` is `29`.

```
writememory(mem,140,[20:29],'BurstType','Fixed')
rd_d = readmemory(mem,140,10)
```

```
rd_d =

  1×10 uint32 row vector

   29   11   12   13   14   15   16   17   18   19
```

Alternatively, specify the address as a hexadecimal string. To cast the read data to a data type other than `uint32`, use the `OutputDataType` property.

```
writememory(mem,'1c',[0:4:64])
rd_d = readmemory(mem,'1c',16,'OutputDataType',numerictype(0,6,4))
```

```
rd_d =

  Columns 1 through 10
         0    0.2500    0.5000    0.7500    1.0000    1.2500    1.5000    1.7500    2.0000    2.2
  Columns 11 through 16
    2.5000    2.7500    3.0000    3.2500    3.5000    3.7500

          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Unsigned
            WordLength: 6
        FractionLength: 4
```

When you are done accessing the board, release the JTAG connection.

```
release(mem)
```

Input Arguments

`mem` — JTAG connection to AXI manager IP running on hardware board

`socAXIManager` object

JTAG connection to AXI manager IP running on your hardware board, specified as an `socAXIManager` object.

`addr` — Starting address for write operation

integer | hexadecimal character vector

Starting address for read operation, specified as an integer or a hexadecimal character vector. The function casts the address to `uint32` data type. The address must refer to an AXI slave memory location controlled by the AXI manager IP on your hardware board.

Example: `'a4'`

`data` — Data words to write
scalar | vector

Data words to write, specified as a scalar or a vector. By default, the function writes the data to a contiguous address block, incrementing the address for each operation. To turn off the address increment and write each data value to the same location, set the `BurstType` property to `'Fixed'` .

Before sending the write request to the board, the function casts the input data to `uint32` or `int32` data type. The data type conversion follows these rules:

- If the input data type is `double`, then the data is cast to `int32` data type.
- If the input data type is `single`, then the data is cast to `uint32` data type.
- If the bit width of the input data type is less than 32 bits, then the data is sign-extended to 32 bits.
- If the bit width of the input data type is longer than 32 bits, then the data is cast to `int32` or `uint32` data type, matching the signedness of the original data type.
- If the input data is a fixed-point data type, then the function writes the stored integer value of the data.

When you specify a large operation size, such as writing a block of DDR memory, the function automatically breaks the operation into multiple bursts, using the maximum supported burst size. The maximum supported burst size is 256 words.

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `writememory(mem,140,[20:29],'BurstType','Fixed')`

`BurstType` — AXI4 burst type
`'Increment'` (default) | `'Fixed'`

AXI4 burst type, specified as the comma-separated pair consisting of `'BurstType'` and either `'Increment'` or `'Fixed'`. If this value is `'Increment'`, the AXI manager writes a vector of data from contiguous memory locations, starting with the specified address. If this value is `'Fixed'`, the AXI manager writers all data from the same address.

Version History

Introduced in R2019a

See Also

`readmemory`

release

Release JTAG cable resource

Syntax

```
release(mem)
```

Description

`release(mem)` releases the JTAG cable resource, freeing the cable for use to reprogram the FPGA. After initialization, the AXI manager object, `mem`, holds the JTAG cable resource, and other programs cannot access that JTAG cable. When you have an active AXI manager object, FPGA programming over JTAG fails. Call the `release` object function before reprogramming the FPGA.

Examples

Access Memory on SoC Hardware Board from MATLAB

For this example, you must have a design running on a hardware board connected to the MATLAB host machine.

Create a MATLAB AXI manager object. The object connects with the hardware board and confirms that the IP is present. You can create the object with a vendor name or an `socHardwareBoard` object.

```
mem = socAXIManager('Xilinx');
```

Write and read one or more addresses with one command. By default, the functions auto-increment the address for each word of data. For instance, write ten addresses, then read the data back from a single location.

```
writememory(mem,140,[10:19])
rd_d = readmemory(mem,140,1)
```

```
rd_d =

  uint32

   10
```

Now, read the written data from ten locations.

```
rd_d = readmemory(mem,140,10)
```

```
rd_d =

  1×10 uint32 row vector

   10   11   12   13   14   15   16   17   18   19
```

Set the `BurstType` property to `'Fixed'` to turn off the auto-increment and access the same address multiple times. For instance, read the written data ten times from the same address.

```
rd_d = readmemory(mem,140,10,'BurstType','Fixed')
```

```
rd_d =

  1×10 uint32 row vector

   10   10   10   10   10   10   10   10   10   10
```

Write incrementing data ten times to the same address. The final value stored in address `140` is `29`.

```
writememory(mem,140,[20:29],'BurstType','Fixed')
rd_d = readmemory(mem,140,10)
```

```
rd_d =

  1×10 uint32 row vector

   29   11   12   13   14   15   16   17   18   19
```

Alternatively, specify the address as a hexadecimal string. To cast the read data to a data type other than `uint32`, use the `OutputDataType` property.

```
writememory(mem,'1c',[0:4:64])
rd_d = readmemory(mem,'1c',16,'OutputDataType',numerictype(0,6,4))
```

```
rd_d =

  Columns 1 through 10
         0    0.2500    0.5000    0.7500    1.0000    1.2500    1.5000    1.7500    2.0000    2.2
  Columns 11 through 16
    2.5000    2.7500    3.0000    3.2500    3.5000    3.7500

          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Unsigned
            WordLength: 6
        FractionLength: 4
```

When you are done accessing the board, release the JTAG connection.

```
release(mem)
```

Input Arguments

`mem` — JTAG connection to AXI manager IP running on hardware board

`socAXIManager` object

JTAG connection to AXI manager IP running on your hardware board, specified as an `socAXIManager` object.

Version History

Introduced in R2019a

See Also

`readmemory` | `writememory`

socTaskSchedulability

Determine whether set of tasks can be scheduled for specified core assignments

Syntax

```
[schedulable,tasks,cores] = socTaskSchedulability(mdl)
[schedulable,tasks,cores] = socTaskSchedulability(allocationSet,
allocationScenario)
```

Description

`[schedulable,tasks,cores] = socTaskSchedulability(mdl)` determines the schedulability of the tasks and cores in the System Composer™ or SoC Blockset software model.

`[schedulable,tasks,cores] = socTaskSchedulability(allocationSet, allocationScenario)` determines the schedulability of the tasks and cores in the System Composer allocation set and scenario captured in **Allocation Editor** tool for software and hardware architecture diagrams using the `PeriodicSoftwareTask` and `AperiodicSoftwareTask` stereotypes.

Input Arguments

`mdl` — Name of model

string

Name of the Simulink model or the System Composer model for the processor.

Data Types: `string`

`allocationSet` — Name of allocation set

string

Name of the allocation set.

Data Types: `string`

`allocationScenario` — Name of allocation scenario

string

Name of the allocation scenario.

Data Types: `string`

Output Arguments

`schedulable` — Can task set be scheduled

0 | 1

Indicates whether the task set can be scheduled.

`tasks` — Array of task information
structure array

Task information, returned as an array of structures with the fields:

`name` — Task name
string

Name of the task in the model.

`scheduable` — Can task be scheduled
0 | 1

Indicates whether the task can be scheduled, where 1 indicates the task can be scheduled and 0 indicates the task cannot be scheduled under the current constraints.

`cores` — Core information
structure array

Core information, returned as an array of structures with the fields:

`name` — Core name
string

Name of the core in the processor.

`usage` — Percent usage of core
double

Usage of the core resources as a percentage.

Version History

Introduced in R2022b

See Also

`PeriodicSoftwareTask` | `AperiodicSoftwareTask` | `soc_blockset_profile` | **Schedule Editor** | **Allocation Editor**

Topics
“Systems Engineering Approach for SoC Applications”
“Design SoC Model Using System Composer”
“Compose and Analyze Systems Using Architecture Models” (System Composer)

socCreateModel

Creates a skeleton SoC Blockset model for task set

Syntax

```
socCreateModel(modelName)
socCreateModel(allocationSet,allocationScenario)
```

Description

`socCreateModel(modelName)` creates an SoC Blockset model with task set that you define from a System Composer model using SoC Blockset software task stereotypes. The SoC Blockset model contains a Task Manager block with software tasks assigned, and the **Schedule Editor** partitions for each task in a reference model. The model is set for a `Custom Board` with a required number of processor cores.

`socCreateModel(allocationSet,allocationScenario)` creates an SoC Blockset model with a task set determined by an allocation scenario captured in **Allocation Editor** tool for software and hardware architecture diagrams using the `PeriodicSoftwareTask` and `AperiodicSoftwareTask` stereotypes.

Input Arguments

`modelName` — Software architecture model

string

Name of the software architecture model.

Data Types: `string`

`allocationSet` — Name of allocation set

string

Name of the allocation set.

Data Types: `string`

`allocationScenario` — Name of allocation scenario

string

Name of the allocation scenario.

Data Types: `string`

Version History

Introduced in R2022b

See Also

`PeriodicSoftwareTask` | `AperiodicSoftwareTask` | `soc_blockset_profile` | **Schedule Editor** | **Allocation Editor**

Topics

“Systems Engineering Approach for SoC Applications”
“Design SoC Model Using System Composer”
“Compose and Analyze Systems Using Architecture Models” (System Composer)

socFunctionAnalyzer

Estimate number of operations in MATLAB function

Syntax

```
socFunctionAnalyzer(functionName)
socFunctionAnalyzer(functionName,Name=Value)
report = socFunctionAnalyzer(___)
[report,y1,...,yn] = socFunctionAnalyzer(___)
```

Description

`socFunctionAnalyzer(functionName)` generates a report with the estimated number of operations in the MATLAB function specified by `functionName`.

The function generates the report as a Microsoft® Excel® spreadsheet and a MAT-file. The function also provides a link to view the report in a separate dialog box.

The report includes information for each mathematical or logical operator in the function, with individual lines for each operator and data type. For example, multiplication with data type `double` and multiplication with data type `uint32` are listed separately. The report lists each instance of the operator as a separate line. The report includes these fields.

- `Path` – The path to the operator within the structural hierarchy of the top function
- `Count` – The number of times the operator is executed in the design
- `Operator` – The operator used
- `DataType` – The data type used for the output of the operator
- `Link` – A link to the location of the operator in the function

For more information, see “Using the Algorithm Analyzer Report”.

`socFunctionAnalyzer(functionName,Name=Value)` specifies options using one or more name-value arguments. For example, `IncludeOperator='+'` specifies that the generated report only includes `'+'` operator counts.

`report = socFunctionAnalyzer(___)` returns a structure of tables that contain report information. Specify any of the input argument combinations from previous syntaxes.

`[report,y1,...,yn] = socFunctionAnalyzer(___)` returns the outputs `y1,...,yn` of the specified function. Specify any of the input argument combinations from previous syntaxes.

Examples

Analyze Resources in Function

This example calculates the number of dynamic operations and static operators in the function `soc_test_func.m`.

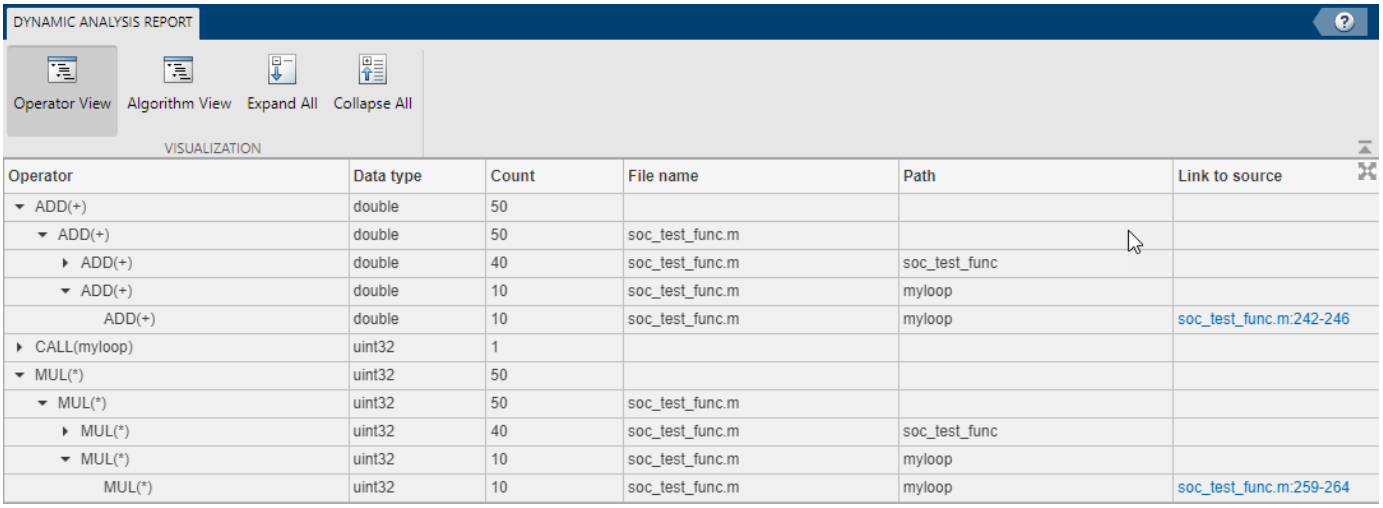
Generate Dynamic Report

`soc_test_func` takes two input arguments of type uint32. Use the `FunctionInputs` argument to create a report with 10 and 20 as inputs to the `soc_test_func` function. The dynamic report is generated in a folder named `report`.

```
socFunctionAnalyzer('soc_test_func.m',FunctionInputs={10,20},Folder='report');
```

View Generated Report

After execution, the `socFunctionAnalyzer` function provides a link to the generated report. Click the link titled *Open report viewer*. The report opens in a separate window:



Operator	Data type	Count	File name	Path	Link to source
ADD(+)	double	50			
ADD(+)	double	50	soc_test_func.m		
ADD(+)	double	40	soc_test_func.m	soc_test_func	
ADD(+)	double	10	soc_test_func.m	myloop	
ADD(+)	double	10	soc_test_func.m	myloop	soc_test_func.m:242-246
CALL(myloop)	uint32	1			
MUL(*)	uint32	50			
MUL(*)	uint32	50	soc_test_func.m		
MUL(*)	uint32	40	soc_test_func.m	soc_test_func	
MUL(*)	uint32	10	soc_test_func.m	myloop	
MUL(*)	uint32	10	soc_test_func.m	myloop	soc_test_func.m:259-264

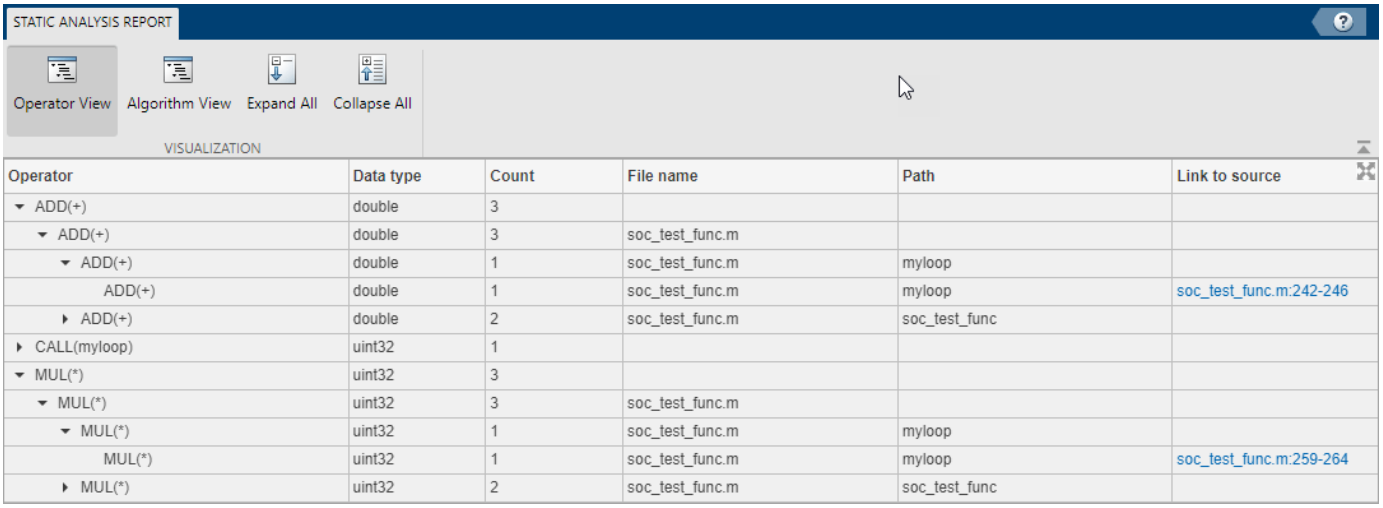
The result shows that the `ADD` operator is used 50 times with data type `double`. The call to `myloop` executes once with data type `uint32`, and the `MUL` operator is used 50 times with data type `uint32`.

Generate Static Report

To statically calculate the number of operators in the function (and not the number of operations performed during simulation), use the `AnalysisMethod` name-value pair argument.

```
socFunctionAnalyzer('soc_test_func.m',AnalysisMethod='static',FunctionInputs={10,20},Folder='repo
```

To view the static analysis report, click the link titled *Open report viewer*. The report opens in a separate window:



Operator	Data type	Count	File name	Path	Link to source
ADD(+)	double	3			
ADD(+)	double	3	soc_test_func.m		
ADD(+)	double	1	soc_test_func.m	myloop	
ADD(+)	double	1	soc_test_func.m	myloop	soc_test_func.m:242-246
ADD(+)	double	2	soc_test_func.m	soc_test_func	
CALL(myloop)	uint32	1			
MUL(*)	uint32	3			
MUL(*)	uint32	3	soc_test_func.m		
MUL(*)	uint32	1	soc_test_func.m	myloop	
MUL(*)	uint32	1	soc_test_func.m	myloop	soc_test_func.m:259-264
MUL(*)	uint32	2	soc_test_func.m	soc_test_func	

Input Arguments

`functionName` — MATLAB function to analyze

character vector | string scalar

MATLAB function to analyze, specified as a character vector or string scalar that indicates the function name or file name.

Example: `'soc_analyze_FFT_tb.m'`

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `socFunctionAnalyzer('mySocFunction.m','Folder'='report_sym')`

`AnalysisMethod` — static or dynamic analysis

`'dynamic'` (default) | `'static'`

Option for static of dynamic analysis, specified as `'static'` or `'dynamic'`. This value specifies the method that the function uses to analyze the MATLAB function.

- `'dynamic'` – Dynamic analysis provides the number of operations that the function executes during a given simulation time.
- `'static'` – Static analysis provides the number of operators in an algorithm.

`FunctionInputs` — Inputs for function to be analyzed

cell array

Inputs for the function to be analyzed, specified as a cell array. The `socFunctionAnalyzer` function evaluates the function to be analyzed, `functionName`, with these specified values as inputs. If you do

not specify this name-value argument, then `socFunctionAnalyzer` passes no arguments to `functionName`.

If `functionName` expects input arguments, then you must specify this name-value pair argument. Otherwise, the `socFunctionAnalyzer` function errors.

Example: `FunctionInputs={10,fi(20)}`

`Folder` — Folder location of generated report
current folder (default) | character vector | string scalar

Folder location of generated report, specified as a character vector or string scalar indicating the folder path. Specify the path to the location for the generated output reports as a full path or relative path.

Example: `Folder='C:/Work/mydir'`

`IncludeOperator` — Operators to include in generated report
all available operators (default) | character vector | string scalar | cell array of strings | cell array of character vectors

Operators to include in the generated report , specified as a character vector or string scalar to specify one operator. Use cell array of character vectors or string scalars to specify multiple operators. When you do not specify this name-value pair argument, the `socFunctionAnalyzer` function includes all operators, except for the operators specified by the `ExcludeOperator` name-value pair argument.

Example: `IncludeOperator='+'`

Example: `IncludeOperator={'+','IF','MUL'}`

`ExcludeOperator` — Operators to exclude from generated report
character vector | string scalar | cell array of strings | cell array of character vectors

Operators to exclude from the generated report, specified as a character vector or string scalar to specify one operator. Use cell array of character vectors or string scalars to specify multiple operators. When you do not specify this name-value pair argument, the `socFunctionAnalyzer` function includes all operators in the report.

Example: `ExcludeOperator='-'`

Example: `ExcludeOperator={'-','CALL'}`

`IncludeFunction` — Functions to include in generated report
all functions in function hierarchy (default) | character vector | string scalar | cell array of strings | cell array of character vectors

Functions to include in generated report, specified as a character vector or string scalar to specify one function or file name. Use cell array of character vectors or string scalars to specify multiple functions or file names. If you do not specify this name-value pair argument, the `socFunctionAnalyzer` function includes all functions in the report, except for the functions specified by the `'ExcludeFunction'` name-value pair argument. Use the `'IncludeFunction'` name-value pair when you have a test bench function, and you only want to analyze one of the functions it calls.

Example: `IncludeFunction='myFunc.m'`

Example: `IncludeFunction={'myFunc.m','func2'}`

`ExcludeFunction` — Functions to exclude from generated report
character vector | string scalar | cell array of strings | cell array of character vectors

Functions to include in generated report, specified as a character vector or string scalar to specify one function or file name. Use cell array of character vectors or string scalars to specify multiple functions or file names. If you do not specify this name-value pair argument, the `socFunctionAnalyzer` function includes all functions in the report.

Example: `ExcludeFunction='myFunc.m'`

Example: `ExcludeFunction={'myFunc.m','func2'}`

`Verbose` — Display verbose messages
`off` (default) | `on` | `quiet`

Display verbose messages, specified as `on`, `off`, or `quiet`. When you set `Verbose` to:

- `on` — the function displays detailed messages during the different stages of execution.
- `off` — the function displays messages during the first and last stages of execution.
- `quiet` — the function does not display messages about execution. Select this option if you want to silence all messages.

Example: `Verbose='on'`

Output Arguments

`report` — Operator count raw data
structure

Function operator count, returned as a structure of five tables:

- `OperatorDetailedReport` – A fully detailed report per operator
- `OperatorAggregatedReport` – An aggregated operator view, with one line for each type of operator
- `OperatorHierarchicalReport` – A hierarchical operator view
- `PathAggregatedReport` – An aggregated model view
- `PathHierarchicalReport` – A Hierarchical model view

Each table contains raw data from which the function generates an HTML view, and a link to view the data in a report window. The generated Excel file has five sheets, containing the information from the five tables. For more information about the generated report, see “Using the Algorithm Analyzer Report”.

`y1,...,yn` — Analyzed function output (as separate arguments)
calculated by `functionName`

Analyzed function output, returned as an output the `functionName` input function.

Version History

Introduced in R2020a

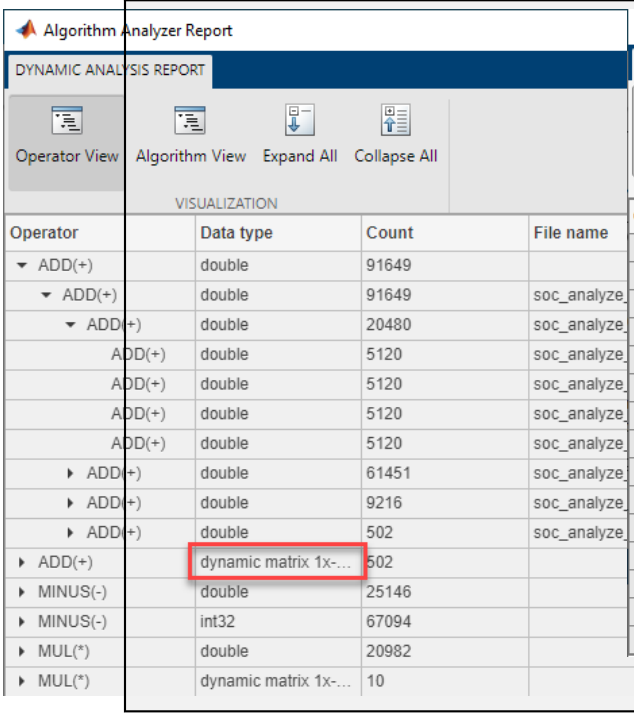
R2022b: Values for `Verbose` argument changed
Behavior change in future release

Before R2022b, the `Verbose` argument accepted `true` or `false` as values. Starting this release, the function throws an error when using `true` or `false`, asking you to set the `Verbose` argument to `on`, `off`, or `quiet`.

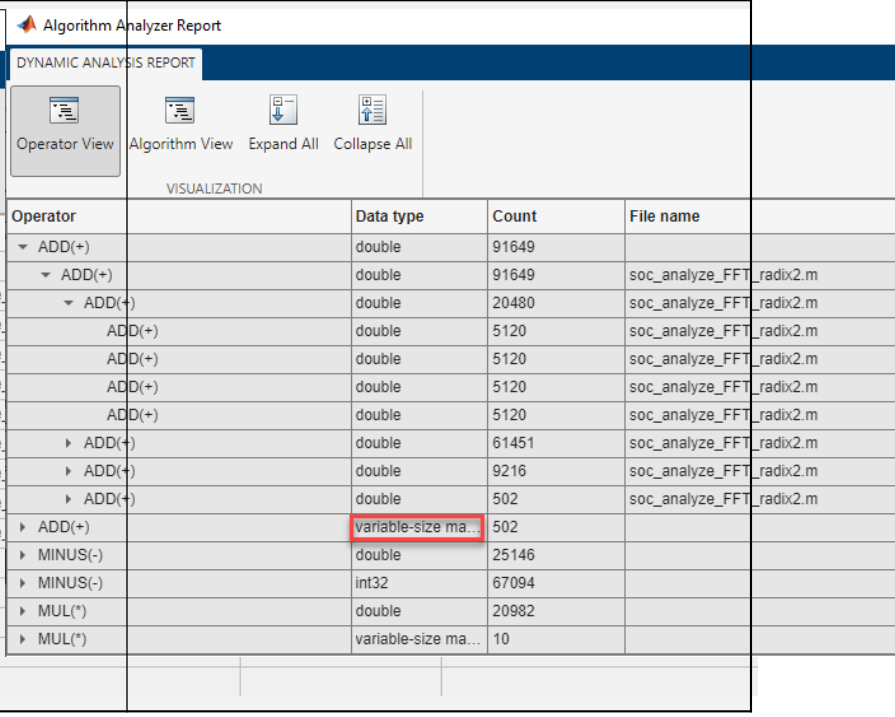
R2022b: `dynamic-matrix` data type is replaced with `variable-size matrix`

The `dynamic-matrix` as a data type of an `Operator` is now replaced with `variable-size matrix`. The table displays this change in **Algorithm Analyzer Report**.

Before R2022b



Starting R2022b



See Also

`socModelAnalyzer` | `socAlgorithmAnalyzerReport`

Topics

“Using the Algorithm Analyzer Report”

socModelAnalyzer

Estimate number of operations in Simulink model

Syntax

```
socModelAnalyzer(modelName)
socModelAnalyzer(modelName,Name=Value)
report = socModelAnalyzer(___)
```

Description

`socModelAnalyzer(modelName)` generates a report with the estimated number of operations in a Simulink model specified by `modelName`.

The function generates the report as a Microsoft Excel spreadsheet and a MAT-file. The function also provides a link to view the report in a separate dialog box.

The report includes information for each mathematical or logical operator in the function, with individual lines for each operator and data type. For example, multiplication with data type `double` and multiplication with data type `uint32` are listed separately. The report lists each instance of the operator as a separate line. The report includes these fields.

- `Path` – The path to the operator within the structural hierarchy of the top function
- `Count` – The number of times the operator is executed in the design
- `Operator` – The operator used
- `DataType` – The data type used for the output of the operator
- `Link` – A link to the location of the operator in the function

For more information, see “Using the Algorithm Analyzer Report”.

`socModelAnalyzer(modelName,Name=Value)` specifies options using one or more name-value pair arguments. For example, `IncludeOperator='+'` specifies that the generated report only includes `'+'` operator counts.

`report = socModelAnalyzer(___)` returns a structure of tables that contain report information. Specify any of the input argument combinations from previous syntaxes.

Examples

Analyze Resources in Model

Calculate the number of dynamic operations and static operators in the model `testmdl.slx`.

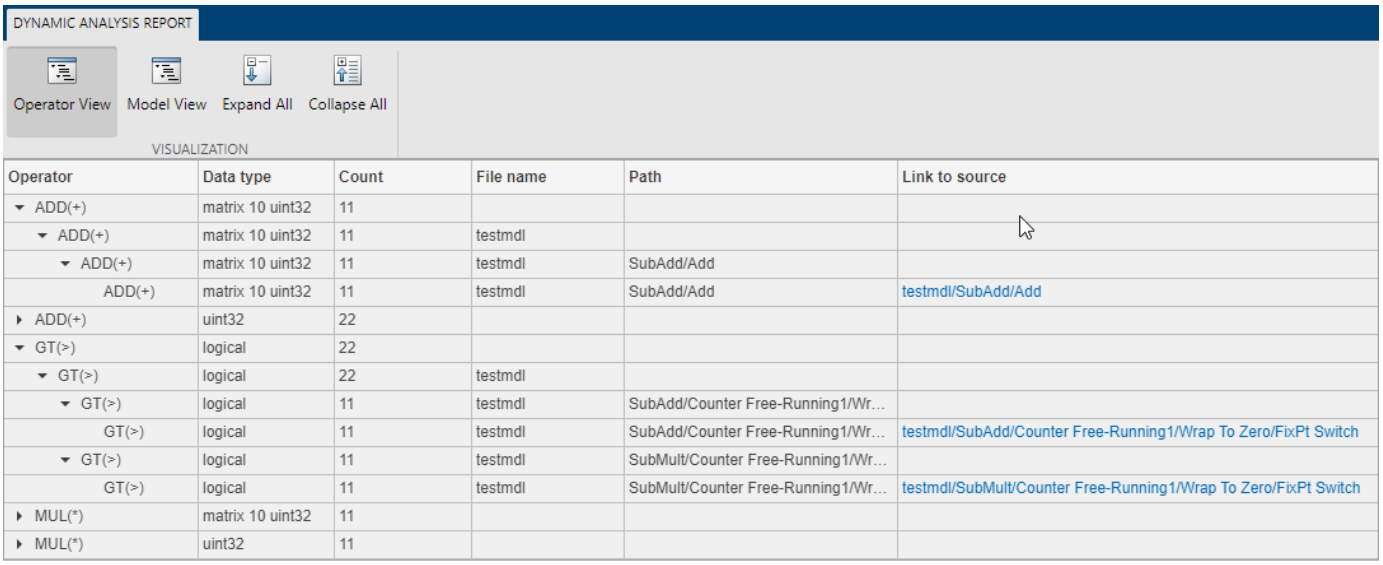
Generate Dynamic Report

Count operations in `testmdl`, and generate a report in a folder named `report`.

```
socModelAnalyzer('testmdl.slx',Folder='report');
```

View Dynamic Report

After generating the dynamic report, the `socModelAnalyzer` function provides a link to the generated report. View the dynamic analysis report by clicking the **Open report viewer** link. The report opens in a separate window.



Operator	Data type	Count	File name	Path	Link to source
ADD(+)	matrix 10 uint32	11			
ADD(+)	matrix 10 uint32	11	testmdl		
ADD(+)	matrix 10 uint32	11	testmdl	SubAdd/Add	
ADD(+)	matrix 10 uint32	11	testmdl	SubAdd/Add	testmdl/SubAdd/Add
ADD(+)	uint32	22			
GT(>)	logical	22			
GT(>)	logical	22	testmdl		
GT(>)	logical	11	testmdl	SubAdd/Counter Free-Running1/Wr...	
GT(>)	logical	11	testmdl	SubAdd/Counter Free-Running1/Wr...	testmdl/SubAdd/Counter Free-Running1/Wrap To Zero/FixPt Switch
GT(>)	logical	11	testmdl	SubMult/Counter Free-Running1/Wr...	
GT(>)	logical	11	testmdl	SubMult/Counter Free-Running1/Wr...	testmdl/SubMult/Counter Free-Running1/Wrap To Zero/FixPt Switch
MUL(*)	matrix 10 uint32	11			
MUL(*)	uint32	11			

The result shows that the `ADD` operator is used 11 times with data type `matrix 10 uint32`, and 22 times with data type `uint32`. The `GT` (greater than) operator was used 22 times total with data type `logical`: 11 times from **SubAdd** model, and 11 times from **SubMult** model. The `MUL` operator is used 11 times with data type `uint32`, and 11 times with `matrix 10 uint32`.

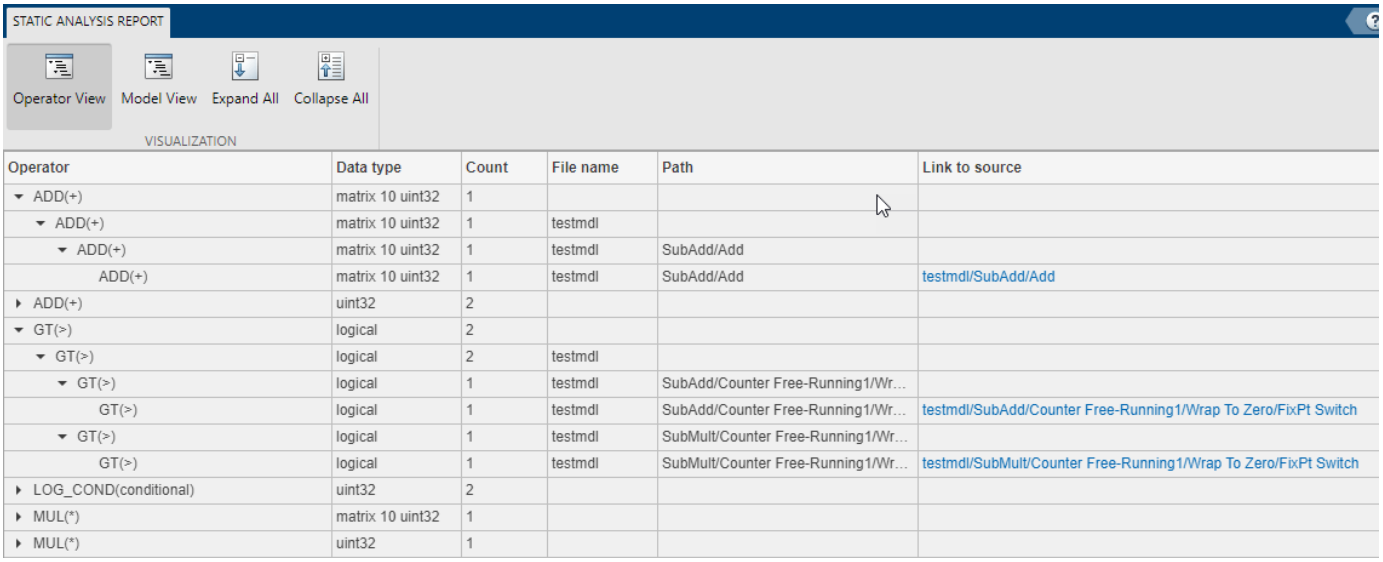
Generate Static Report

Statically calculate the number of operators in the model by using the `AnalysisMethod` name-value argument.

```
socModelAnalyzer('testmdl.slx',AnalysisMethod='static',Folder='report');
```

View Static Report

View the static analysis report, by clicking the **Open report viewer** link. The static report opens in a separate window.



Operator	Data type	Count	File name	Path	Link to source
ADD(+)	matrix 10 uint32	1			
ADD(+)	matrix 10 uint32	1	testmdl		
ADD(+)	matrix 10 uint32	1	testmdl	SubAdd/Add	
ADD(+)	matrix 10 uint32	1	testmdl	SubAdd/Add	testmdl/SubAdd/Add
ADD(+)	uint32	2			
GT(>)	logical	2			
GT(>)	logical	2	testmdl		
GT(>)	logical	1	testmdl	SubAdd/Counter Free-Running1/Wr...	
GT(>)	logical	1	testmdl	SubAdd/Counter Free-Running1/Wr...	testmdl/SubAdd/Counter Free-Running1/Wrap To Zero/FixPt Switch
GT(>)	logical	1	testmdl	SubMult/Counter Free-Running1/Wr...	
GT(>)	logical	1	testmdl	SubMult/Counter Free-Running1/Wr...	testmdl/SubMult/Counter Free-Running1/Wrap To Zero/FixPt Switch
LOG_COND(conditional)	uint32	2			
MUL(*)	matrix 10 uint32	1			
MUL(*)	uint32	1			

Input Arguments

`modelName` — Simulink model to analyze

character vector | string

Simulink model to analyze, specified as a character vector or string scalar.

Example: `'soc_analyze_FFT_top.slx'`

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `socModelAnalyzer('mySocModel.slx',Folder='report_sym')`

`AnalysisMethod` — static or dynamic analysis

`'dynamic'` (default) | `'static'`

Option for static of dynamic analysis, specified as `'static'` or `'dynamic'`. This value specifies the method that the function uses to analyze the Simulink model.

- `'dynamic'` – Dynamic analysis provides the number of operations the model executes during a given simulation time.
- `'static'` – Static analysis provides the number of operators in an algorithm.

Example: `AnalysisMethod='static'`

`Folder` — Folder location of generated report

current folder (default) | character vector | string scalar

Folder location of generated report, specified as a character vector or string scalar indicating the folder path. Specify the path to the location for the generated output reports as a full path or relative path.

Example: `Folder='C:/Work/mydir'`

`IncludeOperator` — Operators to include in generated report
all available operators (default) | character vector | string scalar | cell array of strings | cell array of character vectors

Operators to include in the generated report , specified as a character vector or string scalar to specify one operator. Use cell array of character vectors or string scalars to specify multiple operators. When you do not specify this name-value pair argument, the `socModelAnalyzer` function includes all operators, except for the operators specified by the `ExcludeOperator` name-value pair argument.

Example: `IncludeOperator='+'`

Example: `IncludeOperator={'+','IF','MUL'}`

`ExcludeOperator` — Operators to exclude from generated report
character vector | string scalar | cell array of strings | cell array of character vectors

Operators to exclude from the generated report, specified as a character vector or string scalar to specify one operator. Use cell array of character vectors or string scalars to specify multiple operators. When you do not specify this name-value pair argument, the `socModelAnalyzer` function includes all operators in the report.

Example: `ExcludeOperator='-'`

Example: `ExcludeOperator={'-','CALL'}`

`IncludeBlockPath` — Models to include in generated report
all models or blocks in top model hierarchy (default) | character vector | string | cell array of strings | cell array of character vectors

Models or blocks to include in generated report, specified as a character vector or string scalar to specify one block or model. Use cell array of character vectors or string scalars to specify multiple blocks or models. If you do not specify this name-value pair argument, the `socModelAnalyzer` function includes all models and blocks in the report, except for the blocks specified by the `'ExcludeBlockPath'` name-value pair argument. Use the `'IncludeBlockPath'` name-value pair when you have a test bench model, and you only want to analyze one of the models it includes.

Example: `IncludeBlockPath='myModel.slx'`

Example: `IncludeBlockPath={'myModel.slx','myIfft'}`

`ExcludeBlockPath` — Models to exclude from generated report
character vector | string | cell array of strings | cell array of character vectors

Models or blocks to include in generated report, specified as a character vector or string scalar to specify one block or model. Use cell array of character vectors or string scalars to specify multiple blocks or models. If you do not specify this name-value pair argument, the `socModelAnalyzer` function includes all models and blocks in the report.

Example: `ExcludeBlockPath='myOtherModel.slx'`

Example: `ExcludeBlockPath={'myOtherModel.slx','myIfft'}`

`Verbose` — Display verbose messages
`off` (default) | `on` | `quiet`

Display verbose messages, specified as `on`, `off`, or `quiet`. When you set `Verbose` to:

- `on` — the function displays detailed messages during the different stages of execution.
- `off` — the function displays messages during the first and last stages of execution.
- `quiet` — the function does not display messages about execution. Select this option if you want to silence all messages.

Example: `Verbose='on'`

Output Arguments

`report` — Operator count raw data
structure

Model operator count, returned as a structure of five tables:

- `OperatorDetailedReport` – A fully detailed report per operator
- `OperatorAggregatedReport` – An aggregated operator view, with one line for each type of operator
- `OperatorHierarchicalReport` – A hierarchical operator view
- `PathAggregatedReport` – An aggregated model view
- `PathHierarchicalReport` – A Hierarchical model view

Each table contains raw data from which the function generates an HTML view, and a link to view the data in a report window. The generated Excel file has five sheets, containing the information from the five tables. For more information about the generated report, see “Using the Algorithm Analyzer Report”.

Limitations

- This function does not support AUTOSAR Blockset blocks or models.
- This function does not support Simulink send and receive messages.

Tips

Identical atomic subsystems are optimized by Simulink into a single function by default, which can lead to incorrect results. To avoid this process, perform one of these actions:

- Clear the **Treat as atomic unit** parameter in the subsystem block mask.
- Set the **Function packaging** parameter to `Nonreusable function` in the subsystem block mask.
- Modify one of the two subsystems so that it is not identical to the other one.

Version History

Introduced in R2020a

R2022b: Values for `Verbose` argument changed
Behavior change in future release

Before R2022b, the `Verbose` argument accepted `true` or `false` as values. Starting this release, the function throws an error when using `true` or `false`, asking you to set the `Verbose` argument to `on`, `off`, or `quiet`.

R2022b: `dynamic-matrix` data type is replaced with `variable-size matrix`

The `dynamic-matrix` as a data type of an `Operator` is now replaced with `variable-size matrix`.

See Also

`socFunctionAnalyzer` | `socAlgorithmAnalyzerReport`

Topics
“Using the Algorithm Analyzer Report”

socExportReferenceDesign

Export custom reference design for HDL Workflow Advisor

Syntax

```
socExportReferenceDesign(topModelName)
socExportReferenceDesign(topModelName,Name,Value)
```

Description

`socExportReferenceDesign(topModelName)` exports a custom reference design from an SoC Blockset model with name `topModelName`. To create an SoC Blockset model, you must perform one of these actions.

- Create a model using an SoC Blockset template. For more information, see “Use Template to Create SoC Model”.
- Open Simulink. On the **Apps** tab click **System on Chip (SoC)**.
- In an existing Simulink model, click **Model Settings** in the **Modeling** tab. In the left pane, select **Hardware Implementation**. Then, set **Hardware board** to a supported SoC board. For a list of supported SoC boards, see “Supported Third-Party Tools and Hardware”.

Use this exported design with **HDL Workflow Advisor** (requires HDL Coder™ license). Use this function to eliminate the manual steps for creating a custom reference design, as described in “Create a Custom Hardware Platform” (HDL Coder). Use the exported reference design in the IP core generation workflow with the HDL Workflow advisor. For more information, see “Hardware Targeting Basics” (HDL Coder).

For more information about the **HDL Workflow Advisor** app, see “Getting Started with the HDL Workflow Advisor” (HDL Coder).

To use this function, you must first install Xilinx Vivado® or Intel Quartus®.

`socExportReferenceDesign(topModelName,Name,Value)` specifies options using one or more name-value pair arguments.

Examples

Export Custom Reference Design from SoC Model

Export a custom reference design from the `soc_image_rotaion.slx` model.

```
socExportReferenceDesign('soc_image_rotation')
```

Export Reference Design Using Specific Arguments

Export a custom reference design from the `soc_hwsw_stream_top` model.

- Exclude the DUT named "FPGA Algorithm Wrapper" from the reference design.
- Place the generated output in folder `C:/Work`.
- Generate a board definition file with board name "My ZC706 Board". This name appears in the **Target platform** menu in the **HDL Workflow Advisor** app.
- Generate reference design definition file with the design name `My ZC706 Design`.

```
socExportReferenceDesign('soc_hwsw_stream_top',...
    'DUTName','FPGA Algorithm Wrapper',...
    'Folder','C:/Work',...
    'TargetPlatform','My ZC706 Board',...
    'ReferenceDesign','My ZC706 Design')
```

Input Arguments

`topModelName` — Name of top Simulink model

character vector | string scalar

Name of the top Simulink model, specified as a character vector or string scalar. The reference design is exported from the `topModelName` model. This model must be an SoC Blockset model.

Example: `'soc_hw_sw_stream_top'` specifies the model with name `'soc_hw_sw_stream_top'`.

Data Types: `char` | `string`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example:
`socExportReferenceDesign('soc_image_rotation','Folder','refDesignFolder')`
exports a reference design from the model `soc_image_rotation`, and places the generated files in a folder named `refDesignFolder`.

`DUTName` — Name of DUT subsystem to exclude from reference design

inferred (default) | character vector | string scalar

Name of DUT subsystem to exclude from reference design, specified as a character vector or string scalar. When there is one DUT in the model, the function infers the `DUTName` and sets it as the name of the DUT in the model. You must specify this name-value pair argument when the FPGA model has more than one DUT.

Example: `'soc_image_rotation_fpga/ImageRotation'`

Data Types: `char` | `string`

`Folder` — Folder location for exported reference design files

`topModelName``_refdesign` (default) | character vector | string scalar

Folder location for the exported reference design files, specified as a character vector or string scalar. When not specified, the files are placed in a folder named *`topModelName`*`_refdesign`, where *`topModelName`* is the name of the model.

Example: `'C:/Work/refDesign'`

Data Types: `char` | `string`

`TargetPlatform` — Name of target platform
same as SoC model (default) | character vector | string scalar

Name of the target platform, specified as the comma-separated pair consisting of `'TargetPlatform'` and a character vector or string scalar. When you do not specify this value, the name of the target platform matches the **Hardware Board** parameter value in the SoC model configuration parameters. In the **HDL Workflow Advisor** tool, this target platform name appears as *`TargetPlatform`* `(generated by SoC Blockset)`, where *`TargetPlatform`* is the value for this name-value pair argument.

Example: `'Xilinx Zynq ZC706 evaluation kit'`

Data Types: `char` | `string`

`ReferenceDesign` — Name of generated reference design
`topModelName` `model` (default) | character vector | string scalar

Name of the generated reference design, specified as the comma-separated pair consisting of `'ReferenceDesign'` and a character vector or string scalar. When you do not specify this value, the name of the generated reference design is *`topModelName`* `model`, where *`topModelName`* is specified by the input `topModelName`.

Example: `'My ZC706 Design'`

Data Types: `char` | `string`

Version History

Introduced in R2020a

See Also

SoC Builder | `hdladvisor`

Topics
“Create a Custom Hardware Platform” (HDL Coder)
“Generate SoC Design”

socAlgorithmAnalyzerReport

Open algorithm analysis report

Syntax

```
socAlgorithmAnalyzerReport(reportfile)
```

Description

`socAlgorithmAnalyzerReport(reportfile)` opens the specified report generated by the `socFunctionAnalyzer` or `socModelAnalyzer` function. The report opens in a separate window titled Algorithm Analyzer Report.

Examples

Open Algorithm Analysis Report

Use the `socFunctionAnalyzer` function to generate a report. Then, open the generated report:

```
socFunctionAnalyzer('soc_test_func.m','FunctionInputs',{10,20},'Folder','report');
socAlgorithmAnalyzerReport('report/soc_test_func.mat');
```

Input Arguments

`reportfile` — Path to report file

character vector | string scalar

Path to report file, specified as a character vector or string scalar that indicates the name of a MAT-file, generated by the `socFunctionAnalyzer` or `socModelAnalyzer` function.

Example: `'report/soc_test_func.mat'`

Data Types: `char` | `string`

Version History

Introduced in R2020a

See Also

`socModelAnalyzer` | `socFunctionAnalyzer`

Topics

“Using the Algorithm Analyzer Report”

buildModel

Build SoC model

Syntax

`buildModel(obj)`

Description

`buildModel(obj)` builds the SoC model using the Xilinx Vivado or Intel Quartus tool. The function generates a bitstream for your FPGA design and a compiled executable for your software.

Examples

Build SoC Model Using Command Line Interface

Use MATLAB commands to build an SoC model by creating and configuring the `socModelBuilder` object.

Set the synthesis tool path to point to an installed Vivado Design Suite. When you execute this command, use your own Xilinx Vivado installation path.

```
hdlsetuptoolpath('ToolName','Xilinx Vivado','ToolPath', ...
 'C:\Xilinx\Vivado\2022.1\bin\vivado.bat');
```

Create an `socModelBuilder` object for the `soc_rfsoc_datacapture` SoC model. To run the generated software model in external mode, set `ExternalMode` to `true`.

```
obj = socModelBuilder('soc_rfsoc_datacapture',ExternalMode=true)
```

```
obj =

  socModelBuilder with properties:

              ModelName: 'soc_rfsoc_datacapture'
          ProjectFolder: 'soc_prj'
              BuildType: 'Processor and FPGA'
           ExternalMode: 1
    RunExternalFPGABuild: 1
```

Change the project folder to `G:\R2023a\myProject`.

```
obj.ProjectFolder = 'G:\R2023a\myProject'
```

```
obj =

  socModelBuilder with properties:

              ModelName: 'soc_rfsoc_datacapture'
          ProjectFolder: 'G:\R2023a\myProject'
              BuildType: 'Processor and FPGA'
```

```
            ExternalMode: 1
    RunExternalFPGABuild: 1
```

Build the model.

```
buildModel(obj);
```

Input Arguments

obj — SoC model builder

`socModelBuilder`

SoC model builder, specified as an `socModelBuilder` object. Use this object to build, load, and execute an SoC model on the SoC or FPGA board.

Version History

Introduced in R2023a

See Also

`socModelBuilder`

loadBinaries

Load existing binaries onto hardware board

Syntax

```
loadBinaries(obj)
loadBinaries(obj,hardware)
```

Description

`loadBinaries(obj)` loads the binaries in the specified project folder onto the most recently connected hardware board.

`loadBinaries(obj,hardware)` loads the binaries in the specified project folder onto the hardware board `hardware`.

Examples

Load Existing Binaries Using Command Line Interface

Use MATLAB commands to load binaries in a project folder onto the ZedBoard™.

Set the synthesis tool path to point to an installed Vivado Design Suite. When you execute this command, use your own Xilinx Vivado installation path.

```
hdlsetuptoolpath('ToolName','Xilinx Vivado','ToolPath', ...
 'C:\Xilinx\Vivado\2022.1\bin\vivado.bat');
```

Create an `socModelBuilder` object for the `soc_hwsw_top` SoC model.

```
obj = socModelBuilder('soc_hwsw_top')
```

```
obj =

  socModelBuilder with properties:

               ModelName: 'soc_hwsw_top'
           ProjectFolder: 'soc_prj'
               BuildType: 'Processor and FPGA'
            ExternalMode: 0
      RunExternalFPGABuild: 1
```

Change the project folder to `G:\R2023a\myProject`.

```
obj.ProjectFolder = 'G:\R2023a\myProject'
```

```
obj =

  socModelBuilder with properties:

               ModelName: 'soc_hwsw_top'
```

```
          ProjectFolder: 'G:\R2023a\myProject'
              BuildType: 'Processor and FPGA'
           ExternalMode: 0
    RunExternalFPGABuild: 1
```

Load the binaries in the `G:\R2023a\myProject` project folder to the most recently connected hardware board.

```
loadBinaries(obj);
```

Load the binaries in the `G:\R2023a\myProject` project folder onto the ZedBoard.

```
hw = socHardwareBoard('ZedBoard');
loadBinaries(obj,hw);
```

Input Arguments

`obj` — SoC model builder

`socModelBuilder` object

SoC model builder, specified as an `socModelBuilder` object. Use this object to build, load, and execute an SoC model on the SoC or FPGA board.

`hardware` — Hardware object

`socHardwareBoard`

Hardware object, specified as a `socHardwareBoard` object that represents the connection to the SoC or FPGA board.

Version History

Introduced in R2023a

See Also

`socModelBuilder` | `socHardwareBoard`

Objects

soc.iosource

Input source on SoC hardware board

Description

Create an `soc.iosource` object to connect to an input source on an SoC hardware board. Pass the `soc.iosource` object as an argument to the `addSource` function of the `DataRecorder` object.

The sources available on the design running on the SoC hardware board correspond to the blocks you included in your Simulink model. When you run **SoC Builder**, it connects your FPGA logic with the matching interface on the board.

Source	Block	Action
`'TCP Receive'`	TCP Read	Read UDP (User Datagram Protocol) data from the Linux socket buffer.
`'UDP Receive'`	UDP Read	Read TCP/IP data from Linux socket buffer.
`'AXI Register Read'`	Register Read	Read registers from an IP core using the AXI interface.
`'AXI Stream Read'`	Stream Read	Read AXI-4 Stream data using IIO.

Creation

Syntax

```
availableSources = soc.iosource(hw)
src = soc.iosource(hw,inputSourceName)
```

Description

`availableSources = soc.iosource(hw)` returns a list of input sources available for data logging on the SoC hardware board connected through `hw`. `hw` is an `socHardwareBoard` object.

`src = soc.iosource(hw,inputSourceName)` creates a source object corresponding to `inputSourceName` on the SoC hardware board connected through `hw`.

Input Arguments

`hw` — Hardware object
`socHardwareBoard` object

Hardware object, specified as a `socHardwareBoard` object that represents the connection to the SoC hardware board.

`inputSourceName` — Name of available input source on SoC hardware board

character vector

Name of an available input source on the SoC hardware board, specified as a character vector. To get the list of input sources available for data logging on the specified SoC hardware board, call the `soc.iosource` function without arguments.

Example: 'UDP Receive'

Data Types: `char`

Output Arguments

`availableSources` — List of input data sources available for data logging

cell array

List of input data sources available for data logging on the specified SoC hardware board, returned as a cell array. Each cell contains a character vector with the name of an available input data source for data logging on the specified SoC hardware board. Use one of these names as the `inputSourceName` argument when you create a source object.

`src` — Source object for specified input source

`soc.iosource` object

Source object for specified input source, returned as an `soc.iosource`.

Properties

`DeviceName` — Name of IP core device

character vector

Name of IP core device, specified as a character vector.

Example: `'mwipcore0:s2mm0'`

Dependencies

To enable this property, create a AXI register or AXI stream source object.

Data Types: `char`

`RegisterOffset` — Offset from base address of IP core to register

positive scalar

Offset from the base address of the IP core to the register, specified as a positive scalar.

Dependencies

To enable this property, create a AXI register source object.

Data Types: `uint32`

`LocalPort` — IP port on hardware board where UDP or TCP data is received

`25000` (UDP) (default) | `-1` (TCP) | integer from 1 to 65,535

IP port on hardware board where UDP or TCP data is received specified as a scalar from 1 to 65,535. The object reads UDP or TCP data received on this port of the specified SoC hardware board.

For a TCP object with the `NetworkRole` property to `'Client'`, set `LocalPort` to `-1` to assign any random available port on the hardware board as the local port.

Dependencies

To enable this property, create a TCP or UDP source object.

Data Types: `uint16`

`NetworkRole` — Network role

`'Client'` (default) | character vector

Network role, specified as a character vector.

Example: 'Client'

Dependencies

To enable this property, create a TCP source object.

Data Types: `enumerated string`

`RemoteAddress` — IP address of remote server from which data is received

`'127.0.0.1'` (default) | dotted-quad expression

IP address of the remote server from which data is received, specified as a dotted-quad expression.

Dependencies

To enable this property, create a TCP source object.

Data Types: `char`

`RemotePort` — IP port number of remote server from which data is received

25000 (default) | integer from 1 to 65,535

IP port number of the remote server from which data is received, specified as an integer from 1 to 65,535.

Dependencies

To enable this property, create a TCP source object.

Data Types: `double`

`DataLength` — Length of data packet or register data vector

1 (default) | positive scalar

Maximum length of UDP or TCP data packet, or word length of AXI register data vector, specified as a positive scalar.

Data Types: `double`

`SamplesPerFrame` — Size of data vector read from IP core

nonnegative scalar

Size of the data vector read from the IP core, specified as a nonnegative scalar.

Dependencies

To enable this property, create a AXI stream source object.

Data Types: `double`

`DataType` — Data type of received data
`'uint8'` (default) | `'uint16'` | `'uint32'` | `'int8'` | `'int16'` | `'int32'` | `'double'` | `'single'`

Data type of received data, specified as `'uint8'`, `'uint16'`, `'uint32'`, `'int8'`, `'int16'`, `'int32'`, `'double'` or `'single'`.

Data Types: `char`

`ReceiveBufferSize` — Internal buffer size of object
`65535` (default) | array

Internal buffer size of object, specified as an array.

Dependencies

To enable this property, create a TCP or UDP source object.

Data Types: `double`

`Sample Time` — Sample time
`1` (default) | nonnegative scalar

Sample time, in seconds, at which you want to receive data, specified as an nonnegative scalar.

Data Types: `double`

Examples

Record Data From SoC Hardware Board

Create a connection from MATLAB to the specified SoC hardware board using the IP address, username, and password of the board.

```
hw = socHardwareBoard('Xilinx Zynq ZC706 evaluation kit','hostname','192.168.1.18','username','ro
```

Create a data recording session on the SoC hardware board by using the `hw` object. The resulting `DataRecorder` object represents the data recording session on the SoC hardware board.

```
dr = soc.recorder(hw)
```

```
dr =

  DataRecorder with properties:

         HardwareName: 'Xilinx Zynq ZC706 evaluation kit'
              Sources: {}
            Recording: false
```

List the input sources added to the data recording session.

```
dr.Sources(hw)
```

```
ans =

  1×0 empty cell array
```

By default, `soc.recorder` objects have no added input sources. To add an input source to the data recording session, first create an input source object by using the `soc.iosource` function. For this example, create an User Datagram Protocol (UDP) source object.

```
udpSrc = soc.iosource(hw,'UDP Receive')
```

```
udpSrc =

  soc.iosource.UDPRead with properties:

   Main
              LocalPort: 25000
             DataLength: 1
               DataType: 'uint8'
      ReceiveBufferSize: -1
           BlockingTime: 0
    OutputVarSizeSignal: false
             SampleTime: 0.1000
         HideEventLines: true

  Show all properties
```

Add this UDP source object to the data recording session by using the `addSource` object function.

```
addSource(dr,udpSrc,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
dr.Sources
```

```
ans =

  1×1 cell array

    {'UDPDataOnPort25000'}
```

Call the setup function to initialize all hardware peripheral input sources added to the data recording session, and start the data recording process.

```
setup(dr)
```

Record data for 60 seconds on the SoC hardware board.

```
record(dr, 60);
```

Check the status of the data recording session by using the `isRecording` object function. The recording status when data recording is in progress is `1`.

```
recordingStatus = isRecording(dr)
```

```
recordingStatus =

  logical

   1
```

The recording status when data recording is complete is `0`.

```
isRecording(dr)
```

```
recordingStatus =

  logical

   0
```

Save recorded data to a TGZ-compressed file.

```
save(dr,'UDPDataReceived','UDP Data Testing',{'Recorded On Zynq Board'})
```

This function saves the recorded data as the file `UDPDataReceived.tgz` in your working folder of the host PC. You can read this file by using an `socFileReader` object in MATLAB or an IO Data Source block in your Simulink model.

Remove the added source from the data recording session by using the `removeSource` object function.

```
removeSource(dr,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
ans =

  1×0 empty cell array
```

Version History

Introduced in R2019a

See Also

`socHardwareBoard` | `DataRecorder`

socHardwareBoard

Connection to SoC hardware board

Description

The `socHardwareBoard` object represents a connection to the specified SoC hardware board from MATLAB. Use this object to create `DataRecorder` and `socAXIMaster` objects that record input data and access memory on the specified SoC hardware board.

Creation

Syntax

```
hwList = socHardwareBoard()
hw = socHardwareBoard(boardName)
hw = socHardwareBoard(boardName,Name,Value)
```

Description

`hwList = socHardwareBoard()` returns a list of supported SoC hardware boards.

`hw = socHardwareBoard(boardName)` creates a connection to the specified SoC hardware board. This connection reuses the IP address, username, and password from the most recent connection to that specified SoC hardware board. When you connect MATLAB to an SoC hardware board for the first time, enter the board name, IP address, username, and password of the SoC hardware board as name-value pair arguments.

To see the complete list of supported SoC hardware boards, call the `socHardwareBoard` function without any arguments.

`hw = socHardwareBoard(boardName,Name,Value)` creates a connection to the specified SoC hardware by using the IP address, user name, and password that you specify.

Input Arguments

`boardName` — Name of supported SoC hardware board
character vector | string scalar

Name of supported SoC hardware board, specified as a character vector or string scalar. Specify the name of hardware board to which you want to establish a connection from MATLAB. To get the list of supported hardware boards, call `socHardwareBoard` function without any arguments.

Example: `'Xilinx Zynq ZC706 evaluation kit'`

Name-Value Pair Arguments

Specify optional pairs of arguments as `Name1=Value1,...,NameN=ValueN`, where `Name` is the argument name and `Value` is the corresponding value. Name-value arguments must appear after other arguments, but the order of the pairs does not matter.

Before R2021a, use commas to separate each name and value, and enclose `Name` *in quotes.*

Example: `'username','root'`

`hostname` — IP address of SoC hardware board
character vector | string scalar

IP address of the SoC hardware board connected to the network, specified as the comma-separated pair consisting of `'hostname'` and a character vector or string scalar.

Example: '192.168.1.18'

Data Types: `char` | `string`

`username` — Root username used to log into SoC hardware board
character vector | string scalar

Root username used to log in into SoC hardware board connected to the network, specified as the comma-separated pair consisting of `'username'` and a character vector or string scalar.

Example: 'root'

Data Types: `char` | `string`

`password` — Root password used to log into SoC hardware board
character vector | string scalar

Root password used to log in into SoC hardware board connected to the network, specified as the comma-separated pair consisting of `'password'` and a character vector or string scalar.

Example: 'password'

Data Types: `char` | `string`

Output Arguments

`hwList` — List of supported SoC hardware boards
string array

List of SoC hardware boards that are supported for data logging returned as a string array.

`hw` — Connection to specific SoC hardware board
`socHardwareBoard` object

Connection to specific SoC hardware board, returned as a `socHardwareBoard` object. You can use this connection for data logging of input sources with the `DataRecorder` object, or you can access memory on the board using an `socAXIMaster` object.

Properties

`BoardName` — Name of supported SoC hardware board
character array | string scalar

This property cannot be changed after you create the `socHardwareBoard` object.

Name of supported SoC hardware board, specified as a character array or string scalar.

Example: `'Xilinx Zynq ZC706 evaluation kit'`

Data Types: `char` | `string`

`DeviceAddress` — IP address of SoC hardware board
character array | string scalar

This property cannot be changed after you create the `socHardwareBoard` object.

IP address of SoC hardware board, specified as a character array or string scalar.

Example: `'192.168.1.11'`

Data Types: `char` | `string`

`Port` — IP port number of SoC hardware board
integer from 1 to 65,535

This property cannot be changed.

IP port number of SoC hardware board.

Example: `18735`

Data Types: `double`

Examples

Record Data From SoC Hardware Board

Create a connection from MATLAB to the specified SoC hardware board using the IP address, username, and password of the board.

```
hw = socHardwareBoard('Xilinx Zynq ZC706 evaluation kit','hostname','192.168.1.18','username','ro
```

Create a data recording session on the SoC hardware board by using the `hw` object. The resulting `DataRecorder` object represents the data recording session on the SoC hardware board.

```
dr = soc.recorder(hw)
```

```
dr =

  DataRecorder with properties:

         HardwareName: 'Xilinx Zynq ZC706 evaluation kit'
              Sources: {}
            Recording: false
```

List the input sources added to the data recording session.

```
dr.Sources(hw)
```

```
ans =

  1×0 empty cell array
```

By default, `soc.recorder` objects have no added input sources. To add an input source to the data recording session, first create an input source object by using the `soc.iosource` function. For this example, create an User Datagram Protocol (UDP) source object.

```
udpSrc = soc.iosource(hw,'UDP Receive')
```

```
udpSrc =

  soc.iosource.UDPRead with properties:

   Main
              LocalPort: 25000
             DataLength: 1
               DataType: 'uint8'
      ReceiveBufferSize: -1
           BlockingTime: 0
    OutputVarSizeSignal: false
             SampleTime: 0.1000
         HideEventLines: true

  Show all properties
```

Add this UDP source object to the data recording session by using the `addSource` object function.

```
addSource(dr,udpSrc,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
dr.Sources
```

```
ans =

  1×1 cell array

    {'UDPDataOnPort25000'}
```

Call the setup function to initialize all hardware peripheral input sources added to the data recording session, and start the data recording process.

```
setup(dr)
```

Record data for 60 seconds on the SoC hardware board.

```
record(dr, 60);
```

Check the status of the data recording session by using the `isRecording` object function. The recording status when data recording is in progress is `1`.

```
recordingStatus = isRecording(dr)
```

```
recordingStatus =

  logical

   1
```

The recording status when data recording is complete is `0`.

```
isRecording(dr)
```

```
recordingStatus =

  logical
```

```
   0
```

Save recorded data to a TGZ-compressed file.

```
save(dr,'UDPDataReceived','UDP Data Testing',{'Recorded On Zynq Board'})
```

This function saves the recorded data as the file `UDPDataReceived.tgz` in your working folder of the host PC. You can read this file by using an `socFileReader` object in MATLAB or an IO Data Source block in your Simulink model.

Remove the added source from the data recording session by using the `removeSource` object function.

```
removeSource(dr,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
ans =

  1×0 empty cell array
```

Initialize Memory on SoC Hardware Board from MATLAB

For an example of how to configure and use the AXI manager IP in your design, see “Random Access of External Memory”. Specifically, review the `soc_image_rotation_axi_master.m` script that initializes the memory on the device, starts the FPGA logic, and reads back the modified data. This example shows only the memory initialization step.

Load a `.mat` file that contains structures derived from the board configuration parameters. This file was generated by **SoC Builder**. These structures also describe the IP cores and memory configuration of the design on the board. Set up a JTAG AXI manager connection by creating a `socHardwareBoard` object and passing it to the `socAXIManager` object. The `socAXIManager` object connects with the hardware board and confirms that the IP is present.

```
load('soc_image_rotation_zc706_aximaster.mat');
hwObj = socHardwareBoard('Xilinx Zynq ZC706 evaluation kit','Connect',false);
AXIManagerObj = socAXIManager(hwObj);
```

Initialize the memory contents on the device by loading the figure data and writing it to `Region1`. The FPGA logic is designed to read this data, rotate it, and write it into `Region2`. Clear the contents of `Region2`.

```
load('soc_image_rotation_inputdata.mat');
inputFigure = smallImage;
[x, y] = size(inputFigure);
inputImage = uint32(reshape(inputFigure',1,x*y));
writememory(AXIManagerObj,memRegions.AXI4MasterMemRegion1,inputImage);
writememory(AXIManagerObj,memRegions.AXI4MasterMemRegion2,uint32(zeros(1,x*y)));
```

Version History

Introduced in R2019a

See Also

`DataRecorder` | `soc.iosource` | `socAXIMaster`

DataRecorder

Data recording session for specified SoC hardware board

Description

A `DataRecorder` object can configure and log data from input sources on an SoC hardware board connected to MATLAB. You can save the recorded data to a file for future use to playback in MATLAB and Simulink models.

Creation

Create a `DataRecorder` object using `soc.recorder` function.

Properties

`HardwareName` — Name of supported SoC hardware board
character vector

Name of supported SoC hardware board, specified as a character vector.

Data Types: `char`

`Sources` — List of hardware-peripheral input sources
cell array

List of hardware-peripheral input sources added to data recording session, specified a character vector. To add input sources to a `soc.recorder` object, call the `addSource` object function.

Data Types: `cell`

`Recording` — Status of data recording session
`false` (`0`) | `true` (`1`)

This property is read-only.

Status of data recording session, specified as a logic value of `false` (`0`) or `true` (`1`). To get the status of the data recording session, call the `isRecording` object function.

Data Types: `logical`

Object Functions

addSource	Add a input source to a data recording session
removeSource	Remove input source from data recording session
setup	Set up hardware for data recording
record	Record data from hardware using data recorder object
isRecording	Get data recording status
save	Save recorded data from SoC hardware board to file on host PC

Examples

Record Data From SoC Hardware Board

Create a connection from MATLAB to the specified SoC hardware board using the IP address, username, and password of the board.

```
hw = socHardwareBoard('Xilinx Zynq ZC706 evaluation kit','hostname','192.168.1.18','username','ro
```

Create a data recording session on the SoC hardware board by using the `hw` object. The resulting `DataRecorder` object represents the data recording session on the SoC hardware board.

```
dr = soc.recorder(hw)
```

```
dr =

  DataRecorder with properties:

          HardwareName: 'Xilinx Zynq ZC706 evaluation kit'
               Sources: {}
             Recording: false
```

List the input sources added to the data recording session.

```
dr.Sources(hw)
```

```
ans =

  1×0 empty cell array
```

By default, `soc.recorder` objects have no added input sources. To add an input source to the data recording session, first create an input source object by using the `soc.iosource` function. For this example, create an User Datagram Protocol (UDP) source object.

```
udpSrc = soc.iosource(hw,'UDP Receive')
```

```
udpSrc =

  soc.iosource.UDPRead with properties:

   Main
              LocalPort: 25000
             DataLength: 1
               DataType: 'uint8'
      ReceiveBufferSize: -1
           BlockingTime: 0
    OutputVarSizeSignal: false
             SampleTime: 0.1000
         HideEventLines: true

  Show all properties
```

Add this UDP source object to the data recording session by using the `addSource` object function.

```
addSource(dr,udpSrc,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
dr.Sources

ans =

  1×1 cell array

    {'UDPDataOnPort25000'}
```

Call the setup function to initialize all hardware peripheral input sources added to the data recording session, and start the data recording process.

```
setup(dr)
```

Record data for 60 seconds on the SoC hardware board.

```
record(dr, 60);
```

Check the status of the data recording session by using the `isRecording` object function. The recording status when data recording is in progress is `1`.

```
recordingStatus = isRecording(dr)

recordingStatus =

  logical

   1
```

The recording status when data recording is complete is `0`.

```
isRecording(dr)

recordingStatus =

  logical

   0
```

Save recorded data to a TGZ-compressed file.

```
save(dr,'UDPDataReceived','UDP Data Testing',{'Recorded On Zynq Board'})
```

This function saves the recorded data as the file `UDPDataReceived.tgz` in your working folder of the host PC. You can read this file by using an `socFileReader` object in MATLAB or an IO Data Source block in your Simulink model.

Remove the added source from the data recording session by using the `removeSource` object function.

```
removeSource(dr,'UDPDataReceived-Port25000')
```

Verify the result by inspecting the `Sources` property of the `soc.recorder` object.

```
ans =

  1×0 empty cell array
```

Version History

Introduced in R2019a

See Also

socHardwareBoard | socFileReader | soc.iosource

socFileReader

File reader

Description

The `socFileReader` object is a file reader that reads data from a specified TGZ-compressed file and stores the data sets in the object. The data set contains information about the source objects that represent recorded data sources from the specified TGZ-compressed file. The TGZ file format is created by a previous recording session on an SoC hardware board.

Creation

Syntax

```
fr = socFileReader(filename)
```

Description

`fr = socFileReader(filename)` creates an object, `fr`, from the specified file. The object is a file reader that reads data from a specified TGZ-compressed file and stores the data sets in the object. The `filename` must be a file saved using the `save` function of a `DataRecorder` object.

Input Arguments

`filename` — File from previous data recording session
character vector

File from a previous data recording session on SoC hardware board, specified as a character vector with a `tgz` extension.

Example: 'UDPDataReceived.tgz'

Properties

`Description` — User metadata describing data set
character vector

User meta data describing the data set, specified as a character vector. This value is added to the file when you call the `save` object function.

Data Types: `char`

`HardwareBoard` — Name of SoC hardware board
character vector

Name of the SoC hardware board used for data collection in the `DataRecorder` object, specified as a character vector.

Data Types: `char`

`Tags` — User tags
cell array

User tags, specified as a cell array. This value is added to the file when you call the `save` object function.

Data Types: `cell`

`Filename` — Name of recorded data file
character vector

Name of recorded data file, specified as a character vector. This value represents the file name of a file saved using the `save` object function.

Data Types: `char`

`Sources` — List of sources in data set
cell array

List of sources in data set file, returned as a cell array.

Data Types: `cell`

`Date` — Date of data set creation
character vector

Date of data set creation, returned as a character vector.

Data Types: `char` | `string`

Object Functions
getData Get data from file reader

Examples

Create File Reader Object

Create a file reader to read data from the specified TGZ-compressed file.

```
fr = socFileReader('UDPDataReceived.tgz')
```

```
fr =

  socFileReader with properties:

      Description: ''
    HardwareBoard: 'Xilinx Zynq ZC706 evaluation kit'
             Tags: {}
         Filename: 'H:\UDPDataReceived.tgz'
          Sources: {'UDPDataOnPort25000'}
             Date: 28-Dec-2018 15:17:08
```

Get the data of a specified source from the file using the `getData` function.

```
rd = getData(fr,'UDPDataReceived-Port25000');
```

Version History

Introduced in R2019a

See Also

DataRecorder | save

socIPCore

Create object to represent IP core running on FPGA board

Description

The `socIPCore` object represents an active IP core on an FPGA board and provides read and write access to the IP.

Creation

Syntax

```
myCoreObj = socIPCore(axiMaster,IPCoreInfo,IPCoreName)
myCoreObj = socIPCore(axiMaster,IPCoreInfo,IPCoreName,Name,Value)
```

Description

`myCoreObj = socIPCore(axiMaster,IPCoreInfo,IPCoreName)` creates an `socIPCore` object that connects to an IP core running on an FPGA board. The object uses an `socAXIMaster` object to access memory locations in the IP core. `IPCoreInfo` is a structure generated when you run the **SoC Builder** tool and includes the board and IP core configuration parameters from your model.

You can create `socIPCore` objects representing any of these IPs:

- Traffic generator
- Performance monitor
- Direct memory access (DMA)
- Video DMA (VDMA)
- Video timing controller (VTC)
- VDMA trigger
- Frame buffer
- High definition multimedia interface (HDMI)

`myCoreObj = socIPCore(axiMaster,IPCoreInfo,IPCoreName,Name,Value)` sets properties using one or more name-value pairs. For example,

```
myIPobj=socIPCore(axiMaster, perf_mon,'PerformanceMonitor','Mode','Profile');
```

creates an `socIPCore` object that connects to an IP core on the specified board and sets the performance monitor mode to profile mode.

Input Arguments

`axiMaster` — Name of `socAXIMaster` object used for memory-mapped access
`socAXIMaster` object

Name of `socAXIMaster` object used for memory-mapped access, specified as an `socAXIMaster` object.

Create an `socAXIMaster` object using the `socAXIMaster` function, and use the created object as an input to `socIPCore`.

Example: `mySocAXIObj = socAXIMaster('Xilinx'); myIPObj = socIPCore(mySocAXIObj,IPCoreInfo,'DMA')`

`IPCoreInfo` — IP core information
structure

IP core information, specified as a structure generated by the **SoC Builder** tool. To access the structure, load the `.mat` file which is generated by **SoC Builder** tool. The file is named *`model_name_boardID`*`_aximaster.mat`. Loading the file will load the structures generated by the **SoC Builder** tool to your workspace.

The structures contain information for vendor IP and for user-specified IP which are specific to your model and board. The structures are named as follows:

- `vdma_frame_buffer` – A `struct` representing a frame buffer.
- `perf_mon` – A `struct` representing a performance monitor.
- `vtc` – A `struct` representing a video timing controller.
- `vdma_hdmi_out` – A `struct` representing a VDMA-based HDMI IP.
- `atg` – A `struct` representing an AXI traffic generator.
- *`DUT`*`_ip` – A `struct` representing a user IP named "DUT".

Note The `mat` file loads additional `structs` for IPs, for internal access.

`IPCoreName` — IP core object type
`'TrafficGenerator'` | `'PerformanceMonitor'` | `'VDMA'` | `'DMA'` | `'VDMATrigger'` | `'VTC'` | `'FrameBuffer'` | `'HDMI'`

IP core object type, specified as one of the values in this table:

Value	Description
`'TrafficGenerator'`	SoC Blockset memory traffic generator
`'PerformanceMonitor'`	SoC Blockset performance monitor
`'VDMA'`	Xilinx VDMA IP
`'DMA'`	Analog Devices® DMA controller IP
`'VTC'`	Video timing controller
`'VDMATrigger'`	An IP used to trigger reading frames from the source (mm2s) VDMA
`'FrameBuffer'`	VDMA-based frame buffer IP
`'HDMI'`	VDMA-based HDMI IP

Data Types: `string` | `character vector`

Properties

`PerfMonMode` — Type of performance data to collect
`'Profile'` (default) | `'Trace'`

Type of performance data to collect, specified as `'Profile'` or `'Trace'`. Specify `'Profile'` mode to collect byte and burst counts for bandwidth and latency plots. `'Trace'` mode to collect burst transaction event data for display as waveforms.

Object Functions

initialize	Initialize IP core corresponding to socIPCore object
start	Start IP core execution on hardware board

Version History

Introduced in R2019a

See Also

`socAXIMaster`

Topics

“Analyze Memory Bandwidth Using Traffic Generators”

socAXIManager

Read and write memory locations on hardware board from MATLAB

Description

The `socAXIManager` object communicates with the MATLAB AXI manager IP running on a hardware board. The object uses a JTAG connection to forward read and write commands to the IP and access slave memory locations on the hardware board. Pass an `socAXIManager` object as an argument when you create an `socIPCore` object, so that the object can access memory locations within the IP core on the board.

Creation

Description

`axiManagerObj = socAXIManager(vendor)` creates an object that connects to an AXI manager IP for the specified `vendor`.This connection enables you to access memory locations in an SoC design from MATLAB.

`axiManagerObj = socAXIManager(hw)` creates an object that connects to an AXI manager IP on the specified hardware board.

`axiManagerObj = socAXIManager( ___ ,Name,Value)` creates an object with additional properties specified by one or more `Name,Value` pair arguments. Enclose each property name in quotes. Specify properties in addition to the input arguments in previous syntaxes.

Input Arguments

`vendor` — FPGA brand name

`'Intel'` | `'Xilinx'`

FPGA brand name, specified as `'Intel'` or `'Xilinx'`. The AXI manager IP varies depending on the type of FPGA you have.

`hw` — Hardware object

`socHardwareBoard` object

Hardware object, specified as a `socHardwareBoard` object that represents the connection to the SoC hardware board.

Properties

`JTAGCableType` — Type of JTAG cable used for communication with FPGA board (Xilinx boards only)

`'auto'` (default) | `'FTDI'`

Type of JTAG cable used for communication with the FPGA board (Xilinx boards only), specified as `'auto'` or `'FTDI'`. This property is most useful when more than one cable is connected to the host computer.

When this property is set to `'auto'` (default), the object autodetects the JTAG cable type. The object prioritizes searching for Digilent® cables and uses this process to autodetect the cable type.

1. The `socAXIManager` object searches for a Digilent cable. If the object finds:
   - Exactly one Digilent cable –– The object uses that cable for communication with the FPGA board.
   - More than one Digilent cable –– The object returns an error. To resolve this error, specify the desired cable using the `JTAGCableName` property.
   - No Digilent cables –– The object searches for an FTDI cable (see step 2).
2. If no Digilent cable is found, the `socAXIManager` object searches for an FTDI cable. If the object finds:
   - Exactly one FTDI cable –– The object uses that cable for communication with the FPGA board.
   - More than one FTDI cable –– The object returns an error. To resolve this error, specify the desired cable using the `JTAGCableName` property.
   - No FTDI cables –– The object returns an error. To resolve this error, connect a Digilent or FTDI cable.

The cable search in `'auto'` mode prioritizes connection using a Digilent cable. If one Digilent and one FTDI cable are connected to the host computer and this property is set to `'auto'`, the object selects the Digilent cable for communication with the FPGA board.

When this property is set to `'FTDI'`, the object searches for FTDI cables. If the object finds:

- Exactly one FTDI cable –– The object uses that cable for communication with the FPGA board.
- More than one FTDI cable –– The object returns an error. To resolve this error, specify the desired cable using the `JTAGCableName` property.
- No FTDI cables –– The object returns an error. To resolve this error, connect a Digilent or FTDI cable.

For an example, see “Select from Multiple JTAG Cables” on page 4-28.

`JTAGCableName` — Name of JTAG cable used for communication with FPGA board

`'auto'` (default) | character vector

Name of JTAG cable user for communication with FPGA board, specified as `'auto'` or a character vector. Specify this property if more than one JTAG cable of the same type are connected to the host computer. If the host computer has more than one JTAG cable and you do not specify this property, the object returns an error. The error message contains the names of the available JTAG cables. For an example, see “Select from Multiple JTAG Cables” on page 4-28.

`TckFrequency` — JTAG clock frequency

15 (default) | positive integer

JTAG clock frequency, in MHz, specified as a positive integer. For Intel FPGAs the JTAG clock frequency must be 12 MHz or 24 MHz. For Xilinx FPGAs, the JTAG clock frequency must be 33 MHz or 66 MHz. The JTAG clock frequency depends on the type of cable and the maximum clock frequency supported by the FPGA board.

`JTAGChainPosition` — Position of FPGA in JTAG chain (Xilinx boards only)

1 (default) | positive integer

Position of FPGA in JTAG chain (Xilinx boards only), specified as a positive integer. Specify this property value if more than one FPGA or Zynq device is on the JTAG chain.

`IRLengthBefore` — Sum of instruction register length for all devices before target FPGA (Xilinx boards only)
`0` (default) | nonnegative integer

Sum of instruction register length for all devices before target FPGA (Xilinx boards only), specified as a nonnegative integer. Specify this property value if more than one FPGA or Zynq device is on the JTAG chain.

`IRLengthAfter` — Sum of instruction register length for all devices after target FPGA (Xilinx boards only)
`0` (default) | nonnegative integer

Sum of instruction register length for all devices after target FPGA (Xilinx boards only), specified as a nonnegative integer. Specify this property value if more than one FPGA or Zynq device is on the JTAG chain.

Object Functions

readmemory	Read data from AXI4 memory-mapped locations
release	Release JTAG cable resource
writememory	Write data to AXI4 memory-mapped locations

Examples

Initialize Memory on SoC Hardware Board from MATLAB

For an example of how to configure and use the AXI manager IP in your design, see “Random Access of External Memory”. Specifically, review the `soc_image_rotation_axi_master.m` script that initializes the memory on the device, starts the FPGA logic, and reads back the modified data. This example shows only the memory initialization step.

Load a `.mat` file that contains structures derived from the board configuration parameters. This file was generated by **SoC Builder**. These structures also describe the IP cores and memory configuration of the design on the board. Set up a JTAG AXI manager connection by creating a `socHardwareBoard` object and passing it to the `socAXIManager` object. The `socAXIManager` object connects with the hardware board and confirms that the IP is present.

```
load('soc_image_rotation_zc706_aximaster.mat');
hwObj = socHardwareBoard('Xilinx Zynq ZC706 evaluation kit','Connect',false);
AXIManagerObj = socAXIManager(hwObj);
```

Initialize the memory contents on the device by loading the figure data and writing it to `Region1`. The FPGA logic is designed to read this data, rotate it, and write it into `Region2`. Clear the contents of `Region2`.

```
load('soc_image_rotation_inputdata.mat');
inputFigure = smallImage;
[x, y] = size(inputFigure);
inputImage = uint32(reshape(inputFigure',1,x*y));
```

```
writememory(AXIManagerObj,memRegions.AXI4MasterMemRegion1,inputImage);
writememory(AXIManagerObj,memRegions.AXI4MasterMemRegion2,uint32(zeros(1,x*y)));
```

Access Memory on SoC Hardware Board from MATLAB

For this example, you must have a design running on a hardware board connected to the MATLAB host machine.

Create a MATLAB AXI manager object. The object connects with the hardware board and confirms that the IP is present. You can create the object with a vendor name or an `socHardwareBoard` object.

```
mem = socAXIManager('Xilinx');
```

Write and read one or more addresses with one command. By default, the functions auto-increment the address for each word of data. For instance, write ten addresses, then read the data back from a single location.

```
writememory(mem,140,[10:19])
rd_d = readmemory(mem,140,1)
```

```
rd_d =

  uint32

   10
```

Now, read the written data from ten locations.

```
rd_d = readmemory(mem,140,10)
```

```
rd_d =

  1×10 uint32 row vector

   10   11   12   13   14   15   16   17   18   19
```

Set the `BurstType` property to `'Fixed'` to turn off the auto-increment and access the same address multiple times. For instance, read the written data ten times from the same address.

```
rd_d = readmemory(mem,140,10,'BurstType','Fixed')
```

```
rd_d =

  1×10 uint32 row vector

   10   10   10   10   10   10   10   10   10   10
```

Write incrementing data ten times to the same address. The final value stored in address `140` is `29`.

```
writememory(mem,140,[20:29],'BurstType','Fixed')
rd_d = readmemory(mem,140,10)
```

```
rd_d =

  1×10 uint32 row vector
```

```
   29   11   12   13   14   15   16   17   18   19
```

Alternatively, specify the address as a hexadecimal string. To cast the read data to a data type other than `uint32`, use the `OutputDataType` property.

```
writememory(mem,'1c',[0:4:64])
rd_d = readmemory(mem,'1c',16,'OutputDataType',numerictype(0,6,4))
```

```
rd_d =

  Columns 1 through 10
         0    0.2500    0.5000    0.7500    1.0000    1.2500    1.5000    1.7500    2.0000    2.2
  Columns 11 through 16
    2.5000    2.7500    3.0000    3.2500    3.5000    3.7500

          DataTypeMode: Fixed-point: binary point scaling
            Signedness: Unsigned
            WordLength: 6
        FractionLength: 4
```

When you are done accessing the board, release the JTAG connection.

```
release(mem)
```

Select from Multiple JTAG Cables

When multiple JTAG cables are connected to your host computer, the object prioritizes Digilent cables over FTDI cables. To use an FTDI cable, specify the JTAG cable type property.

```
h = socAXIManager('Xilinx', 'JTAGCableType', 'FTDI')
```

If two cables of the same type are connected to your host computer, specify the `JTAGCableName` property for the board where the JTAG manager IP is running. To see the JTAG cable identifiers, attempt to create an `socAXIManager` object, which, in this case, errors and returns a list of the current JTAG cable names.

```
h = socAXIManager('Xilinx')
```

```
Error using fpgadebug_mex
Found more than one JTAG cable:
0 (JtagSmt1): #tpt_0001#ptc_0002#210203991642
1 (Arty): #tpt_0001#ptc_0002#210319789795
Please disconnect the extra cable, or specify the cable name as an input argument.
See documentation of FPGA Data Capture or MATLAB as AXI manager to learn how to set
the cable name.
```

To communicate with this Arty board, specify the matching JTAG cable name.

```
h = socAXIManager('Xilinx','JTAGCableName','#tpt_0001#ptc_0002#210319789795')
```

Version History

Introduced in R2019a

See Also

`socIPCore`

Topics

“Random Access of External Memory”

socMemoryProfiler

Retrieve and display memory performance data

Description

This object collects and displays two types of memory performance data from an AXI memory interconnect IP running on your SoC hardware board. You can collect average transaction latency and counts of bytes and bursts and then plot bandwidth, burst counts, and transaction latency, or collect detailed memory transaction event data and view the data as waveforms.

Creation

Syntax

```
profiler = socMemoryProfiler(hw,performanceMonitor)
```

Description

`profiler = socMemoryProfiler(hw,performanceMonitor)` creates an object that accesses the AXI interconnect monitor IP on the board specified by the `socHardwareBoard` object, `hardware`, and uses the IP configuration from the IP core object, `performanceMonitor`.

Input Arguments

`hw` — Hardware object
`socHardwareBoard` object

Hardware object, specified as a `socHardwareBoard` object that represents the connection to the SoC hardware board.

`performanceMonitor` — AXI interconnect monitor IP core object
`socIPCore` object

AXI interconnect monitor IP core object, specified as an `socIPCore` object that was created with the `IPCoreName` argument set to `'PerformanceMonitor'`, and then initialized. For example,

```
apmCoreObj = socIPCore(AXIMasterObj,perf_mon,'PerformanceMonitor','Mode',perfMonMode);
initialize(apmCoreObj);
```

- `AXIMasterObj` is an `socAXIMaster` object.
- `perf_mon` is a structure generated by the **SoC Builder** tool.
- `perfMonMode` is a string equal to either `'Profile'` or `'Trace'`. `'Profile'` mode collects byte and burst counts for bandwidth and latency plots. `'Trace'` mode collects burst transaction event data for display as waveforms.

Object Functions

collectMemoryStatistics	Retrieve performance data from AXI interconnect monitor
plotMemoryStatistics	Plot performance data obtained from AXI interconnect monitor

Examples

Configure and Query AXI Interconnect Monitor

The AXI interconnect monitor (AIM) is an IP core that collects performance metrics for an AXI-based FPGA design. Create an `socIPCore` object to setup and configure the AIM IP, and use the `socMemoryProfiler` object to retrieve and display the data.

For an example of how to configure and query the AIM IP in your design using MATLAB as AXI Master, see "Analyze Memory Bandwidth Using Traffic Generators". Specifically, review the `soc_memory_traffic_generator_axi_master.m` script that configures and monitors the design on the device.

The performance monitor can collect two types of data. Choose *Profile* mode to collect average transaction latency and counts of bytes and bursts. In this mode, you can launch a performance plot tool, and then configure the tool to plot bandwidth, burst count, and transaction latency. Choose *Trace* mode to collect detailed memory transaction event data and view the data as waveforms.

```
Mode = 'Profile'; % or 'Trace'
```

To obtain diagnostic performance metrics from your generated FPGA design, you must set up a JTAG connection to the device from MATLAB. Load a `.mat` file that contains structures derived from the board configuration parameters. This file was generated by the **SoC Builder** tool. These structures describe the memory interconnect and masters configuration such as buffer sizes and addresses. Use the `socHardwareBoard` object to set up the JTAG connection.

```
load('soc_memory_traffic_generator_zc706_aximaster.mat');
hwObj = socHardwareBoard('Xilinx Zynq ZC706 evaluation kit','Connect',false);
AXIMasterObj = socAXIMaster(hwObj);
```

Configure the AIM. The `socIPCore` object provides a function that performs this initialization. Then, create an `socMemoryProfiler` object to gather the metrics.

```
apmCoreObj = socIPCore(AXIMasterObj,perf_mon,'PerformanceMonitor','Mode',Mode);
initialize(apmCoreObj);
profilerObj = socMemoryProfiler(hwObj,apmCoreObj);
```

Retrieve performance metrics or signal data from a design running on the FPGA by using the `socMemoryProfiler` object functions.

For `'Profile'` mode, call the `collectMemoryStatistics` function in a loop.

```
NumRuns = 100;
for n = 1:NumRuns
    collectMemoryStatistics(profilerObj);
end
```

JTAG design setup time is long relative to FPGA transaction times, and if you have a small number of transactions in your design, they might have already completed by the time you query the monitor. In

this case, the bandwidth plot shows only one sample, and the throughput calculation is not accurate. If this situation occurs, increase the total number of transactions the design executes.

For `'Trace'` mode, call the `collectMemoryStatistics` function once. This function stops the IP from writing transactions into the FIFO in the AXI interconnect monitor IP, although the transactions continue on the interconnect. Set the size of the transaction FIFO, **Trace capture depth**, in the configuration parameters of the model, under **Hardware Implementation** > **Target hardware resources** > **FPGA design (debug)**.

```
collectMemoryStatistics(profilerObj);
```

Visualize the performance data by using the `plotMemoryStatistics` function. In `'Profile'` mode, this function launches a performance plot tool, and you can configure the tool to plot bandwidth, burst count, and average transaction latency. In `'Trace'` mode, this function opens the **Logic Analyzer** tool to view burst transaction event data.

```
plotMemoryStatistics(profilerObj);
```

Version History

Introduced in R2019a

See Also

“Memory Performance Information from FPGA Execution”

Topics

“Analyze Memory Bandwidth Using Traffic Generators”

soc_blockset_profile

Profile for SoC Blockset systems

Description

The `soc_blockset_profile` specifies the set of stereotypes that define the general properties and constraints of an SoC Blockset system, specifically, the properties of the processors and software. For more information on how to use the `soc_blockset_profile` in SoC application design, see “Design SoC Model Using System Composer”.

Creation

The `soc_blockset_profile` is included in the SoC Blockset. To access the profile, load the file `<matlabroot>/toolbox/soc/processor/utilities/soc_blockset_profile.xml`.

Properties

`Stereotypes` — Stereotypes
array of stereotype objects

The set of stereotypes specific to an SoC Blockset software system. The stereotypes include `ProcessorCore`, `PeriodicSoftwareTask`, and `AperiodicSoftwareTask`.

Data Types: `char`

Version History

Introduced in R2022b

See Also

`ProcessorCore` | `PeriodicSoftwareTask` | `AperiodicSoftwareTask` | `systemcomposer.profile.Profile` (System Composer) | `systemcomposer.profile.Stereotype` (System Composer)

Topics
“Systems Engineering Approach for SoC Applications”
“Design SoC Model Using System Composer”
“Compose and Analyze Systems Using Architecture Models” (System Composer)

ProcessorCore

Stereotype for a processor core

Description

The `ProcessorCore` stereotype allows the assignment of a component in a System Composer architecture model with the properties, specification, and constraints of a processor core.

Creation

The `ProcessorCore` stereotype is included in `soc_blockset_profile`.

Properties

`CoreNum` — Processor core number
0 (default)

Number of the processor core.

Data Types: `uint16`

Version History

Introduced in R2022b

See Also

`PeriodicSoftwareTask` | `AperiodicSoftwareTask` | `soc_blockset_profile` | `socCreateModel` | `socTaskSchedulability` | `systemcomposer.profile.Profile` (System Composer) | `systemcomposer.profile.Stereotype` (System Composer)

Topics
“Systems Engineering Approach for SoC Applications”
“Design SoC Model Using System Composer”
“Compose and Analyze Systems Using Architecture Models” (System Composer)

PeriodicSoftwareTask

Stereotype for a periodic software task

Description

The `PeriodicSoftwareTask` stereotype allows the assignment of a component in a System Composer architecture model with the properties, specification, and constraints of a software task. The `PeriodicSoftwareTask` derives from an abstract `SoftwareTask` stereotype.

Creation

The `PeriodicSoftwareTask` stereotype is included in `soc_blockset_profile`.

Properties

`Period` — Period of task
1 (default) | positive non-zero number

Specify the period of periodic tasks in seconds.

Data Types: `double`

`MinExecutionTime` — Minimum execution time
1 (default) | positive non-zero number

Specify the minimum execution time of periodic tasks in seconds.

Data Types: `double`

`MeanExecutionTime` — Mean execution time
1 (default) | positive non-zero number

Specify the mean execution time of periodic tasks in seconds.

Data Types: `double`

`MaxExecutionTime` — Maximum execution time
1 (default) | positive non-zero number

Specify the maximum execution time of periodic tasks in seconds.

Data Types: `double`

`ExecutionTimeStd` — Standard deviation of execution time
0 (default) | positive number

Specify the standard deviation of execution time of periodic tasks in seconds.

Data Types: `double`

`CoreAffinity` — Core affinity
0 (default) | integer number

Specify the core affinity of periodic tasks (-1 indicates an unassigned task).

Data Types: `double`

Version History

Introduced in R2022b

See Also

`ProcessorCore` | `AperiodicSoftwareTask` | `soc_blockset_profile` | `socCreateModel` | `socTaskSchedulability`

Topics

“Systems Engineering Approach for SoC Applications”
“Design SoC Model Using System Composer”
“Compose and Analyze Systems Using Architecture Models” (System Composer)

AperiodicSoftwareTask

Stereotype for an aperiodic software task

Description

The `AperiodicSoftwareTask` stereotype allows the assignment of a component in a System Composer architecture model with the properties, specification, and constraints of an aperiodic software task. The `AperiodicSoftwareTask` derives from an abstract `SoftwareTask` stereotype.

Creation

The `AperiodicSoftwareTask` stereotype is included in `soc_blockset_profile`.

Properties

`MinRepetitionTime` — Minimum repetition time
1 (default) | positive non-zero number

Specify the minimum repetition of aperiodic tasks in seconds.

Data Types: `double`

`MinExecutionTime` — Minimum execution time
1 (default) | positive non-zero number

Specify the minimum execution time of aperiodic tasks in seconds.

Data Types: `double`

`MeanExecutionTime` — Mean execution time
1 (default) | positive non-zero number

Specify the mean execution time of aperiodic tasks in seconds.

Data Types: `double`

`MaxExecutionTime` — Maximum execution time
1 (default) | positive non-zero number

Specify the maximum execution time of aperiodic tasks in seconds.

Data Types: `double`

`ExecutionTimeStd` — Standard deviation of execution time
0 (default) | positive non-zero number

Specify the standard deviation of execution time of aperiodic tasks in seconds.

Data Types: `double`

`CoreAffinity` — Core affinity
0 (default) | integer number

Specify core affinity of the periodic tasks (-1 indicates an unassigned task).

Data Types: `double`

Version History

Introduced in R2022b

See Also

`ProcessorCore` | `PeriodicSoftwareTask` | `soc_blockset_profile` | `socCreateModel` | `socTaskSchedulability`

Topics

“Systems Engineering Approach for SoC Applications”
“Design SoC Model Using System Composer”
“Compose and Analyze Systems Using Architecture Models” (System Composer)

socModelBuilder

Build, load, and execute SoC model on SoC and FPGA boards

Description

The `socModelBuilder` object controls and runs the steps for building and executing an SoC model on an SoC or FPGA board. Use this object to load existing binaries onto the hardware board.

Creation

Syntax

```
obj = socModelBuilder(modelName)
obj = socModelBuilder(modelName,Name=Value)
obj = socModelBuilder
```

Description

`obj = socModelBuilder(modelName)` creates an object that you can use to build the model with the name `modelName`.

`obj = socModelBuilder(modelName,Name=Value)` sets properties on page 4-39 using one or more name-value arguments.

`obj = socModelBuilder` sets properties on page 4-39 to their default value.

Input Arguments

`modelName` — Model name
character vector | string scalar

Model name, specified as a character vector or string scalar. You must specify this input as a valid Simulink model name. To update the model name after creating the object, use the `ModelName` property.

Example: `'soc_rfsoc_datacapture'`

Data Types: `char` | `string`

Properties

`ModelName` — Model name
`''` (default) | character vector | string scalar

Model name, specified as a character vector or string scalar. Use this property to update the model name that you specify in the `modelName` input argument.

Example: `'soc_rfsoc_datacapture'`

Data Types: `char` | `string`

`ProjectFolder` — Folder for generated project files
`'soc_prj'` (default) | character vector | string scalar

Folder for the generated project files, specified as a character vector or string scalar. The `buildModel` function places all the generated files, including reports, executables, and bitstream, in this folder.

Example: `'soc_prj'`

Data Types: `char` | `string`

`BuildType` — Model part to build
`'Processor and FPGA'` (default) | `'Processor only'` | `'FPGA only'`

Model part to build, specified as one of these values.

- `'Processor and FPGA'` — Build the processor and FPGA models in your top model.
- `'Processor only'` — Build only the processor model in your top model.
- `'FPGA only'` — Build only the FPGA model in your top model.

Example: `'Processor and FPGA'`

Data Types: `char` | `string`

`ExternalMode` — Option to run generated software model in external mode
`false` (default) | `true`

Option to run the generated software model in external mode, specified as one of these values.

- `true` — Run the generated software model in external mode. Use external mode to tune parameters on the FPGA without rebuilding the FPGA design. You can also log data from the FPGA and display the data on the host computer. For more information about external mode, see "External Mode Simulations for Parameter Tuning, Signal Monitoring, and Code Execution Profiling" (Simulink Coder).
- `false` — Build a standalone application for the processor model.

Example: `false`

Data Types: `logical`

`RunExternalFPGABuild` — Option to run FPGA build process externally
`true` (default) | `false`

Option to run the FPGA build process externally in a Windows® command or Linux shell, specified as one of these values.

- `true` — Run the FPGA build process parallel to MATLAB in an external shell.
- `false` — Keep MATLAB busy until the FPGA build process completes.

Example: `true`

Data Types: `logical`

Object Functions

buildModel Build SoC model

loadBinaries Load existing binaries onto hardware board

Examples

Build and Load SoC Model Using Command Line Interface

Use MATLAB commands to build an SoC model by creating and configuring the `socModelBuilder` object. Load the binaries in the specified project folder onto the Xilinx Zynq UltraScale+ RFSoC ZCU111 Evaluation Kit.

Set the synthesis tool path to point to an installed Vivado Design Suite. When you execute this command, use your own Xilinx Vivado installation path.

```
hdlsetuptoolpath('ToolName','Xilinx Vivado', ...
  'ToolPath','C:\Xilinx\Vivado\2022.1\bin\vivado.bat');
```

Create an `socModelBuilder` object for the `soc_rfsoc_datacapture` SoC model. To run the generated software model in external mode, set `ExternalMode` to `true`.

```
obj = socModelBuilder('soc_rfsoc_datacapture',ExternalMode=true)
```

```
obj =

  socModelBuilder with properties:

              ModelName: 'soc_rfsoc_datacapture'
          ProjectFolder: 'soc_prj'
              BuildType: 'Processor and FPGA'
           ExternalMode: 1
    RunExternalFPGABuild: 1
```

Change the project folder to `G:\R2023a\myProject`.

```
obj.ProjectFolder = 'G:\R2023a\myProject'
```

```
obj =

  socModelBuilder with properties:

              ModelName: 'soc_rfsoc_datacapture'
          ProjectFolder: 'G:\R2023a\myProject'
              BuildType: 'Processor and FPGA'
           ExternalMode: 1
    RunExternalFPGABuild: 1
```

Build the model.

```
buildModel(obj);
```

Load the binaries in the `G:\R2023a\myProject` project folder to the last connected hardware board.

```
loadBinaries(obj);
```

Load the binaries in the `G:\R2023a\myProject` project folder onto the Xilinx Zynq UltraScale+ RFSoC ZCU111 Evaluation Kit.

```
hw = socHardwareBoard('Xilinx Zynq UltraScale+ RFSoC ZCU111 Evaluation Kit');
loadBinaries(obj,hw);
```

Version History

Introduced in R2023a

See Also

SoC Builder

Tools

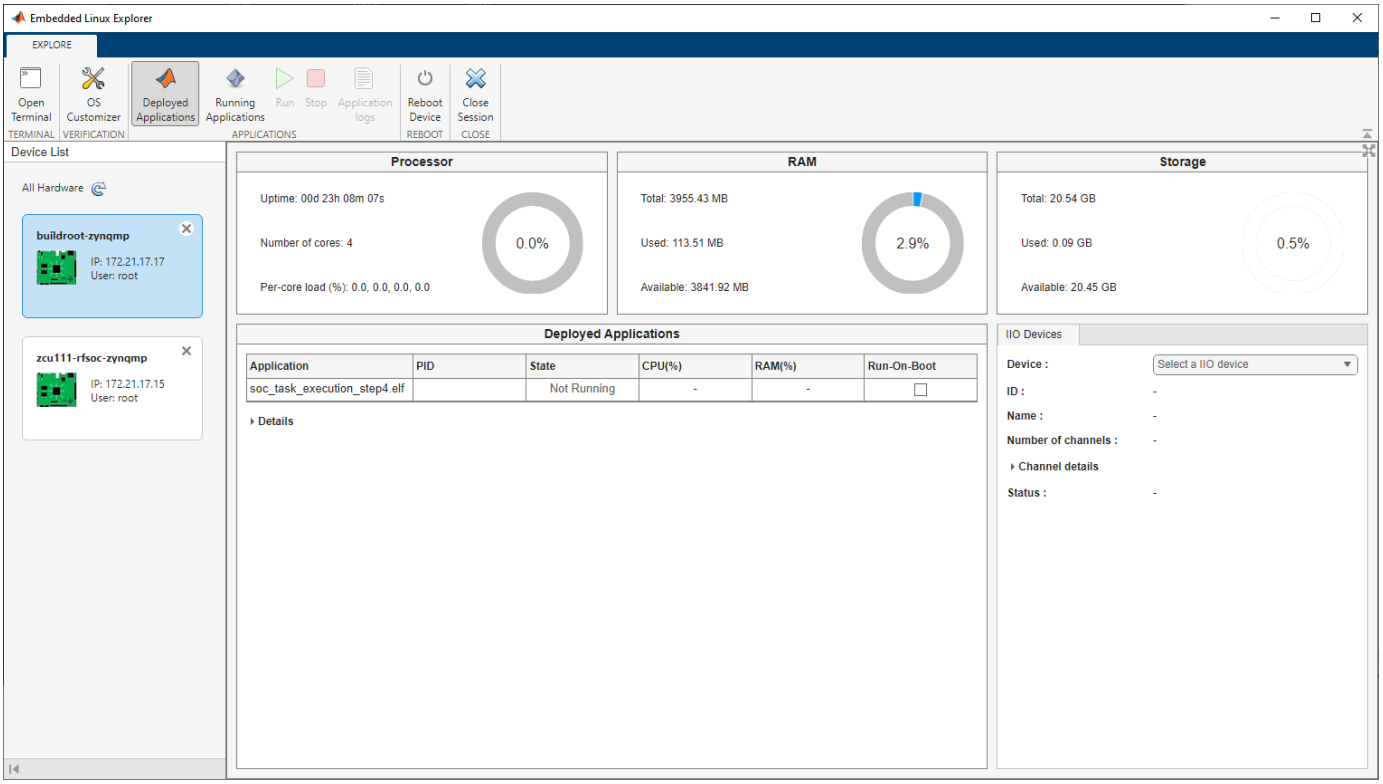
Embedded Linux Explorer

Connect to hardware, monitor resources, and control applications running embedded Linux

Description

The **Embedded Linux Explorer** app provides utilities that help you to develop and deploy embedded Linux applications. With the app, you can:

- Connect to hardware boards running embedded Linux.
- Monitor resource usage.
- Manage processes on the board.
- Troubleshoot issues with resources and processes.



Open the Embedded Linux Explorer App

- MATLAB toolstrip: On the **Apps** tab, scroll to the **Test and Measurement** section, and select **Embedded Linux Explorer**.
- Simulink toolstrip: On the **Apps** tab, select **System on Chip**. Then, in the **Prepare** section, select **Embedded Linux Explorer**.
- MATLAB command line: Enter the following.

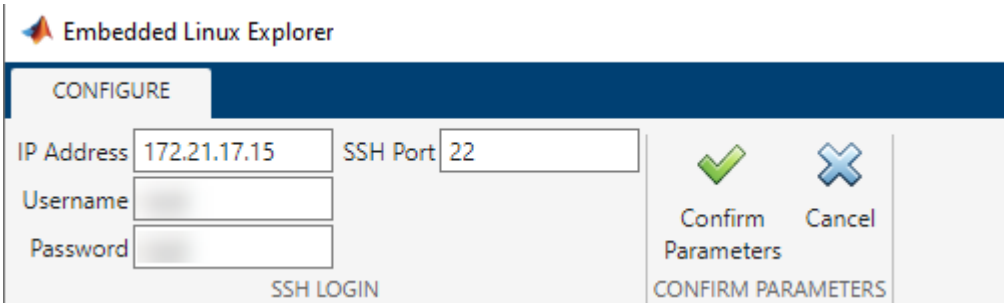
  ```
  embeddedLinuxExplorer
  ```

Examples

Configure SoC Hardware Board

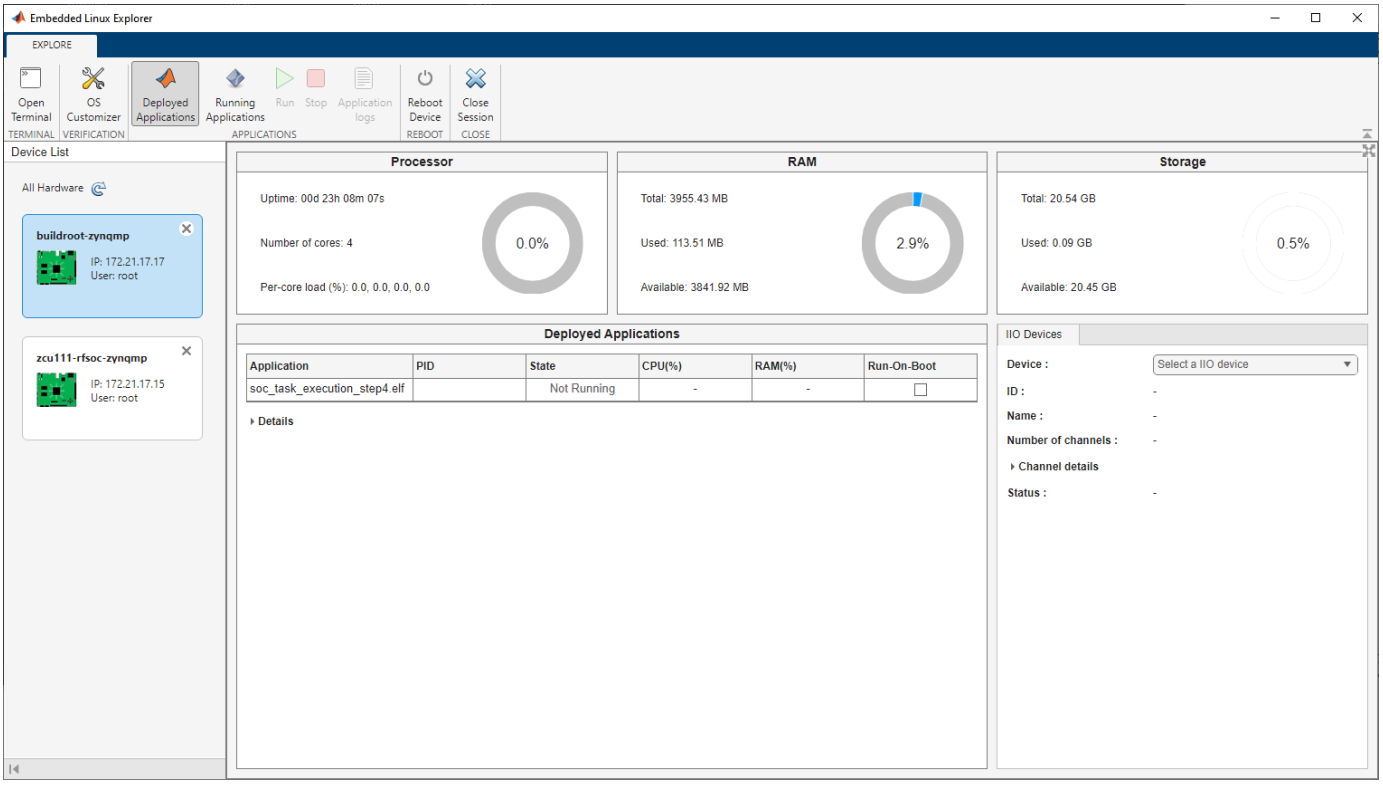
This example shows how to use the **Embedded Linux Explorer** app in SoC Blockset to configure a hardware board on your host computer. For successful communication, ensure that the board and the host computer connect to the same network.

1. Open the **Embedded Linux Explorer** app, and select **Configure SoC Hardware Board**.
2. Enter the IP address, username, password, and SSH port for the board.



3. Click **Confirm Parameters**. You can view the configured board in the **Device List** pane of the app.

Follow the same steps to configure another SoC Blockset board on the host computer. Each configured SoC Blockset board appears in **Device List** pane of the app.



Monitor Deployed Application on SoC Board

This example shows how to use the **Embedded Linux Explorer** app in SoC Blockset to monitor resources for deployed applications.

1. Open **SoC Builder** and load the model from the last stage of the “Task Execution” example.

   ```
   socBuilder('soc_task_execution_step4')
   ```

   a. In the **Prepare** section, on the **Select Build Action** screen, select **Build, load, and run**.
   b. In the **Connect Hardware** screen, enter your login credentials.
   c. In the **Run Application** screen, select **Load and Run**.
2. Open the **Embedded Linux Explorer** app.
3. Enter login credentials that match the device selected in **SoC Builder**.
4. Observe the information about the selected device in the **Deployed Applications** table:

   - The application `soc_task_execution_step4_sw.elf` appears in the table.
   - The state of the application is `Running`.
   - The **CPU (%)** and **RAM (MB)** columns display the total core and RAM percentage occupancy, respectively.
   - A process ID (PID) is assigned to the application.
5. Observe the information in the **Processor** section:

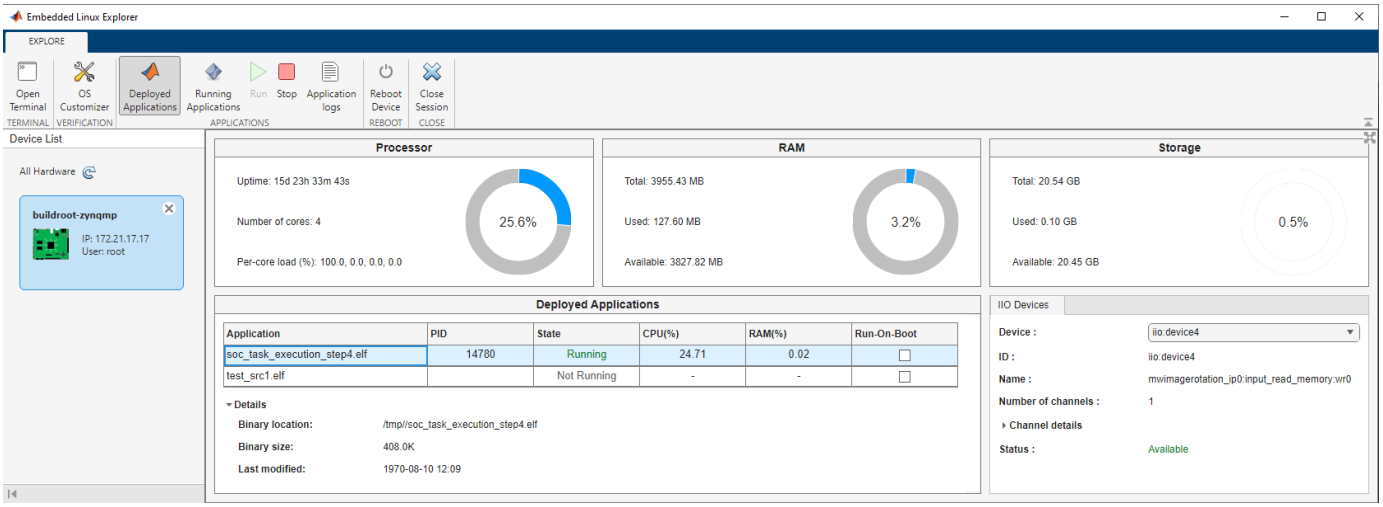
- System uptime
- Number of cores in the processor
- Per-core load: A comma separated list, showing the load percentage per core. In this example, there are four values for four cores.
- A round chart showing the average load per processor.

6 Observe the information in the **RAM** section:

- Total — shows the total processor random access memory (RAM).
- Used — shows the amount of used RAM.
- Available — shows the amount of available RAM.
- A round chart showing RAM usage, as a percentage of total RAM.

7 Observe the information in the **Storage** section:

- Total — shows the total storage memory.
- Used — shows the amount of used storage memory.
- Available — shows the amount of available storage memory.
- A round chart showing storage usage, as a percentage of total storage memory.



To run the application every time the board reboots, select the row for the model in the **Deployed Applications** table, and select **Run-On-Boot**.

Troubleshoot SSH Connections

Error Message	Possible Cause	Solution
Hardware board with address <ip-address> already present.	A device with the same IP address, username, password, and SSH port has already been connected.	In the app, select the appropriate device to connect (instead of creating new connection).

Error Message	Possible Cause	Solution
There was no response to a ping command at address <ip-address>.	• The hardware board is not powered. • The hardware board is not connected to the network. • The host computer does not have access to the network that the hardware board is on.	Ensure that the hardware board has power, is connected to the network, and that the host device and board are visible to each other over the network.
Could not SSH to hardware board at address <ip-address>. Check login credentials and try again.	• The hardware board is not powered. • The hardware board is not connected to the network. • The host computer does not have access to the network that the hardware board is on. • Provided Username, Password and SSH port are incorrect	Ensure that the hardware board has power, is connected to the network, that the host computer and hardware board are visible to each other over the network, and that the SSH credentials are valid.
Unsupported host development platform	Only Windows and Linux hosts are supported.	Change to a supported operating system.
The operating system present on the hardware board with address <ip-address> is not supported.	The hardware board is not running a Linux operating system.	Change to a supported operating system.

Version History

Introduced in R2023a

See Also

Apps
Hardware Manager | SoC Builder

Hardware Mapping

Map tasks and peripherals in a model to hardware board configurations

Description

The **Hardware Mapping** tool allows you to configure the software tasks and peripherals on the selected hardware board.

In this tool, you can map the tasks in your software model to the available event sources and interrupts:

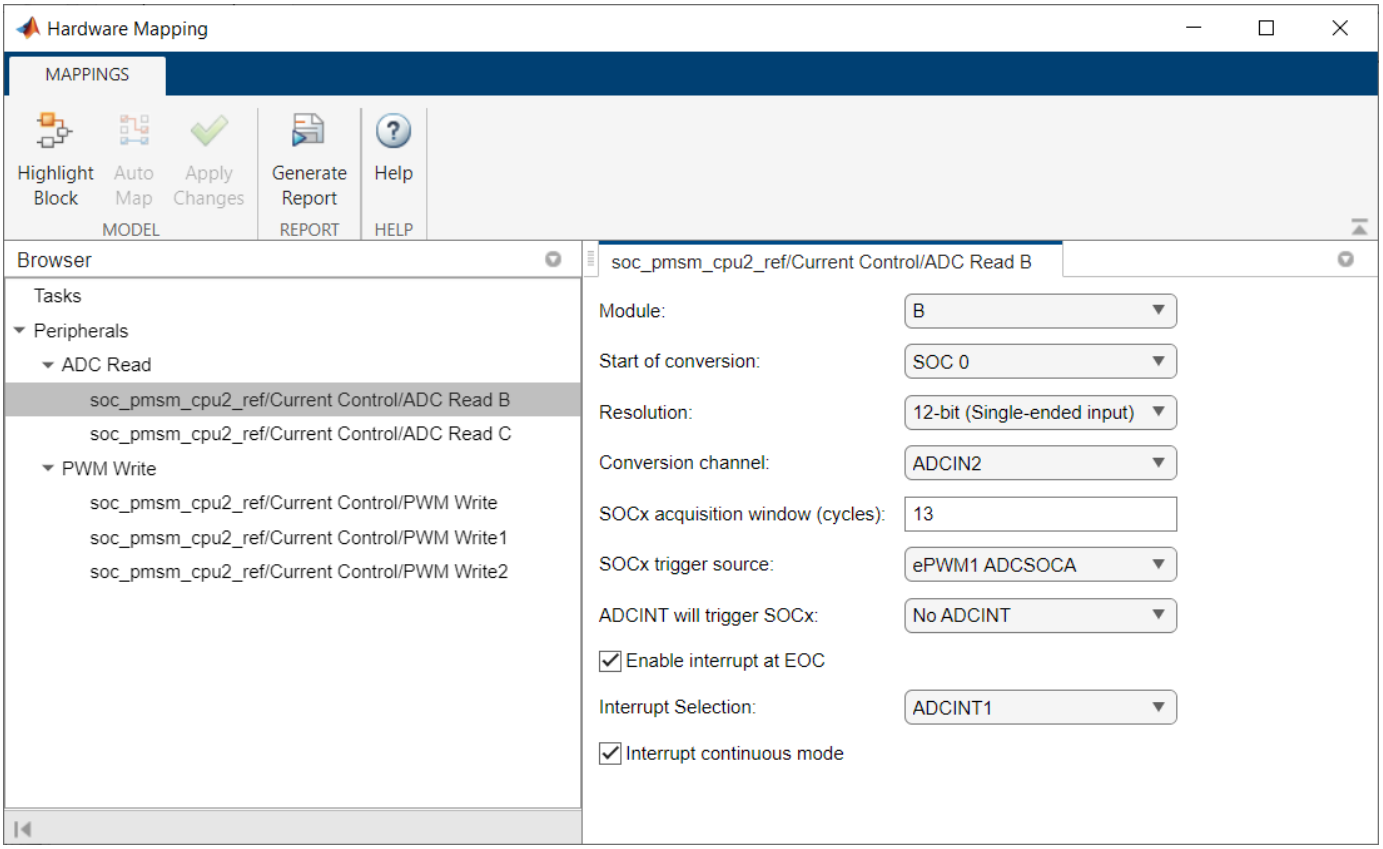
- Manually select the task in **Browser** > **Tasks** > ***`task name`***. Select the desired event or interrupt. source. Click the **Apply Changes** button in the toolstrip.
- Automatically by clicking the **Auto Map** button on the toolstrip.

The sources of events or interrupts depend on the choice of hardware board and peripherals available in the model.

In this tool, you can configure the peripheral by setting hardware specific parameters. To set the peripheral, select the peripheral in **Browser** > **Peripherals** > ***`Peripheral group name`*** > ***`Peripheral block name`***. In the *`Peripheral block name`* pane, choose the appropriate settings to apply to the hardware peripheral when deployed to the hardware board. The available parameters depend on the selected hardware board for the model and the peripheral.

For a complete set of parameters for available hardware boards, see:

- Hardware Mapping Peripherals for Embedded Linux Processors Properties
- Hardware Mapping Peripherals for Texas Instruments C2000 Processors Properties



Open the Hardware Mapping

- A screen within the **SoC Builder** app
- “Hardware Implementation Pane” on page 2-2 for SoC Blockset

Version History

Introduced in R2022b

See Also

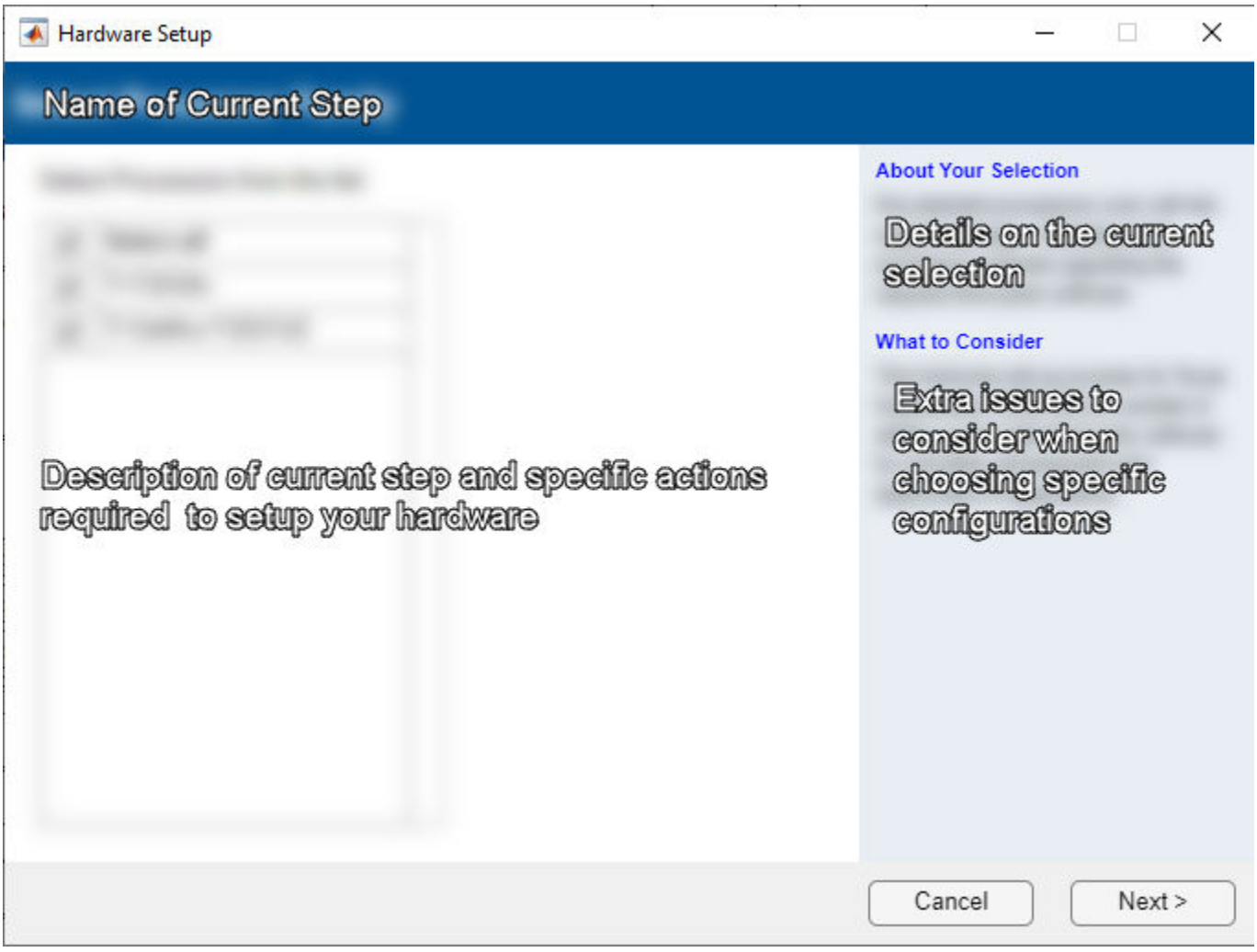
Hardware Mapping Peripherals for Embedded Linux Processors Properties

Hardware Setup

Setup, connect, and configure your hardware boards, development kits, or devices

Description

Hardware boards, development kits, and devices supported by MathWorks require additional configuration and setup steps to connect to MATLAB and Simulink. Each support package provides a hardware setup process that guides you through registering, configuring, and connecting to your hardware board.



Open the Hardware Setup

- In the install window, at the end of the workflow, click the **Setup Now** button.
- After installing the Add-Ons, use **Get and Manage Add-Ons**, at the end of the Installation, click the **Setup** button from Add-Ons Manager

- Simulink Toolstrip: On the **System on Chip** tab, click `Setup Hardware` in **Hardware Board**.

Version History

Introduced in R2022b

Logic Analyzer

Visualize, measure, and analyze transitions and states over time

Description

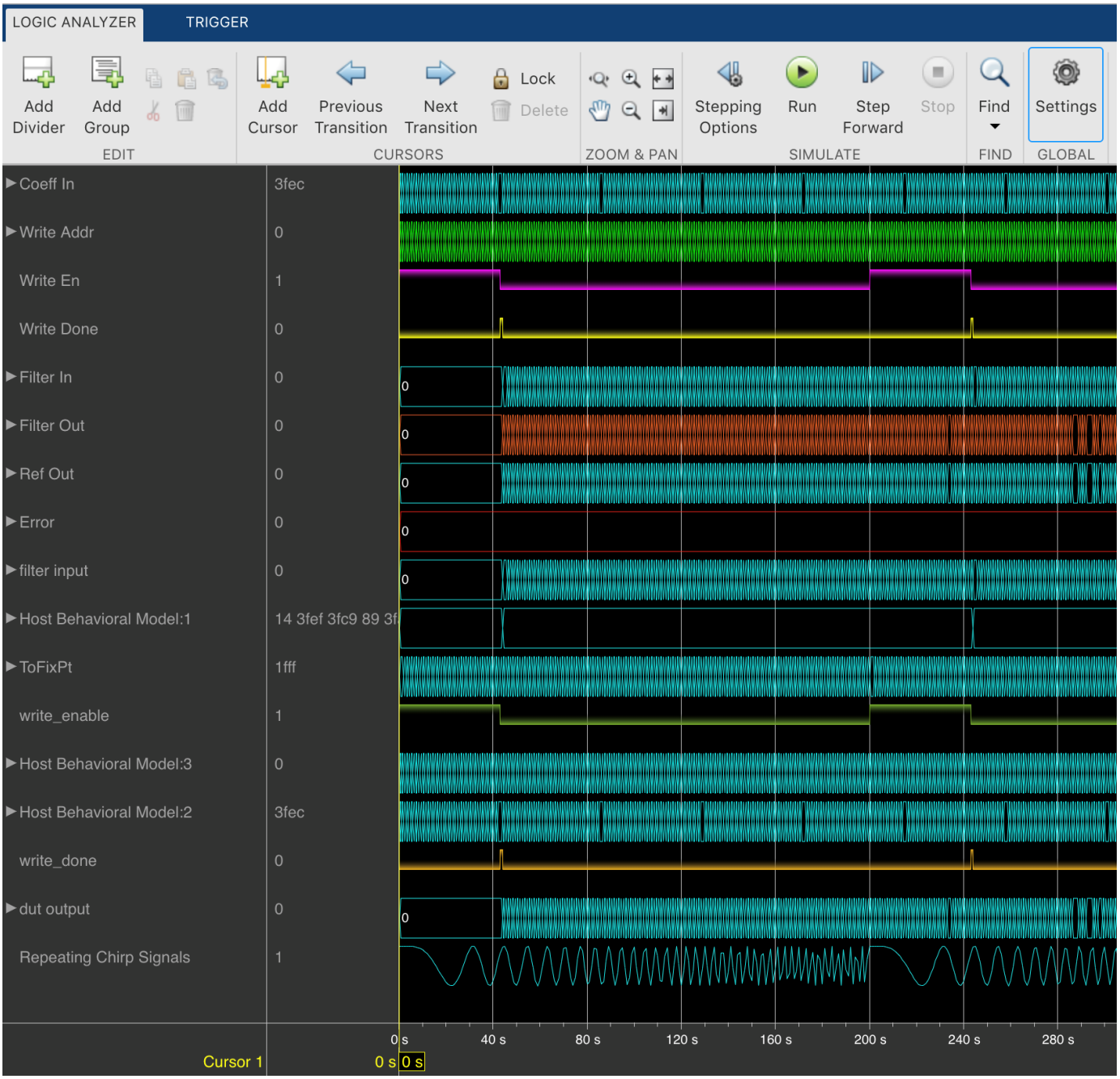
The **Logic Analyzer** is a tool for visualizing and inspecting signals and states in your Simulink model. Using the **Logic Analyzer**, you can:

- Debug and analyze models
- Trace and correlate many signals simultaneously
- Detect and analyze timing violations
- Trace system execution
- Detect signal changes using triggers

For keyboard shortcuts, click `More`.

Keyboard Shortcuts

Actions	Description	Applicable When
Ctrl+X	Cut	Wave is selected
Ctrl+C	Copy	Wave is selected
Ctrl+V	Paste	Wave is selected
Delete	Delete	Wave is selected
Ctrl+-	Zoom out	Always
Shift+Ctrl+-	Zoom out around active cursor	Always
Ctrl++	Zoom in	Always
Shift+Ctrl++	Zoom out around active cursor	Always
Shift+Ctrl+C	Move display to active cursor	When cursor is not in the display range
Space	Zoom out full	Always
Tab, **Right Arrow**	Next transition	Digital format wave is selected
Shift+Tab, **Left Arrow**	Previous transition	Digital format wave is selected
Ctrl+A	Select all waves	Always
Up Arrow	Select wave above selected	Wave is selected
Down Arrow	Select wave below selection	Wave is selected
Ctrl+Up Arrow	Move selected waves up	Wave is selected
Ctrl+Down Arrow	Move selected waves down	Wave is selected
Escape	Unselect all signals	Wave is selected
Page Up	Scroll up	Always
Page Down	Scroll down	Always



Open the Logic Analyzer App

On the Simulink toolstrip Simulation tab, click the **Logic Analyzer** app button. If the button is not displayed, expand the review results app gallery. Your most recent choice for data visualization is saved across Simulink sessions.

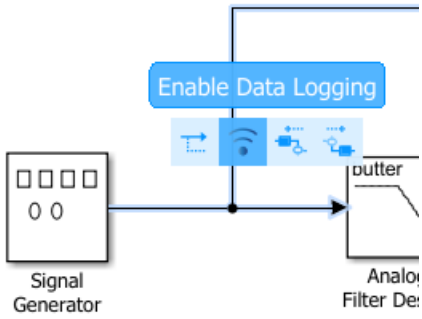
To visualize referenced models, you must open the Logic Analyzer from the referenced model. You should see the name of the referenced model in the Logic Analyzer toolbar.

Examples

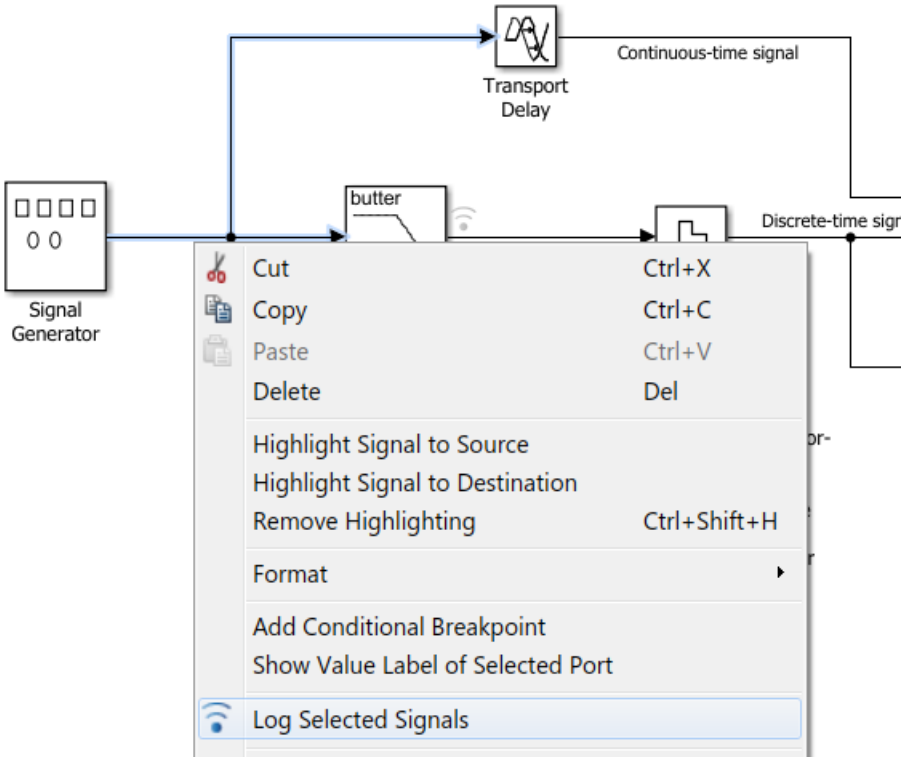
Select Signals to Analyze

The **Logic Analyzer** supports several methods for selecting data to visualize.

- Select a signal in your model. When you select a signal, an ellipsis appears above the signal line. Hover over the ellipsis to view options and then select the **Enable Data Logging** option.



- Right-click a signal in your model to open an options dialog box. Select the **Log Selected Signals** option.



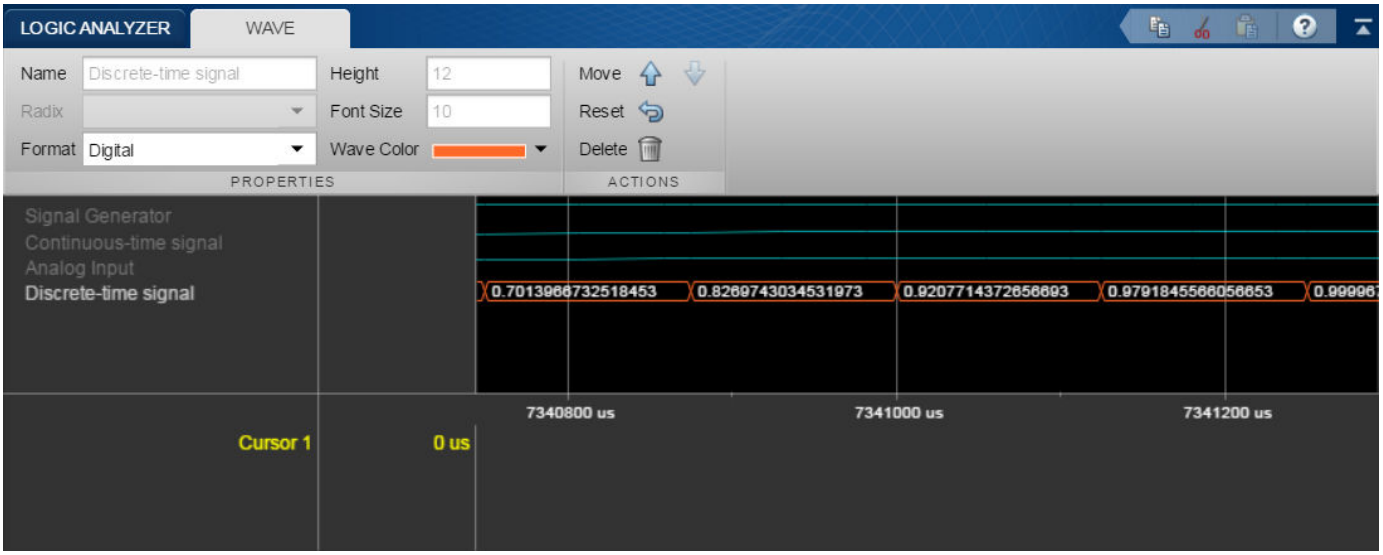
- Use any method to select multiple signal lines in your model. For example, use **Shift**+click to select multiple lines individually or **CTRL+A** to select all lines at once. Then, on the **Signal** tab, select the **Log Signals** button.

To visualize data in the Logic Analyzer, you must enable signal logging for the model. (Logging is on by default.) To enable signal logging, open **Model Settings** from the toolstrip, navigate to the **Data Import/Export** pane, and select **Signal logging**.

When you open the **Logic Analyzer**, all signals marked for logging are listed. You can add and delete waves from your **Logic Analyzer** while it is open. Adding and deleting signals does not disable logging, only removes the signal from the Logic Analyzer.

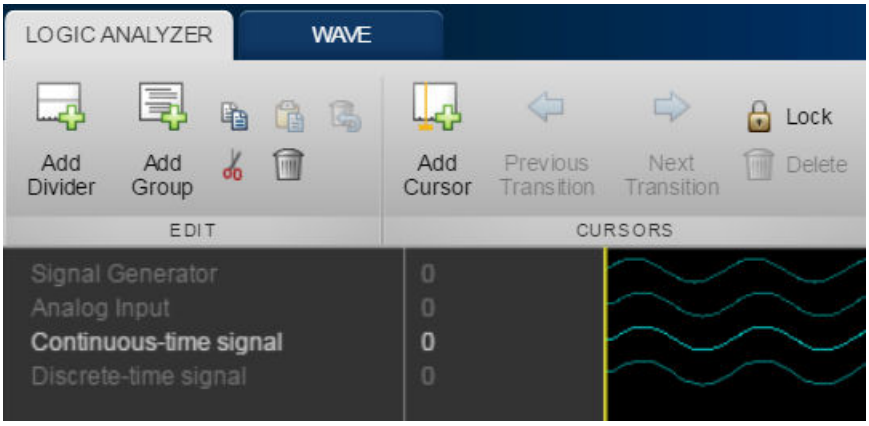
Modify Individual Wave Settings

Open the **Logic Analyzer** and select a wave by double-clicking the wave name. Then from the **Wave** tab, set parameters specific to the individual wave you selected. Any setting made on individual signals supersedes the global setting. To return individual wave parameters to the global settings, click **Reset**.



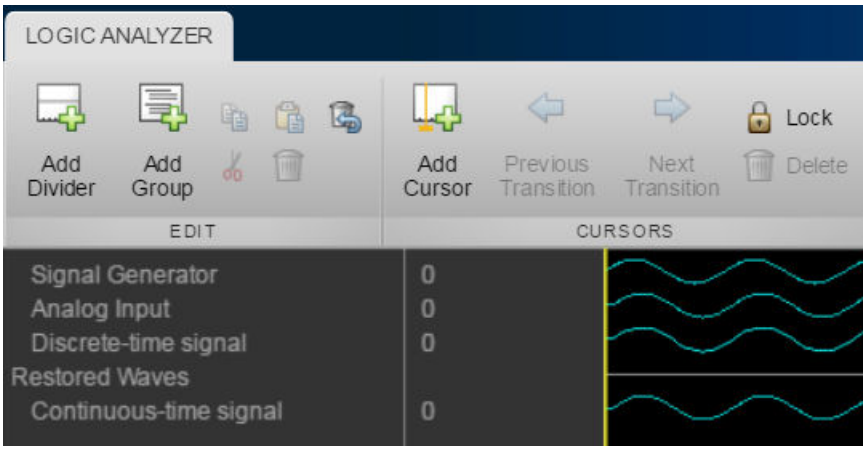
Delete and Restore Waves

1 Open the **Logic Analyzer** and select a wave by clicking the wave name.



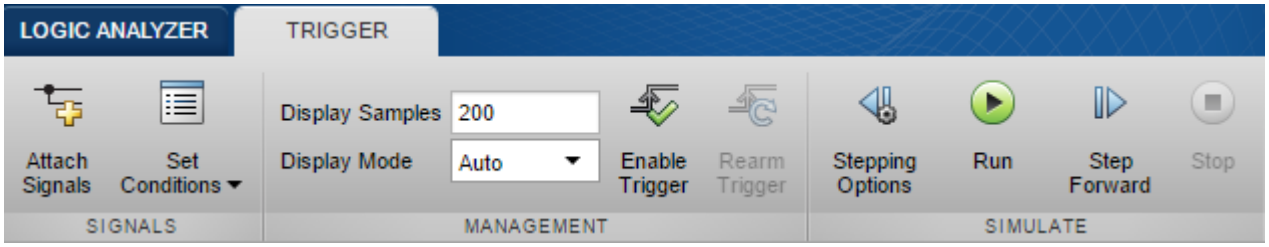
2. From the **Logic Analyzer** toolstrip, click . The wave is removed from the **Logic Analyzer**.
3. To restore the wave, from the **Logic Analyzer** toolstrip, click .

   A divider named **Restored Waves** is added to the bottom of your channels, with all deleted waves placed below it.



Add Trigger

1. Open the **Logic Analyzer** and select the **Trigger** tab.



2. To attach a signal to the trigger, select **Attach Signals**, then select the signal you want to trigger on. You can attach up to 20 signals to the trigger. Each signal can have only one triggering condition.
3. By default, the trigger looks for rising edges in the attached signals. You can set the trigger to look for rising or falling edges, bit sequences, or a comparison value. To change the triggering conditions, select **Set Conditions**.

If you add multiple signals to the trigger, control the trigger logic using the **Operator** option:

- `AND` - match all conditions.
- `OR` - match any condition.

4 To control how many samples you see before triggering, set the **Display Samples** option. For example, if you set this option to `500`, the **Logic Analyzer** tries to give you 500 samples before the trigger. Depending on the simulation, the **Logic Analyzer** may show more or fewer than 500 samples before the trigger. However, if the trigger is found before the 500th sample, the Logic Analyzer still shows the trigger.

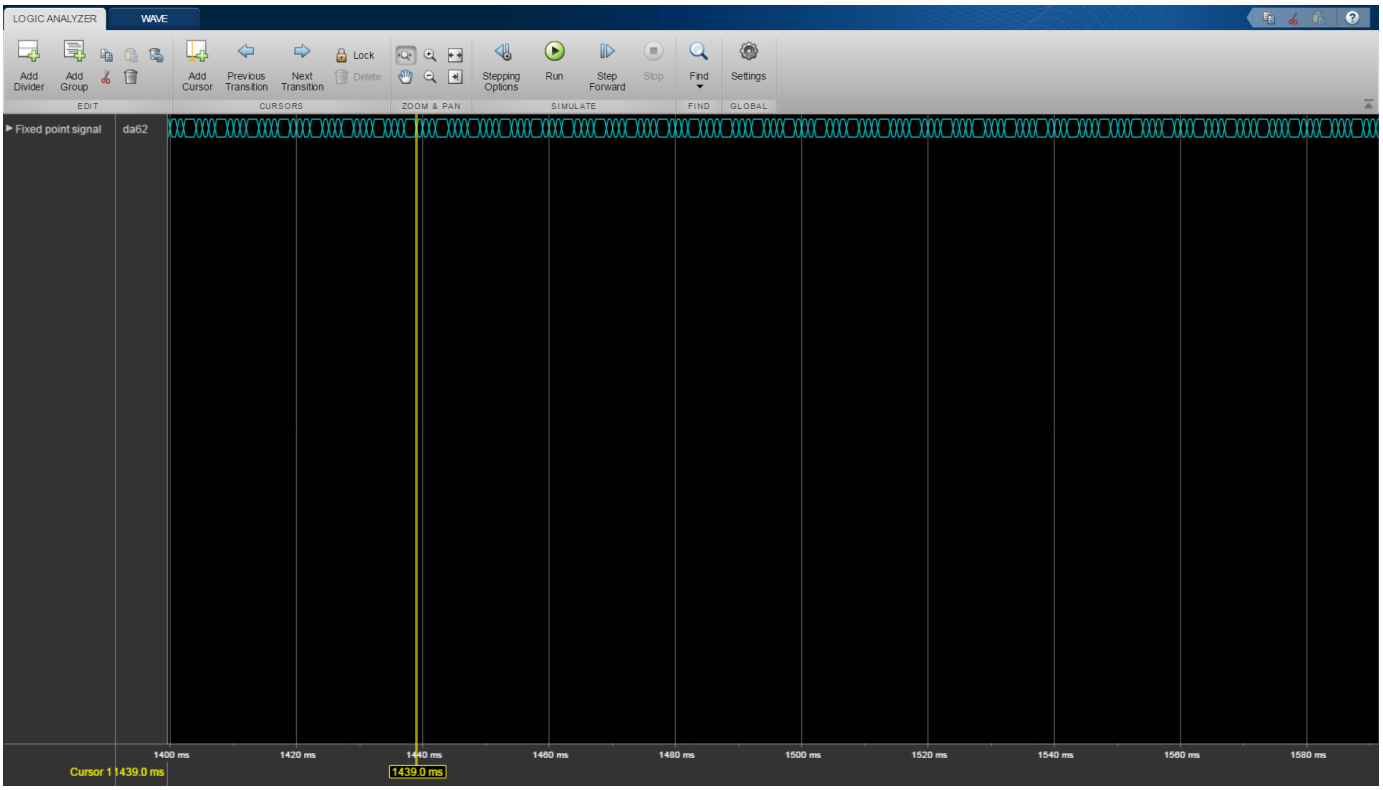
5 Control the trigger mode using **Display Mode**.

- `Once` - The **Logic Analyzer** marks only the first location matching the trigger conditions and stops showing updates to the Logic Analyzer. If you want to reset the trigger, select **Rearm Trigger**. Relative to the current simulation time, the **Logic Analyzer** shows the next matching trigger event.
- `Auto` - The **Logic Analyzer** marks every location matching the trigger conditions.

6 Before running the simulation, select **Enable Trigger**. A blue cursor appears as time 0. Then, run the simulation. When a trigger is found, the **Logic Analyzer** marks the location with a locked blue cursor.

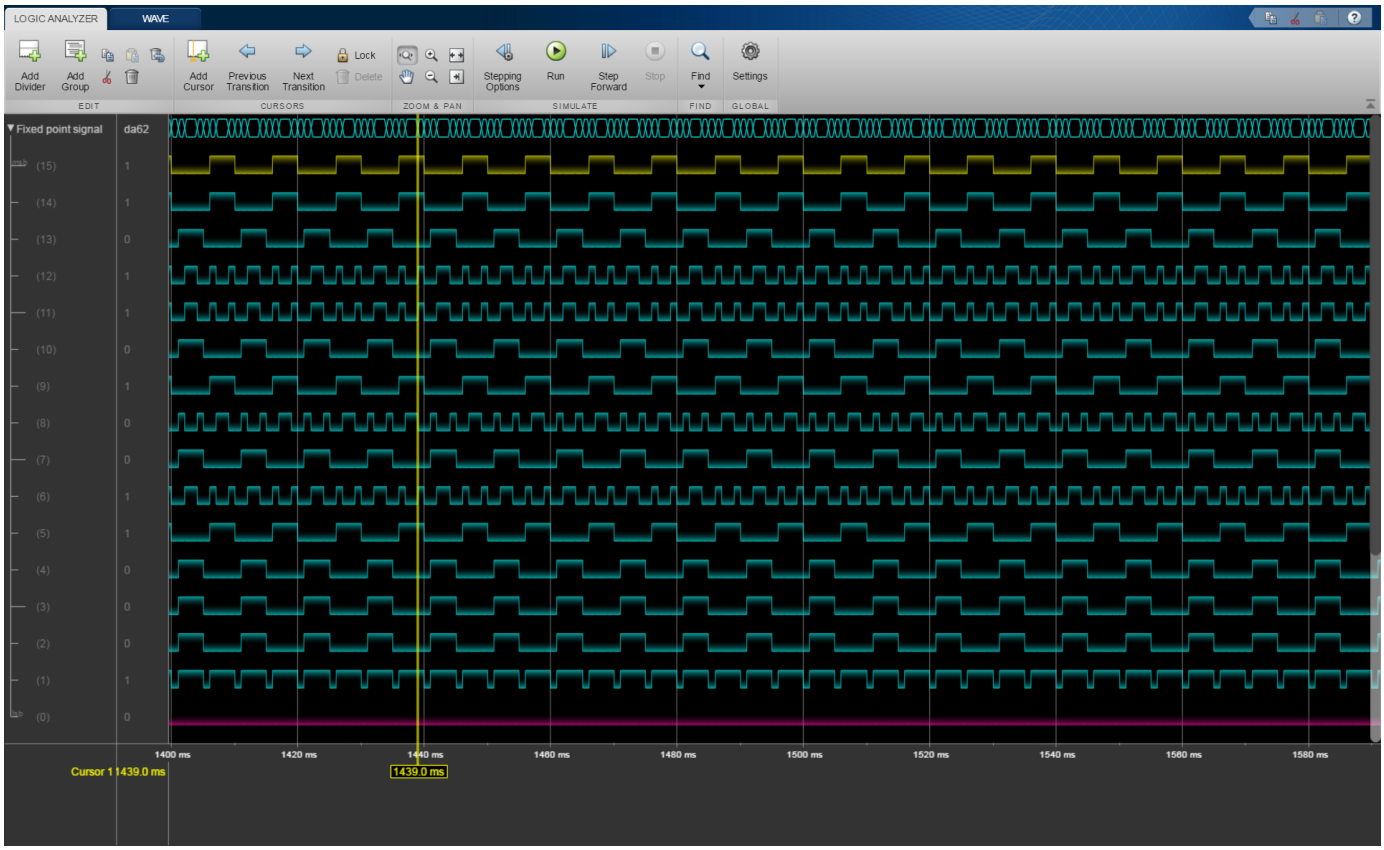
View Bit-Expanded Wave and Reverse Display Order of Bits

The **Logic Analyzer** enables you to bit-expand fixed-point and integer waves.

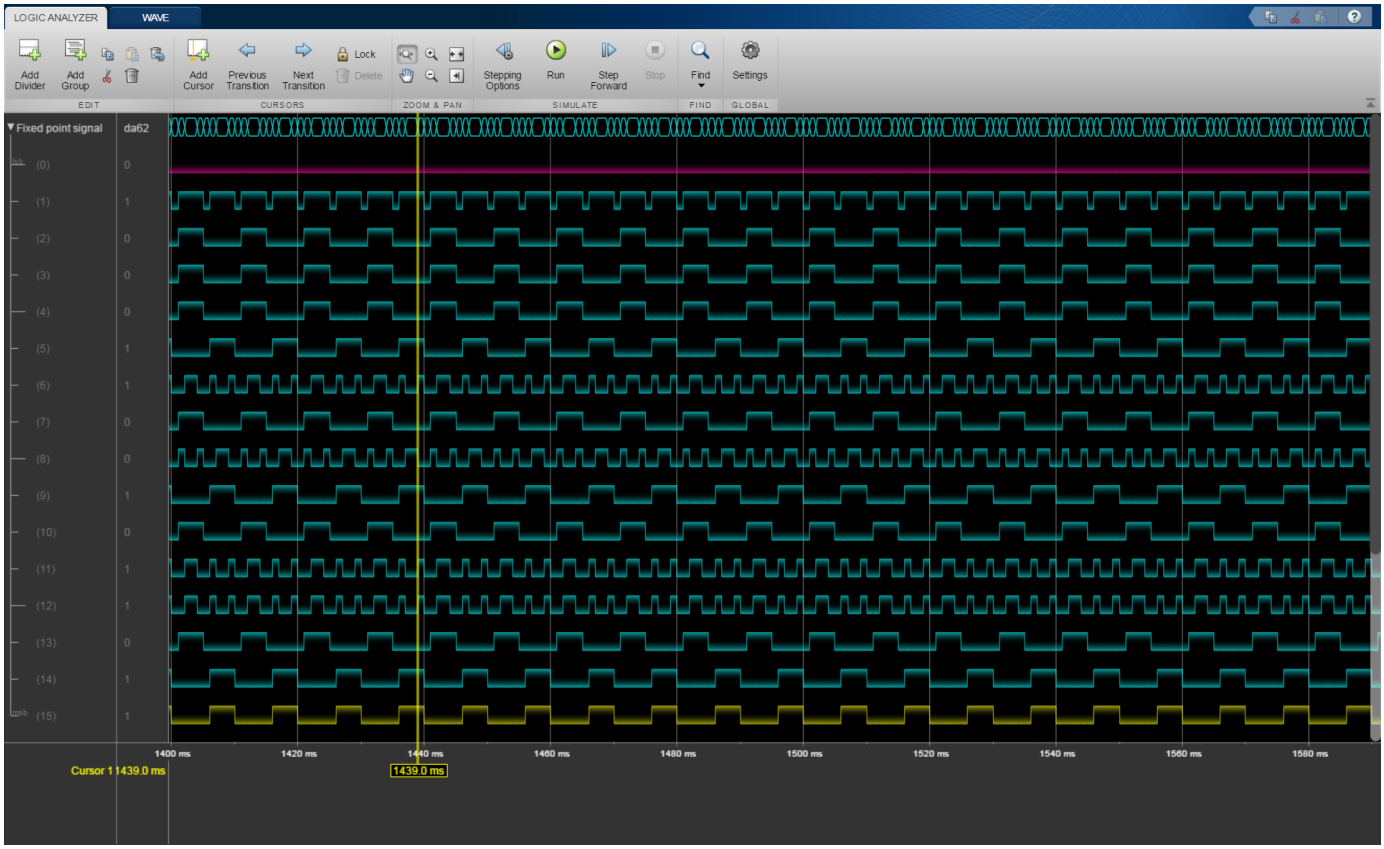


1 In the **Logic Analyzer**, click the arrow next to a fixed-point or integer wave to view the bits.

The least significant bit and the most significant bit are marked with **lsb** and **msb** next to the wave names.



2 Click Settings, and then select **Display Least Significant bit first** to reverse the order of the displayed bits.



- “Programmable FIR Filter for FPGA” (HDL Coder)
- “Packet-Based ADS-B Transceiver”
- “Log Simulation Output for States and Data” (Stateflow)
- “View Stateflow States in the Logic Analyzer” (Stateflow)

Limitations

Logging Settings

- If you enable the configuration parameter **Log Dataset data to file**, you cannot stream logged data to the **Logic Analyzer**.
- Signals marked for logging using `Simulink.sdi.markSignalForStreaming` or visualized with a Dashboard Scope do not appear on the **Logic Analyzer**.
- You cannot visualize Data Store Memory block signals in the **Logic Analyzer** if you set the **Log data store data** parameter to on.

Input Signal Limitations

- Signals marked for logging for the **Logic Analyzer** must have fewer than 8000 samples per simulation step.
- The **Logic Analyzer** does not support frame-based processing.

- For 64-bit integers and fixed-point numbers greater than 53 bits, if the numbers are greater than the maximum value of double precision, the transitions between numbers might not display correctly.
- You may see performance degradation in the **Logic Analyzer** for large matrices (greater than 500 elements) and buses with more than 1000 signals.
- The **Logic Analyzer** does not support Stateflow data output.

Graphical Settings

- While the simulation is running, you cannot zoom, pan, or modify the trigger.
- To visualize constant signals, in the settings, you must set the **Format** to `Digital`. Constants marked for logging are visualized as a continuous transition.

Supported Simulation Modes

Mode	Supported	Notes and Limitations
Normal	Yes	
Accelerator	Yes	You cannot use the **Logic Analyzer** to visualize signals in Model blocks with **Simulation mode** set to `Accelerator`.
Rapid Accelerator	Yes	Data is not available in the **Logic Analyzer** during simulation. If you simulate a model with the simulation mode set to rapid accelerator, after simulation the following signals cannot be visualized in the **Logic Analyzer**: • Multi-instance model reference signals • Nonvirtual bus signals
Processor-in-the-loop (PIL)	No	
Software-in-the-loop (SIL)	No	
External	No	

For more information about these modes, see “How Acceleration Modes Work”.

Version History

Introduced in R2016b

Objects

Topics

“Programmable FIR Filter for FPGA” (HDL Coder)
“Packet-Based ADS-B Transceiver”
“Log Simulation Output for States and Data” (Stateflow)
“View Stateflow States in the Logic Analyzer” (Stateflow)

Memory Mapper

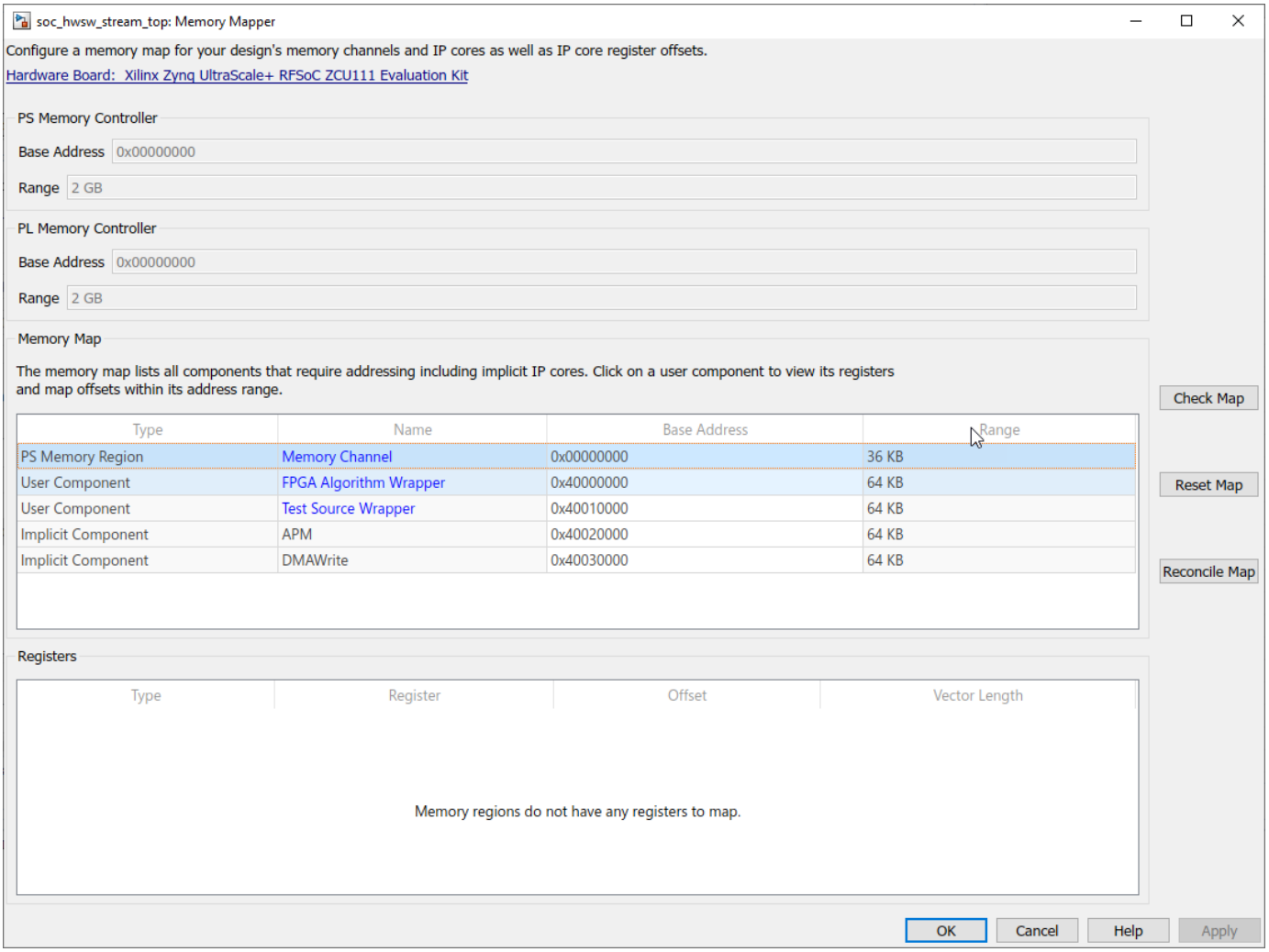
Configure memory map for SoC application

Description

View and edit memory regions of an SoC application. Edit device base addresses and offsets for memory-mapped devices.

Using the **Memory Mapper** tool, you can:

- View and edit base addresses, offsets, and memory locations of various channels and memory-mapped components in your design.
- Check the memory map of your model for any conflicts between different memory channel configurations.
- Reset the memory map to its default settings.
- Reconcile an edited map to match model settings.



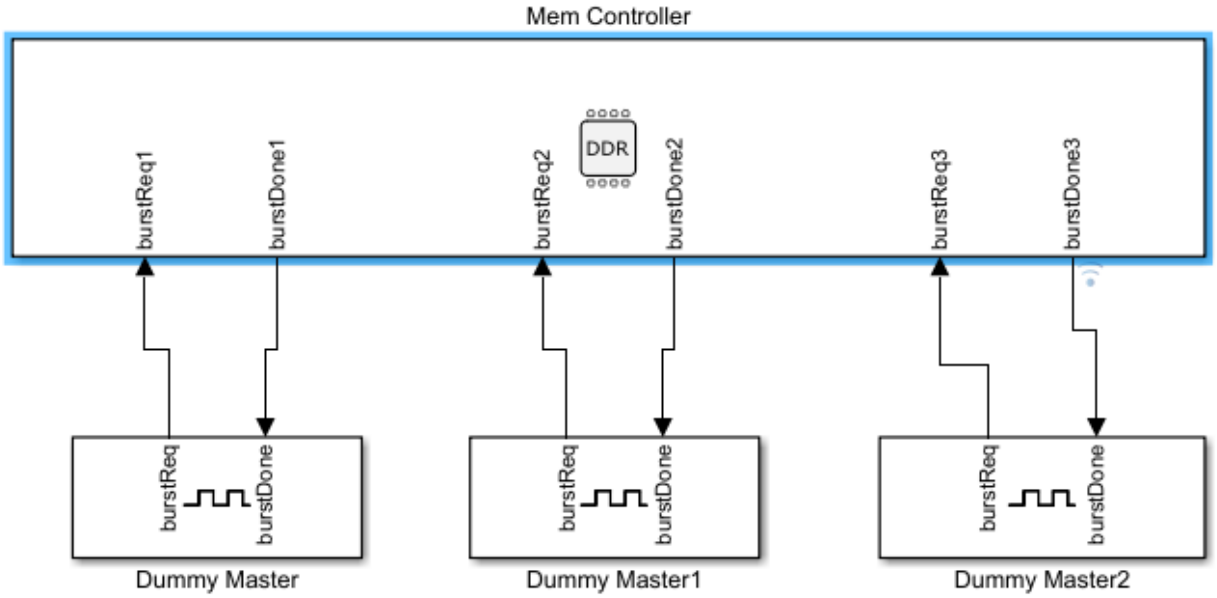
Open the Memory Mapper

- In the Configuration Parameters dialog box, select **Hardware Implementation** from the left pane. Under **Target hardware resources**, select **FPGA design (top-level)** and click **View/Edit Memory Map**.
- In the SoC Builder tool, in the **Review Memory Map** section, click **View/Edit**.

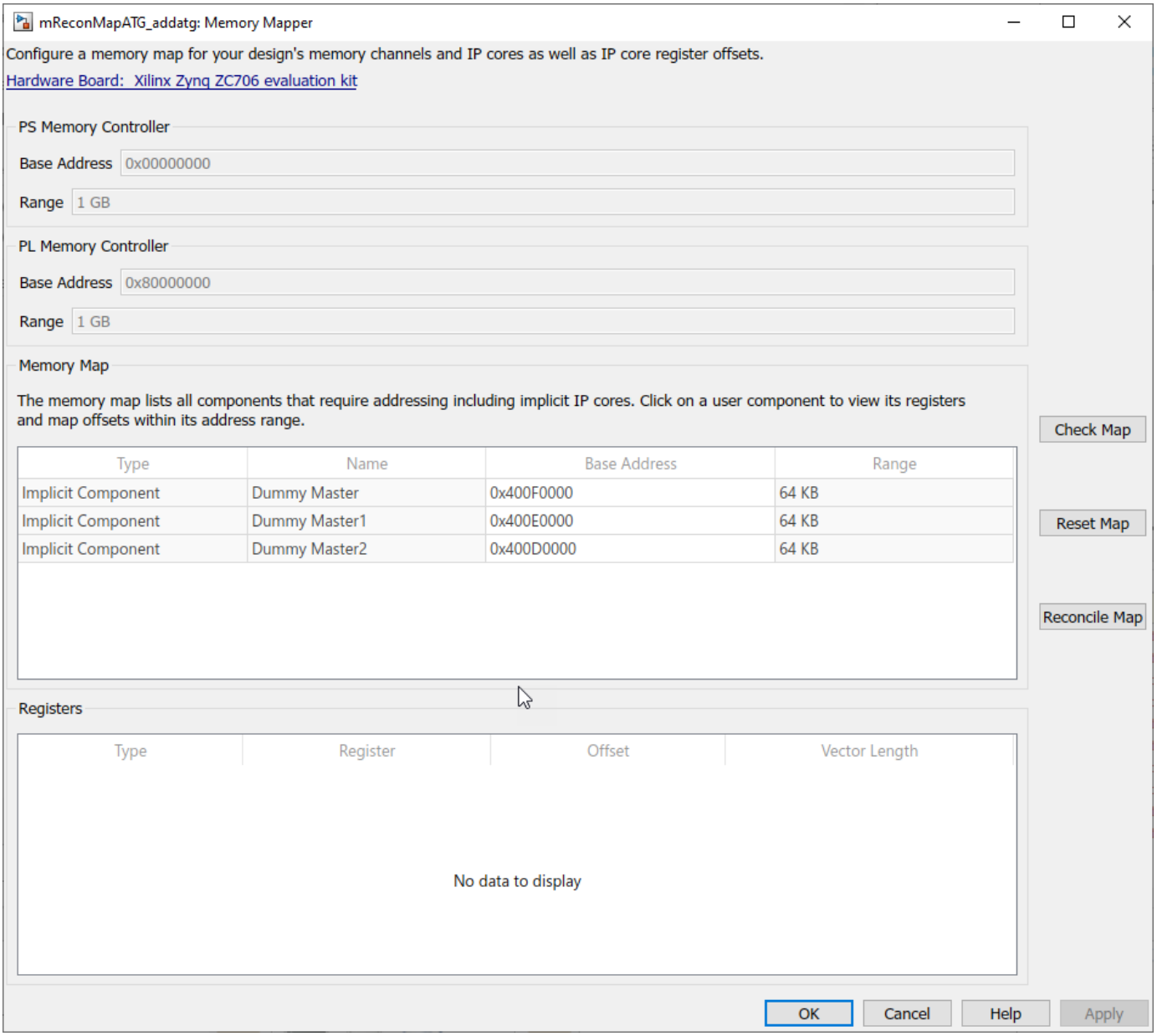
Examples

Reconcile Model with Memory Map

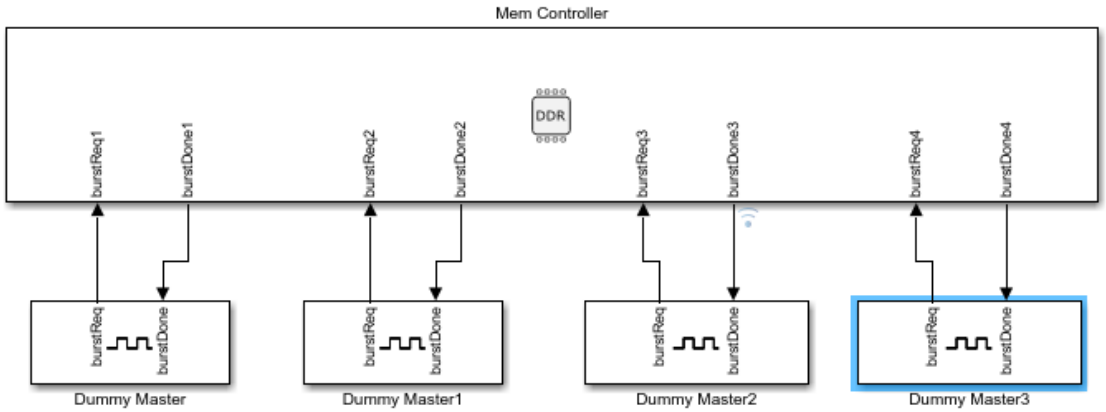
Consider a model with three masters (represented by Memory Traffic Generator blocks), connected to a Memory Controller block.



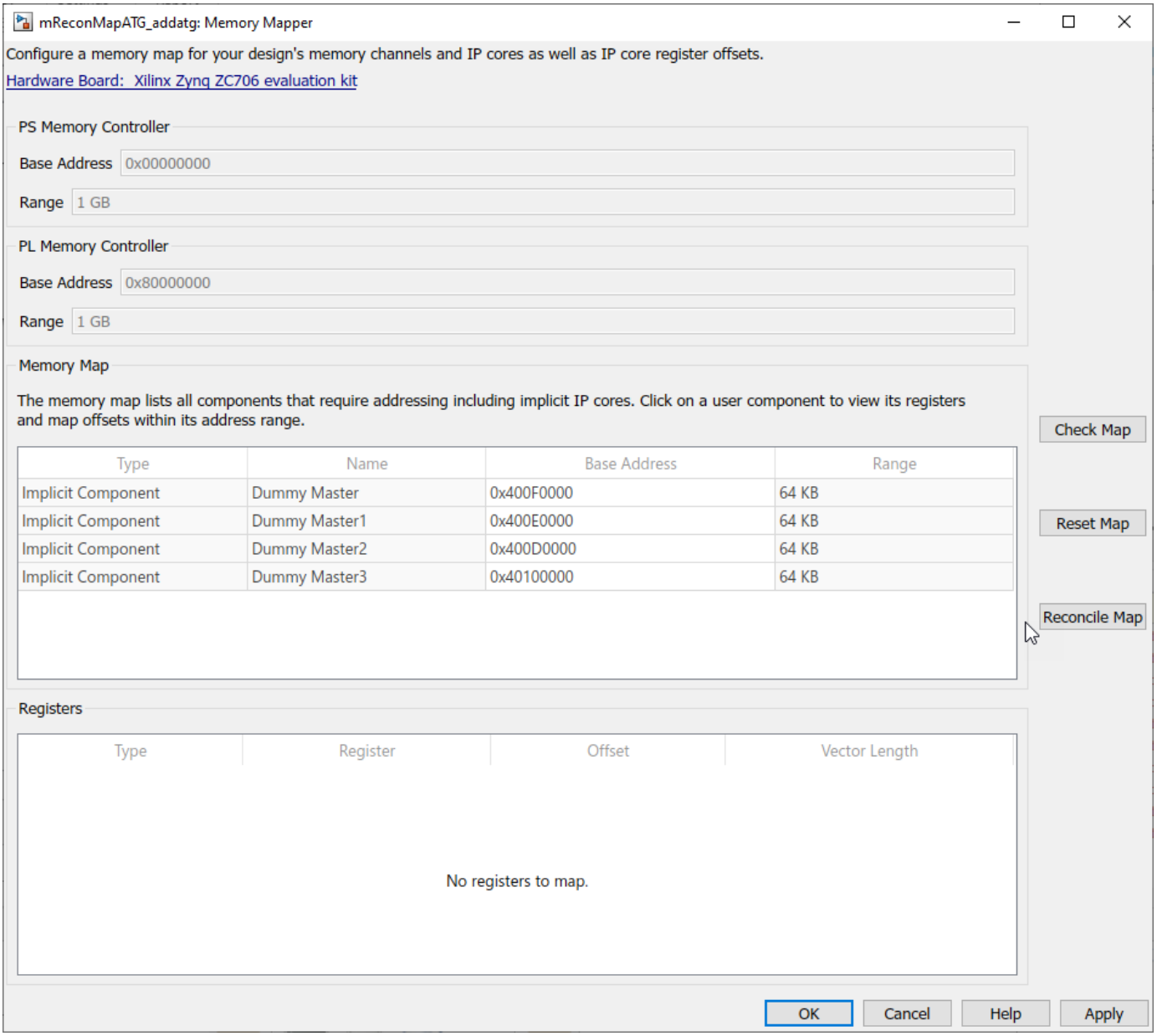
To open the **Memory Mapper** tool, first open the Configuration Parameters dialog box, and then select **Hardware Implementation** from the left pane. Under **Target hardware resources**, select **FPGA design (top-level)** and click **View/Edit Memory Map**.



The **Memory Mapper** lists the three masters in the design. Edit their base addresses as per your requirements. Add another channel to your model.

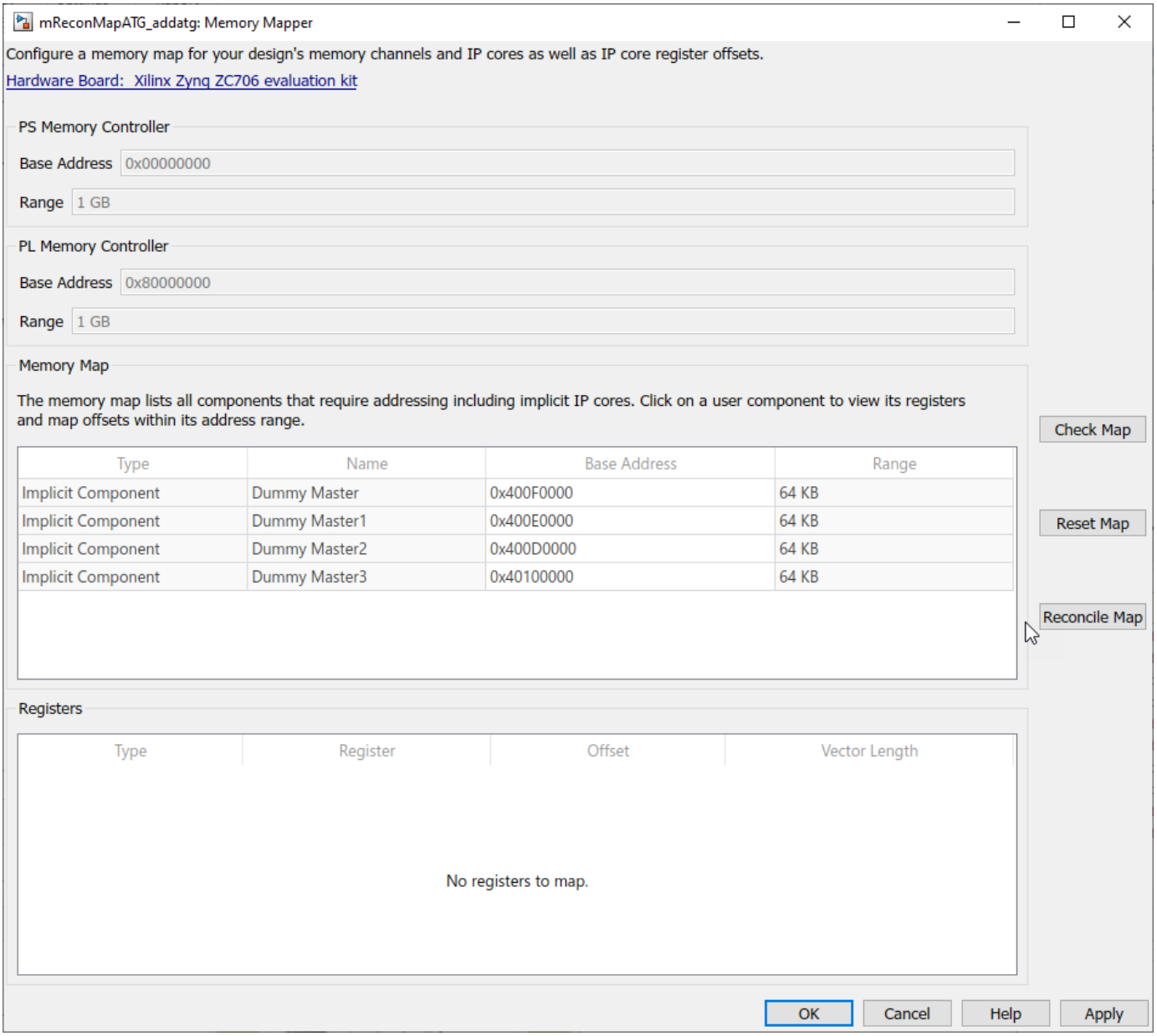


The model consists of four memory channels, while the **Memory Map** section shows only three. To resolve this conflict, click **Reconcile Map**. This adds another line, which represents the added channel, to the memory map table.



Reset Map

Click **Reset Map** to create a new, autogenerated map. The base addresses of the channels are reset to a default value.



Parameters

`Hardware Board` — View selected hardware board
selected hardware board

This parameter is read-only.

This Parameter shows the targeted hardware board. Click the link to open the configuration parameters on the **Hardware Implementation** pane, and change any of the hardware configurations. To learn more about board configuration parameters, see **Hardware Implementation Pane Overview**.

PS Memory Controller

Base Address — Base address of PS memory
`0x00000000` (default) | 32-bit hexadecimal address

This parameter is read-only.

This parameter shows the base address of the external memory. This value is a 32-bit hexadecimal value.

Range — PS memory address range
positive integer

This parameter is read-only.

This parameter shows the size of the external memory available for the selected hardware board. The memory size range is displayed in units of:

- `KB` - kilobytes
- `MB` - megabytes
- `GB` - gigabytes

This value is derived from the hardware board selected in the configuration parameters.

PL Memory Controller

Base Address — Base address of PL memory
`0x00000000` (default) | 32-bit hexadecimal address

This parameter is read-only.

This parameter shows the base address of the external memory. This value is a 32-bit hexadecimal value.

Range — PL memory address range
positive integer

This parameter is read-only.

This parameter shows the size of the external memory available for the selected hardware board. The memory size range is displayed in units of:

- `KB` - kilobytes
- `MB` - megabytes
- `GB` - gigabytes

This value is derived from the hardware board selected in the configuration parameters.

Memory Map

Check Map — Check memory map
button

Check that the memory map has no overlapping regions or registers, and that memory addresses are properly aligned.

`Reset Map` — Reset memory map
button

Reset the memory map to its initial values.

`Reconcile Map` — Reconcile memory map with existing model
button

Reconcile the memory map with the existing model. After adding or deleting a channel or a memory-mapped register to your model, click this button to synchronize between the model and the memory map. To verify that the reconciled memory map is valid, click **Check Map** after reconciling.

Note Clicking **Reconcile Map** matches the memory map to the model but does not reset the base address values of the memory areas.

Version History

Introduced in R2019a

See Also

SoC Builder | Memory Controller | Memory Channel

Topics

“Random Access of External Memory”

SoC Builder

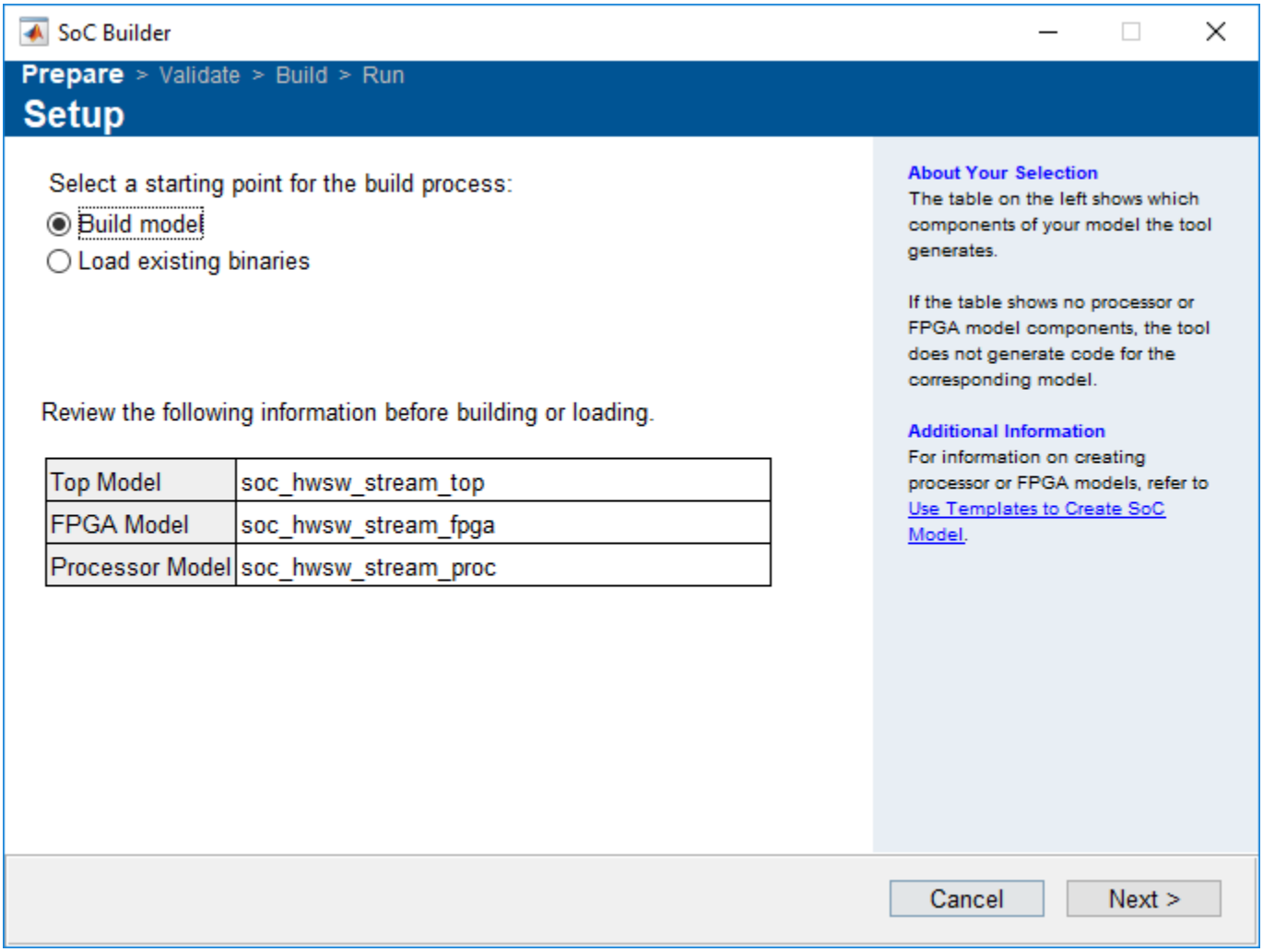
Build, load, and execute SoC model on SoC, FPGA, and MCU boards

Description

The **SoC Builder** tool steps through the various stages for building and executing an SoC model on an SoC, FPGA, or MCU board.

Using this tool, you can:

- Review the model information provided to the tool.
- Choose between different build actions.
- Set up a folder to store all generated files.
- Map model tasks to interrupt service routines.
- Configure the peripheral register settings.
- Review the memory map and edit it if needed.
- Validate that the model has all required components for generating a programming file.
- Build the model using Xilinx Vivado, Intel Quartus, Texas Instruments Code Composer Studio™ tool families.
- Configure the Ethernet connectivity.
- Load the programming file to your board.
- Run the application.



Open the SoC Builder

- Simulink Toolstrip: On the **System on Chip** tab, click **Configure, Build & Deploy**.
- Simulink Toolstrip: click the **System on Chip** tab, and then press **Ctrl+B**.
- MATLAB command prompt: Enter `socBuilder('modelname')`.

Note If the **System on Chip** tab is not visible, on the **Apps** tab, under **Setup to Run on Hardware** click the **System on Chip (SoC)** app icon.

Examples

- “Generate SoC Design”
- “Generate Design Using SoC Builder” (SoC Blockset Support Package for Xilinx Devices)

Programmatic Use

`socBuilder('modelname')` opens SoC Builder and loads the specified model into the tool.

Version History

Introduced in R2019a

See Also

Tools
Memory Mapper | Hardware Mapping

Objects
`socModelBuilder`

Topics
“Generate SoC Design”
“Generate Design Using SoC Builder” (SoC Blockset Support Package for Xilinx Devices)

Task Execution Report

Display summary of task execution performance and core usage metrics following model simulation or execution on processor

Description

The **Task Execution Report** tool displays summaries of the task executions and core usage of the processor. You can use the report following model simulation or following execution on a hardware processor.

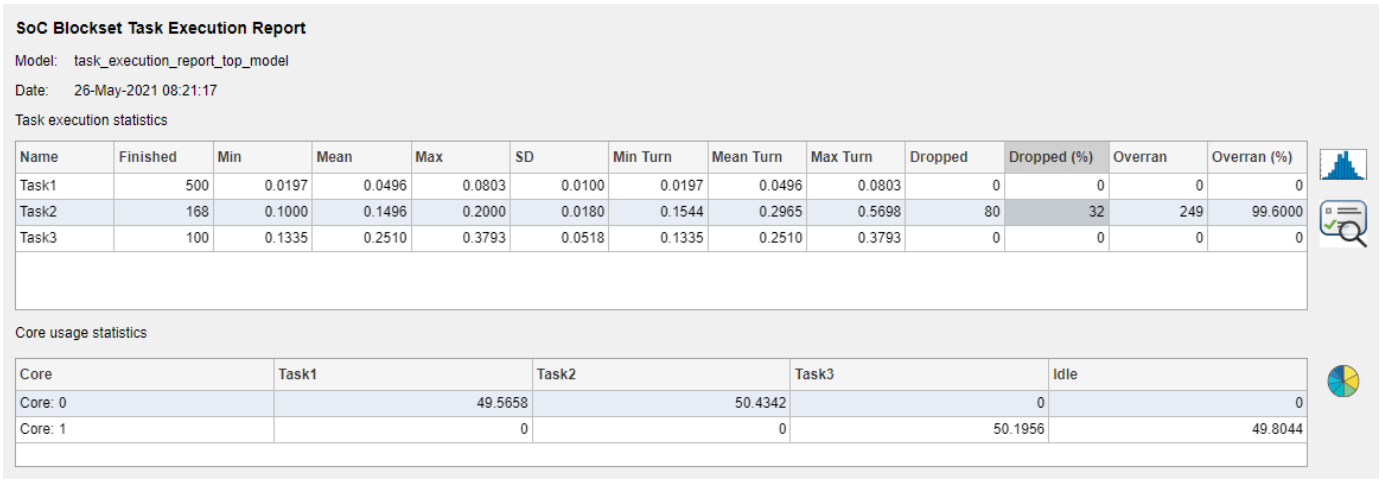
This tool displays these values:

- Total executions per task.
- Minimum, mean, maximum, and standard deviation (SD) of execution times per task.
- Minimum, mean, and maximum turnaround times per task.
- Total and percentage of dropped task instances.
- Total and percentage of overrun task instances.
- Task distribution per processor core.

In this tool, you can open the SoC Blockset Overrun Inspector subwindow, which has these features:

- Time instance when the task should have run.
- Indication if the task overran resulted in a task drop.
- Filter switch to **Show only dropped** tasks for a more focused analysis.

Using the overrun inspector window, you can also analyze each overran task instance by examining a detailed execution timing and core usage plots in the vicinity of the overran (indicated by a dasher vertical line). You can use the filter switch to Show only dropped tasks for a more focused analysis.



SoC Blockset Task Execution Report

Model: task_execution_report_top_model

Date: 26-May-2021 08:21:17

Task execution statistics

Name	Finished	Min	Mean	Max	SD	Min Turn	Mean Turn	Max Turn	Dropped	Dropped (%)	Overran	Overran (%)
Task1	500	0.0197	0.0496	0.0803	0.0100	0.0197	0.0496	0.0803	0	0	0	0
Task2	168	0.1000	0.1496	0.2000	0.0180	0.1544	0.2965	0.5698	80	32	249	99.6000
Task3	100	0.1335	0.2510	0.3793	0.0518	0.1335	0.2510	0.3793	0	0	0	0

Core usage statistics

Core	Task1	Task2	Task3	Idle
Core: 0	49.5658	50.4342	0	0
Core: 1	0	0	50.1956	49.8044

Open the Task Execution Report

- Simulink Toolstrip: On the **System on Chip** tab, click **Execution Report**.

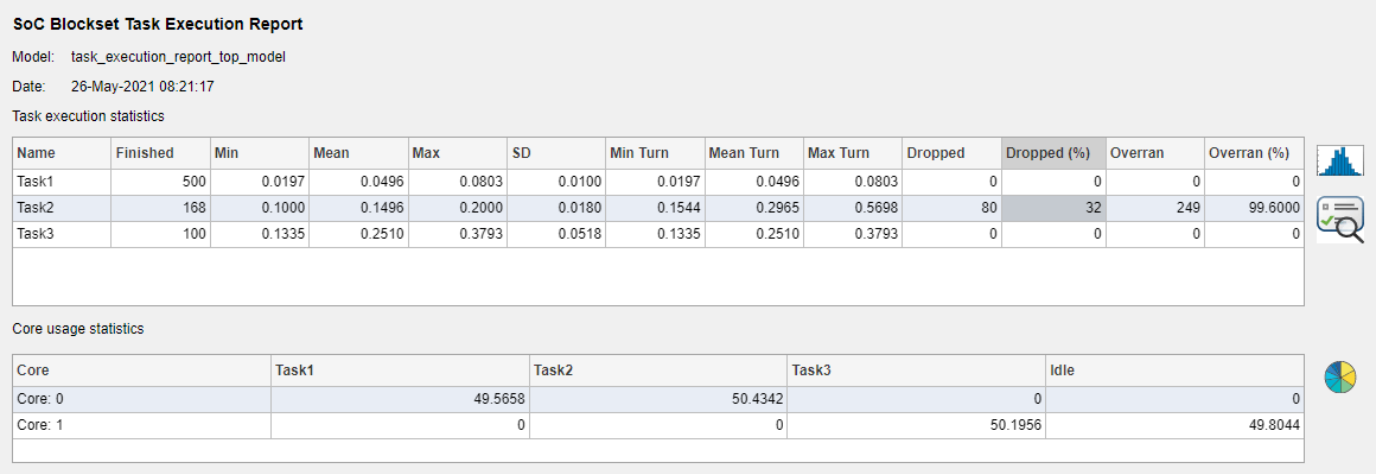
Examples

Task Execution Report for SoC Model

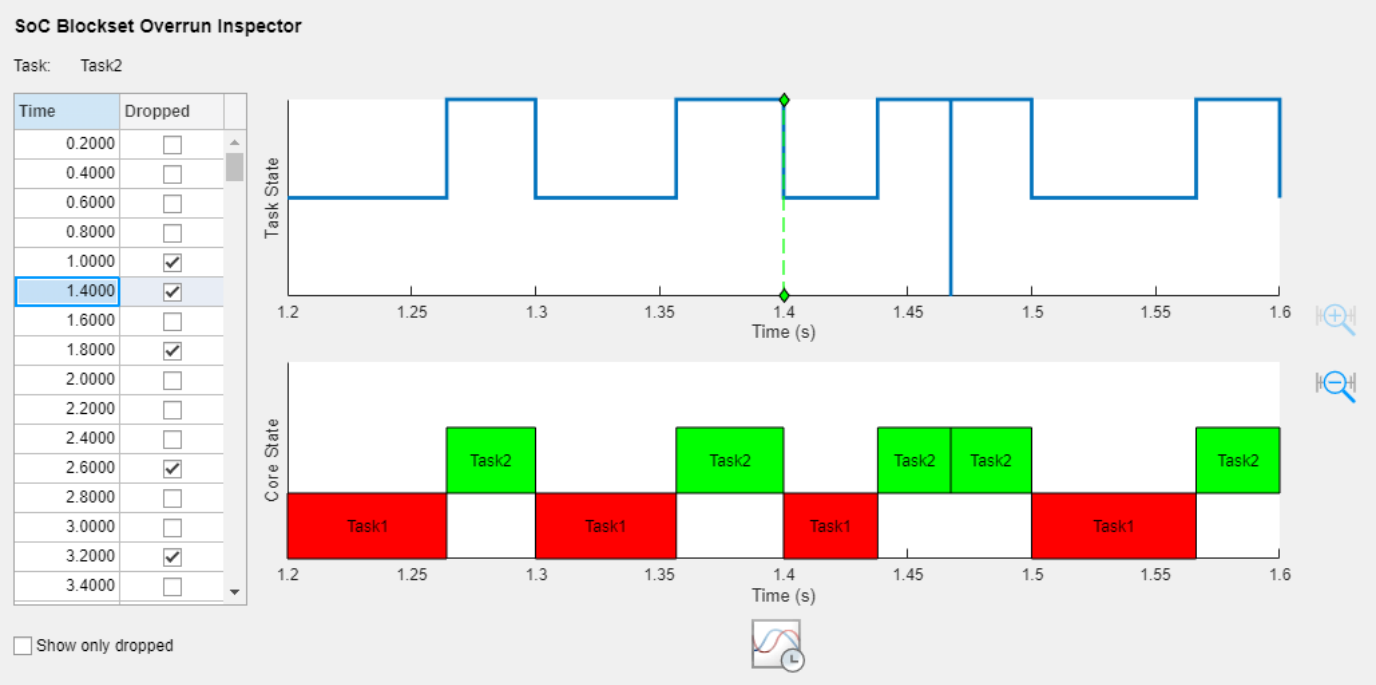
View the execution data for an SoC Blockset model using the **Task Execution Report** tool. The `task_execution_report_top_model` model contains three timer-driven tasks that execute over two processor cores. Run the model.

```
load_system('task_execution_report_top_model.slx');
sim('task_execution_report_top_model');
```

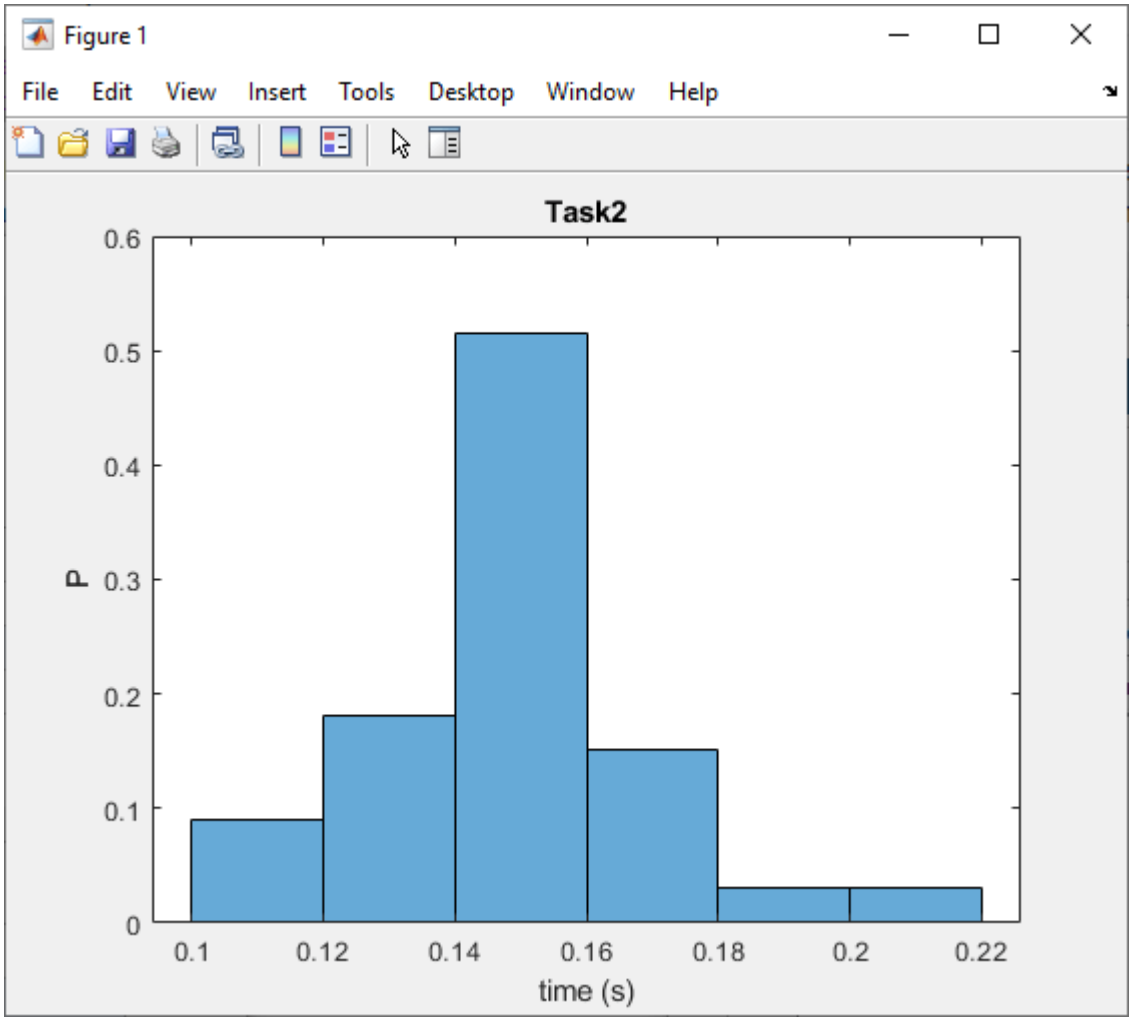
Open the **Task Execution Report** tool by clicking **Execution Report o**n the **System on Chip** tab. The tool shows the summary of data, including the execution time, turnaround time, dropped tasks, and task overrun, for each task. The tool also shows the usage statistics for each processor core. This figure shows a snapshot of the tool.



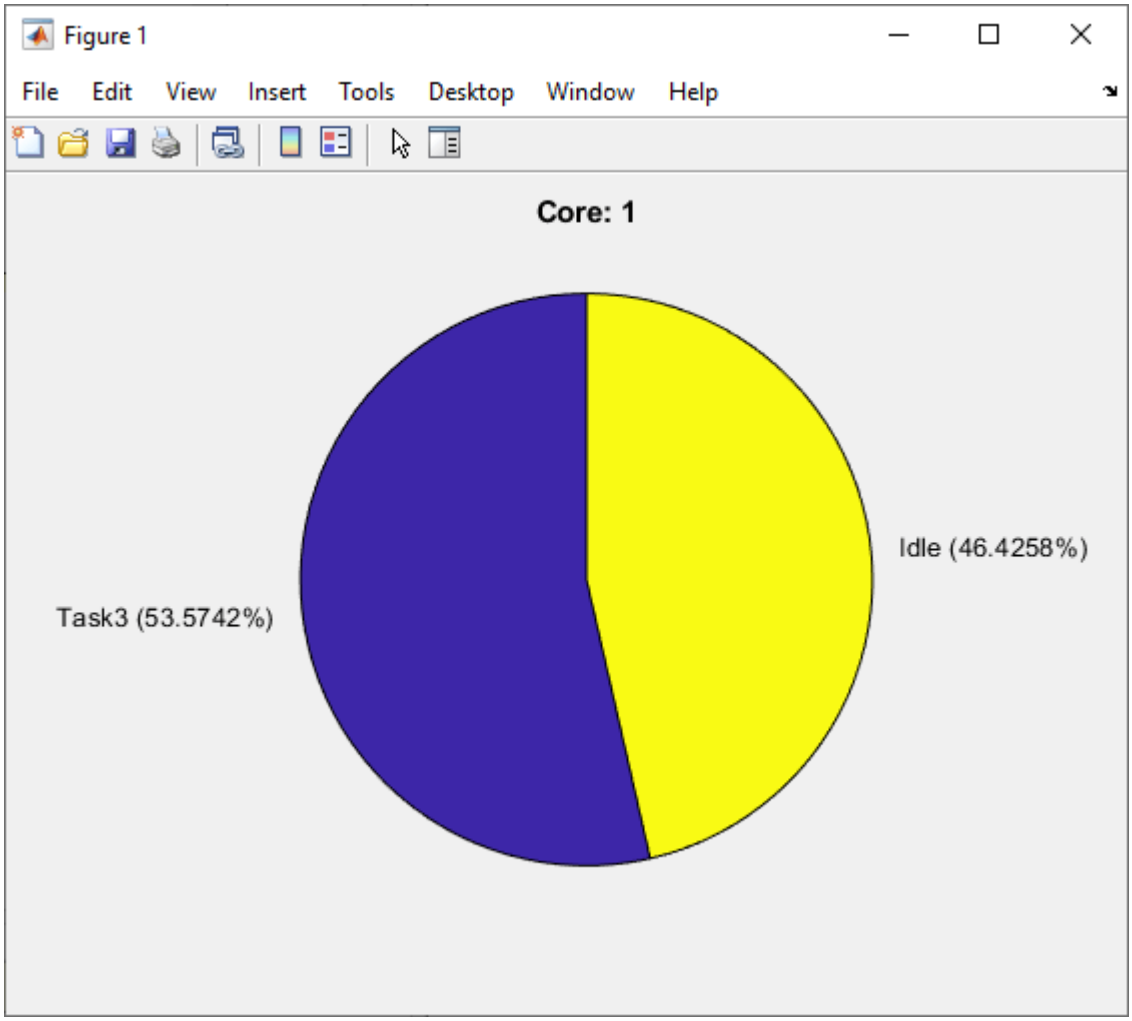
In this case, the tool shows that **Task1** and **Task2** compete for resources on `Core 0`. **Task 2**, the lower priority task, gets dropped 32% of the time with 99.6% overrun. Examine the overruns by clicking the Analysis of the Task Overruns icon to open the Overrun Inspector tool. This figure shows the Overrun Inspector window.



View a histogram of the task execution over the course of a simulation or hardware execution. To view the histogram, select a task and click **Histogram of Task Execution Times** icon. This figure shows the execution time of Task 1.



View a pie chart of the core utilization over the course of a simulation or hardware execution by clicking the `Chart of the Core Usage` icon. This figure shows the core usage of Core 1.



Version History

Introduced in R2021b

See Also

`socHardwareUsage`

HDL IP Importer

Import HDL IP core into SoC model

Description

Use the **HDL IP Importer** tool to import an existing HDL IP core into an SoC Simulink model. The tool takes you through the steps to generate a library block that you can integrate into your Simulink model. You can then simulate, build, and deploy the model on a hardware board using the **SoC Builder** tool.

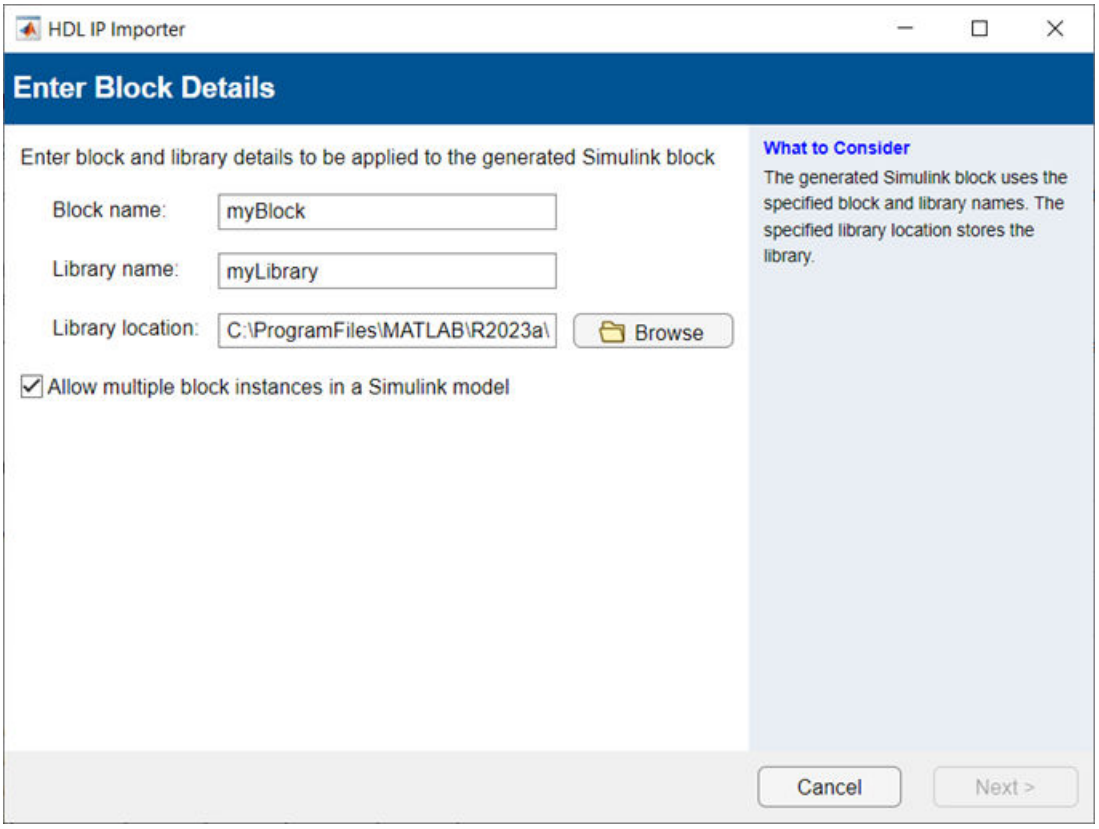
Note Use this tool only to target Xilinx devices.

Using this tool, you can:

- Import a Xilinx or custom HDL IP as a Simulink block.
- Generate the editable simulation and customization files with the `.m` and `.tcl` extensions, respectively.
- Specify the block name, library name, and library location.
- Include multiple instances of the generated block in the same Simulink model.
- Specify constraints on the HDL IP core.
- Review or update the HDL IP interfaces that the tool adds as ports to the generated block.
- Review clock and reset interfaces and select the required source for these interfaces.
- Specify a device tree file for the HDL IP and the pre- and post-load scripts.

You can also import multiple HDL IP cores into the Simulink model as a single block. To import multiple HDL IP cores, package multiple IPs into single IP in Vivado, and then use the **HDL IP Importer** tool.

The tool supports the AXI4-Stream, AXI4-Stream Video, AXI4 Master, I/O, clock, and reset HDL IP interfaces for the block generation. You can add these interfaces as ports to the generated block. The tool supports the AXI4 Slave HDL IP interface for block generation only. The tool does not add the AXI4 Slave interface to the generated block as a port.



Open the HDL IP Importer

- Simulink toolstrip: On the **System on Chip** tab, click **HDL IP Importer**.
- MATLAB command prompt: Enter `hdlIPImporter`.

Examples

- “Import HDL IP Core into SoC Model”
- “Import Custom HDL IP into SoC Blockset Design” (SoC Blockset Support Package for Xilinx Devices)

Version History

Introduced in R2023a

See Also

Tools
SoC Builder

Topics
“Import HDL IP Core into SoC Model”

“Import Custom HDL IP into SoC Blockset Design” (SoC Blockset Support Package for Xilinx Devices)
“Generate SoC Design”

SoC Model Creator

Create SoC model based on selected reference design

Description

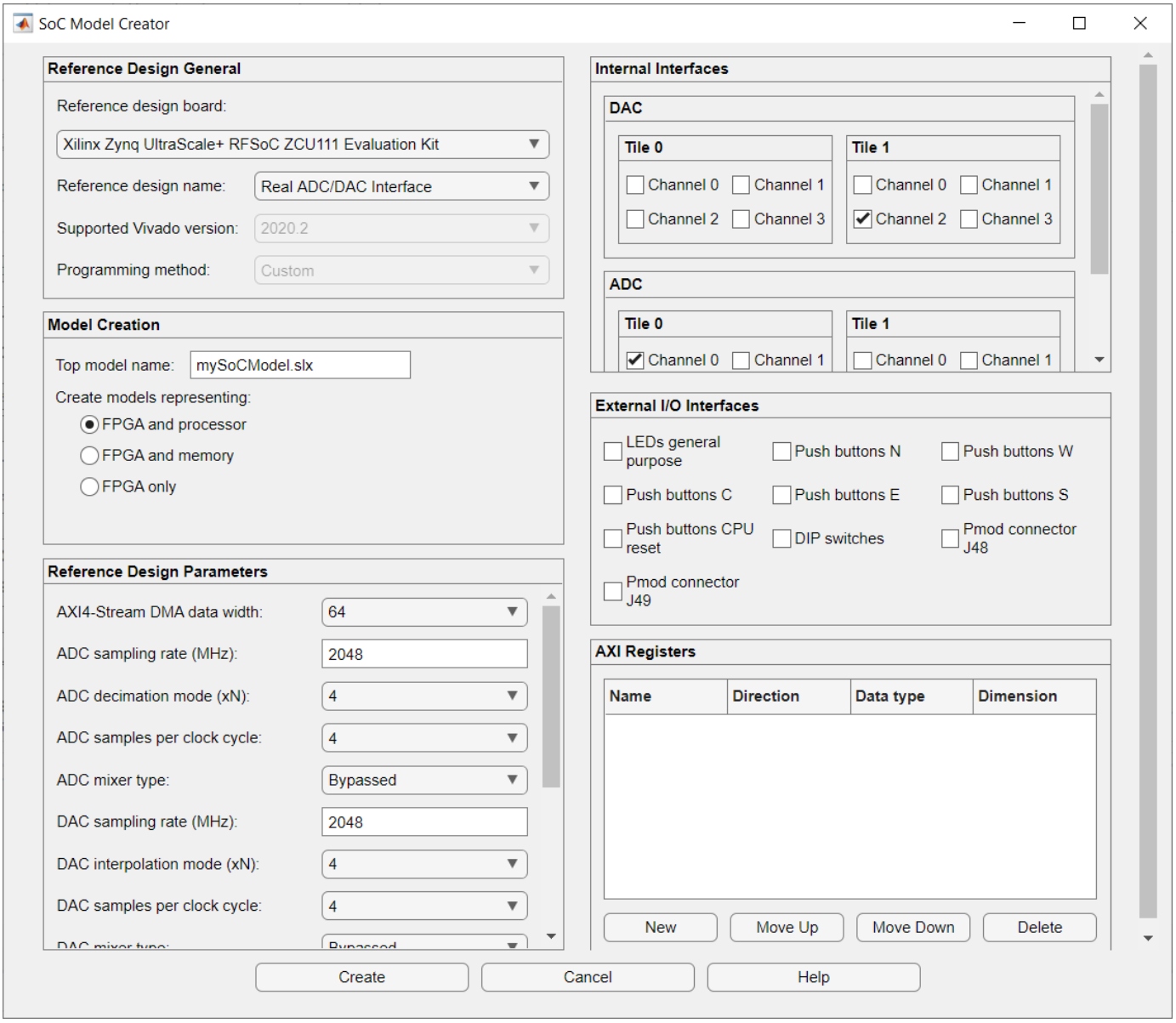
The **SoC Model Creator** tool enables you to create an SoC model that is compatible with a selected reference design. Use this tool to select a reference design and configure its parameters before you create an SoC model. The created model has a subsystem that adheres to the interfaces as per your selection. You can add an algorithm in the subsystem in the created model for simulation, HDL code generation, and SoC deployment.

Note The **SoC Model Creator** tool replaces the removed **SoC Template Builder** tool. For more information, see “Compatibility Considerations” (SoC Blockset Support Package for Xilinx Devices).

The **SoC Model Creator** tool enables you to choose the reference design for which you want to create an SoC model. The tool also enables you to select the type of model and customize the model by using the reference design parameters, predefined internal interfaces, external input/output (I/O) interfaces, and AXI registers. Use the **SoC Model Creator** to perform these actions.

- Create an SoC model with the required I/O blocks connected.
- Configure the SoC model with the selected board and reference design.
- Set the target interface for the SoC model input and output.

Edit the created model to include the required algorithm. Navigate to the blocks **FPGA Algorithm** in the FPGA model or **Processor Algorithm** in the processor model. Replace these blocks with your own algorithm model. Then, simulate the system and use the **SoC Builder** tool to build software executables and an FPGA programming file from your model and program the target hardware board.



Open the SoC Model Creator

- MATLAB command prompt: Enter `socModelCreator`.

Examples

- “Transmit and Receive Tone Using Fixed Reference Design Workflow on RFSoC Device” (SoC Blockset Support Package for Xilinx Devices)
- “Transmit Signal Waveform Using DDR4 on Xilinx RFSoC Device” (SoC Blockset Support Package for Xilinx Devices)

- “Full Duplex Data Streaming between Software and Hardware on Versal Platform”

Limitations

Use this tool only when targeting the Xilinx Zynq UltraScale+ or Xilinx Versal® ACAP devices.

Version History

Introduced in R2021b

R2021b: SoC Model Creator tool replaces removed SoC Template Builder tool

Behavior changed in R2021b

The **SoC Template Builder** tool has been removed. Instead, use the **SoC Model Creator** tool to create an SoC model based on the selected reference design. The **SoC Model Creator** tool has these advantages over the removed **SoC Template Builder** tool.

- Create a template model with the interfaces that are compatible with the reference design.
- Create these types of template models.
  - FPGA and processor
  - FPGA and memory
  - FPGA only

To open the **SoC Model Creator** tool, enter the `socModelCreator` command at the MATLAB command prompt. Using the `socTemplateBuilder` command, which previously opened the **SoC Template Builder** tool, is not recommended. If you use the `socTemplateBuilder` command in the MATLAB command prompt, the command now opens the **SoC Model Creator** tool.

See Also

Tools

SoC Builder

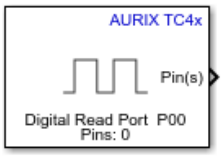
Topics

“Transmit and Receive Tone Using Fixed Reference Design Workflow on RFSoC Device” (SoC Blockset Support Package for Xilinx Devices)
“Transmit Signal Waveform Using DDR4 on Xilinx RFSoC Device” (SoC Blockset Support Package for Xilinx Devices)
“Full Duplex Data Streaming between Software and Hardware on Versal Platform”
“Configure Design Using SoC Model Creator” (SoC Blockset Support Package for Xilinx Devices)
“Generate Design Using SoC Builder” (SoC Blockset Support Package for Xilinx Devices)
“Configure Versal Design Using SoC Model Creator”
“Generate Versal Design Using SoC Builder”

Blocks

Digital Port Read

Read GPIO pin(s) status



Libraries:
Embedded Coder Support Package for Infineon AURIX TC4x Microcontrollers
SoC Blockset Support Package for Infineon AURIX Microcontrollers

Description

Read the logical status of a GPIO port pin for the Infineon AURIX TC4x Microcontrollers.

Ports

Input

msg — SoC Blockset simulation message
`scalar`

This port receives the SoC Blockset message from the interface blocks and the Digital Port Read block converts the **msg** into data to be emitted at the output port during simulation.

Dependencies

To view this port, select the **Enable simulation port** parameter and set the port type as **SoC Data (Message based)**.

sig — Simulink based simulation signal
`scalar` | `vector`

The simulation-only signal input port acts as pass-through with the output of the Digital Port Read block during simulation.

Dependencies

To view this port, select the **Enable simulation port** parameter and set the port type as **Signal**.

Output

Pin(s) — GPIO pin status
`scalar` | `vector`

The port outputs the status of the digital pin you select in the **Port number** and **Pin number** parameter from Digital Port Read Peripheral Configuration.

The port outputs the status as a logical scalar if the pin number is 1 or as a logical vector if it is greater than 1.

Data Types: `Boolean`

Parameters

Number of pins — Number of pins you want to configure
`1` (default) | `2` | `3` | `...`

Select the number of pins (`1` to `16`) whose output you want to configure in the block.

Sample time — Frequency at which block reads input pin values
`-1` (default) | `scalar` | `vector`

Specify in seconds how often the block should read the pin value.

When you specify this parameter as `-1`, Simulink determines the best sample time for the block based on the block context within the model.

Enable simulation port — Enable peripheral simulation ports in the block
`off` (default) | `on`

Select this parameter to enable peripheral simulation capability.

Note If you do not have a SoC Blockset license, the **Enable simulation port** parameter is visible but you cannot enable it.

Port Type — Specify the type of peripheral simulation port
`SoC Data (Message based)` (default) | `Signal`

Select this parameter to configure either the SoC Blockset compatible or the Simulink signal based simulation ports to enable peripheral simulation capability.

Dependencies

To view this parameter, select the **Enable simulation port** parameter.

Version History

Introduced in R2022b

See Also

Digital Port Write | Digital Port Read Peripheral Configuration

Digital Port Write

Set GPIO pin(s) status

Libraries:
Embedded Coder Support Package for Infineon AURIX TC4x Microcontrollers
SoC Blockset Support Package for Infineon AURIX Microcontrollers

Description

Write logical status of pins of a GPIO port for the Infineon AURIX TC4x Microcontrollers.

Ports

Input

Pin(s) — Status of GPIO pin(s)
`scalar` | `vector`

Specify a value at this port to set the status of GPIO pin(s). You can specify a nonnegative scalar or a vector of nonnegative elements.

- Specify `0` to set the pin(s) to a low value.
- Specify a `nonzero` value to set the pin(s) to a high value.

The size of input ports available depends on the **Number of pins** parameter.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Output

msg — SoC Blockset simulation message
`scalar`

The Digital Port Write block converts the data received at input port into SoC Blockset message form and the **msg** port outputs that message during simulation.

Dependencies

To view this port, select the **Enable simulation port** parameter and set the port type as **SoC Data (Message based)**.

sig — Simulink based simulation signal
`scalar` | `vector`

This port outputs the data received at the input port during simulation.

Dependencies

To view this port, select the **Enable simulation port** parameter and set the port type as **Signal**.

Parameters

Number of pins — Number of pins you want to configure
`1` (default) | `2` | `3` | `...`

Select the number of pins (1 to 16) whose output you want to configure in the block.

Enable simulation port — Enable peripheral simulation ports in the block
`off` (default) | `on`

Select this parameter to enable peripheral simulation capability.

Note If you do not have a SoC Blockset license, the **Enable simulation port** parameter is visible but you cannot enable it.

Port Type — Specify the type of peripheral simulation port
`SoC Data (Message based)` (default) | `Signal`

Select this parameter to configure either the SoC Blockset compatible or the Simulink signal based simulation ports to enable peripheral simulation capability.

Dependencies

To view this parameter, select the **Enable simulation port** parameter.

Version History

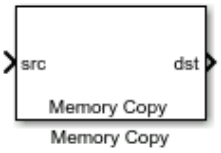
Introduced in R2022b

See Also

Digital Port Read | Digital Port Write Peripheral Configuration

Memory Copy

Copy to and from memory section



Libraries:
Simulink Support Package for Arduino Hardware / Utilities
Embedded Coder Support Package for STMicroelectronics Discovery Boards/Utilities
Simulink Coder Support Package for STMicroelectronics Nucleo Boards/ Utilities
Embedded Coder Support Package for Infineon AURIX TC4x Microcontrollers/Utilities

Description

In generated code, the Memory Copy block copies variables or data to and from processor memory as you have configured with block parameters. Your model can contain as many of these blocks as you require to manipulate memory on your processor.

Each block works with one variable, address, or set of addresses provided to the block. Using the parameters of this block you specify the source and the destination for the memory copy and options for initializing the memory locations.

At run time, you can change options like the memory stride and offset. You can write to memory at program initialization, at program termination, and at every sample time by selecting various parameters in the block. The initialization process occurs once, rather than occurring for every read and write operation.

With the custom source code options, the block enables you to add custom ANSI® C source code before and after each memory read and write (copy) operation.

Block Operations

This block performs operations at three periods during program execution: initialization, real-time operations, and termination. With the options for setting the memory initialization and termination, you control when and how the block initializes memory, copies to and from memory, and terminates memory operations. The parameters enable you to turn on and off memory operations in the three periods, independently of each other.

Copy Memory

When you use this block to copy an individual data element from the source to the destination, the block copies the element from the source in the source data type, and then casts the data element to the destination data type that you specify in the block parameters.

Input/Output Ports

Input

src — Input signal
scalar | vector

When reading the data from this input port, set the **Copy from** parameter to `Input port`. When you choose this option, an input port becomes available. The port label on the block changes to `src`, indicating that the block expects data to come from the input port.

To inherit data type from the incoming signal, set the source **Data Type** parameter to `Inherit from input port`. Specify the source **Offset** parameter as `0` and the source **Stride** parameter as `1`. If you specify any other value for these parameters, you receive an error message that the specified values are invalid.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

&src — Start address of source memory location
scalar

When reading the source address from this input port, set the **Copy from** parameter to `Specified address` and set the source **Specify address source** parameter to `Input port`. When you choose these options, an input port becomes available. The port label on the block changes to `&src`, indicating that the block expects the address to come from the input port. The ability to change the address dynamically means that you can the block to copy different variables by providing the variable address from an upstream block in your model.

When reading the address from the input port, change the source **Data type** from the default `Inherit from input port` to one of the data types in the **Data type** list. If you do not change the parameter value, you receive an error message. The error message indicates that the data type cannot be inherited because the input port does not exist.

Data Types: `uint32`

src ofs — Number of memory locations added to the start address before reading data
scalar

When reading the offset value of data from this input port, set the source **Specify offset source** parameter to `Input port`. The source **Specify offset source** parameter appears only when you select the **Use offset when reading** parameter.

When you choose this option, an input port becomes available. The port label on the block changes to `src ofs`, indicating that the block expects the offset value to come from the input port. This port enables your program to change the offset dynamically during execution by providing the offset value as an input to the block.

When reading the offset value from the input port, do not set the **Copy from** parameter to `Input port`. If you choose this option, you receive an error message that you cannot specify the source offset parameter as an input port when copying data from the input port.

Data Types: `uint32`

&dst — Start address of destination memory location
scalar

When reading the start address of the destination memory location from this input port, set the **Copy to** parameter to `Specified address` and set the destination **Specify address source** parameter to `Input port`. When you choose this option, an input port becomes available. The port label on the block changes to `&dst`, indicating that the block expects the address to come from the input port. The ability to change the address dynamically means that you can the block to copy different variables by providing the variable address from an upstream block in your model.

Data Types: `uint32`

dst ofs — Number of memory locations added to the start address before writing data
scalar

When reading the offset value of the destination memory location from this input port, set the destination **Specify offset source** parameter to `Input port`. The destination **Specify offset source** parameter appears only when you select the **Use offset when reading** parameter.

When you choose this option, an input port becomes available. The port label on the block changes to `dst ofs`, indicating that the block expects the offset value to come from the input port. This port enables your program to change the offset dynamically during execution by providing the offset value as an input to the block.

When reading the offset from the input port, do not set the **Copy to** parameter to `Output port`. If you choose this option, you receive an error message that cannot specify the destination offset parameter as an input port when copying data to output port.

Data Types: `uint32`

Output

dst — Output signal
scalar | vector

When writing data to this output port, set the **Copy to** parameter to `Output port`. When you choose this option, an output port becomes available. The port label on the block changes to `dst`, indicating that the block writes data to the output port.

To inherit data type from the incoming signal, set the **Data Type** parameter to `Inherit from source`. Specify the destination **Offset** parameter as `0` and the destination **Stride** parameter as `1`. If you specify any other value for these parameters, you receive an error message that the specified values are invalid.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Parameters

The Memory Copy dialog box contains multiple tabs:

- **Source** — Identifies the sequential memory location to copy from. Specify the data type, size, and other attributes of the source variable.
- **Destination** — Specify the memory location to copy the source to and specify the attributes of the destination.
- **Options** — Select various parameters to control the copy process.

Source

Copy from — Source from where the block reads data
`Input port` (default) | `Specified address` | `Specified source code symbol`

- `Input port` — An input port becomes available. The port label on the block changes to `src`, indicating that the block expects data to come from the input port.

- `Specified address` — The block reads the source address from the option that you select from the **Specify address source** parameter.

  If you set the **Specify address source** parameter to `Specify via dialog`, the block reads the source address that you specify in the **Address** parameter.

  If you set the **Specify address source** parameter to `Input Port`, an input port becomes available. The port label on the block changes to `&src`, indicating that the block expects the address to come from the input port
- `Specified source code symbol` — The block reads data from the symbol (variable) that you specify in the **Source code symbol** parameter.

If you select `Specified address` or `Specified source code symbol` for the **Copy from** parameter, change **Data type** from the default `Inherit from input port` to one of the data types on the **Data type** list.

Specify address source — Source of memory location to read data
`Specify via dialog` (default) | `Input port`

- `Specify via dialog` — The block reads data from the memory location that you specify in the **Address** parameter.
- `Input port` — An input port becomes available. The port label on the block changes to `&src`, indicating that the block expects the source address to come from the input port. The ability to change the address dynamically means that you can the block to copy different variables by providing the variable address from an upstream block in your model.

Dependencies

This parameter appears only when you set the **Copy from** parameter to `Specified address`.

Source code symbol — Symbol (variable) that contains the source address
`myVariableSrc` (default) | `Any defined symbol`

The block does not verify whether the symbol is valid. A symbol is valid only if it has an entry in the symbol table. If it is not valid you get an error when you try to compile, link, and run your generated code.

If the symbol contains a scalar value, specify & before the symbol in this parameter. If the symbol contains a vector value, specify only the symbol in this parameter.

Dependencies

This parameter appears only when you set the **Copy from** parameter to `Specified source code symbol`.

Address — Memory location of input data
`hex2dec('00001000')` (default)

Specify the address in a decimal format or as a hexadecimal string with single quotation marks. When you specify the address as a hexadecimal string, pass that string as a parameter to the `hex2dec` function. This function converts the string from the hexadecimal format to the decimal format. The following example converts `0x1000` to its decimal form.

```
4096 = hex2dec('1000');
```

For this example, you can specify either `4096` or `hex2dec('1000')` as the address.

Dependencies

This parameter appears only when you set the **Copy from** parameter to `Specified address` and set the **Specify address source** to `Specify via dialog`.

Data type — Type of input data
`Inherit from input port` (default) | `double` | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32` | `boolean`

Use this parameter to specify the data type of your input data. When you set this parameter to `Inherit from input port`, the block inherits the data type from the value at the block input port.

Note If you select the **Copy from** parameter to `Specified address` or `Specified source code symbol`, change the **Data type** parameter from the default `Inherit from input port` to one of the data types on the **Data type** list.

Data length — Number of elements to copy
`1` (default) | `Any integer value greater than 1`

Use offset when reading — Specify an offset when reading data
off (default) | on

When you select this parameter, the block reads the source offset from the option that you select from the **Specify offset source** parameter. When you clear this parameter, the block reads data from the start address that you specify in the **Copy from** parameter.

Specify offset source — Source of offset for input data
`Specify via dialog` (default) | `Input port`

- `Specify via dialog` — The block reads the offset value that you specify in the **Offset** parameter.
- `Input port` — An input port becomes available. The port label on the block changes to `src ofs`, indicating that the block expects the offset value to come from the input port. This port enables your program to change the offset dynamically during execution by providing the offset value as an input to the block.

Dependencies

This parameter appears only when you select the **Use offset when reading** parameter.

Offset — Number of memory locations added to the start address before starting to read data
`0` (default) | `Any integer value greater than 0`

An **Offset** instructs the block whether to copy the first element of data at the input address, or skip values before starting to copy the input to the destination. The **Offset** defines how many input elements to skip before copying the first value to the destination.

Dependencies

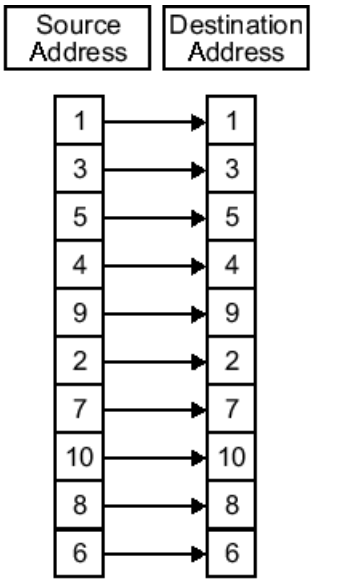
This parameter appears only when you select the **Use offset when reading** parameter.

Stride — Spacing between the elements when reading input data
`1` (default) | `Any integer value greater than 1`

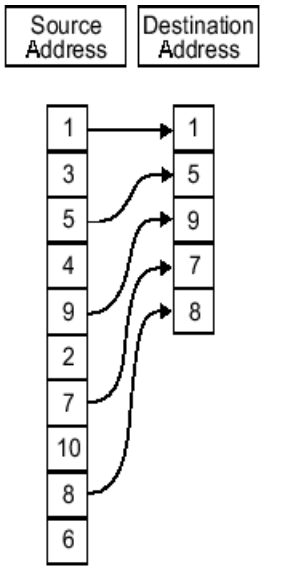
Specify spacing between the elements when reading the input data. A stride value equal to 1 means that the generated code reads the input data sequentially. When you add a stride value that is more than 1, the block skips that number of elements and reads the elements from the resulting address.

The next two figures illustrate the stride concept. In the first figure, you see data copied without a stride.

The second figure shows a stride value equal to 2. A stride value 2 is applied when the block is copying the input to an output location. You can specify a stride value for the output from the **Destination** tab by using the **Stride** parameter.



Input Stride = 1
Output Stride = 1
Number of Elements Copied = 10



Input Stride = 2
Output Stride = 1
Number of Elements Copied = 5

Destination

Copy to — Source to where the block writes data
`Input port` (default) | `Specified address` | `Specified source code symbol`

- `Output port` — An output port becomes available. The port label on the block changes to `dst`, indicating that the block writes data to the output port.
- `Specified address` — The block copies data to the memory location that you that you specify in the **Specify address source** parameter.

  If you set the **Specify address source** parameter to `Specify via dialog`, the block reads the destination address that you specify in the **Address** parameter.

  If you set the **Specify address source** parameter to `Input Port`, an input port becomes available. The port label on the block changes to `&dst`, indicating that the block expects the destination address to come from the input port
- `Specified source code symbol` — The block copies data to the symbol (variable) that you specify in the **Source code symbol** parameter.

Specify address source — Source of memory location to write data
`Specify via dialog` (default) | `Input port`

- `Specify via dialog` — The block writes data to the location that you specify in the **Address** parameter.
- `Input port` — An input port becomes available. The port label on the block changes to `&dst`, indicating that the block expects the destination address to come from the input port. The ability to change the address dynamically means that you can the block to copy different variables by providing the variable address from an upstream block in your model.

Dependencies

This parameter appears only when you set the **Copy to** parameter to `Specified address`.

Source code symbol — Symbol (variable) that contains the destination address
`myVariableDst` (default) | `Any defined symbol`

The block does not verify whether the symbol is valid. A symbol is valid only if it has an entry in the symbol table. If it is not valid you get an error when you try to compile, link, and run your generated code.

If the symbol contains a scalar value, specify & before the symbol in this parameter. If the symbol contains a vector value, specify only the symbol in this parameter.

Dependencies

This parameter appears only when you set the **Copy to** parameter to `Specified source code symbol`.

Address — Memory location where the block writes data
`hex2dec('00001000')` (default)

Specify the address in a decimal format or as a hexadecimal string with single quotation marks. When you specify the address as a hexadecimal string, pass that vector as a parameter to the `hex2dec` function. This function converts the string from hexadecimal format to decimal format. The following example converts `0x2000` to decimal form.

```
8192 = hex2dec('2000');
```

For this example, you can specify either `8192` or `hex2dec('2000')` as the address.

Dependencies

This parameter appears only when you set the **Copy to** parameter to `Specified address` and set the **Specify address source** to `Specify via dialog`.

Data type — Type of output data
`uint32` (default) | `double` | `single` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `boolean` | `Inherit from source`

Use this parameter to specify the data type of your output. When you set this parameter to `Inherit from source`, the block inherits the data type from the input signal at the block input port.

Use offset when writing — Specify an offset when writing data
off (default) | on

When you select this parameter, the block reads the offset from the option that you select from the **Specify offset source** parameter. When you clear this parameter, the block reads data from the start address from the option that you specify in the **Copy to** parameter.

Specify offset source — Source of offset for writing data
`Specify via dialog` (default) | `Input port`

- `Specify via dialog` — The block reads the offset value that you specify in the **Offset** parameter.

- `Input port` — An input port becomes available. The port label on the block changes to `dst ofs`, indicating that the block expects the offset value to come from the input port. This port enables your program to change the offset dynamically during execution by providing the offset value as an input to the block.

Dependencies

This parameter appears only when you select the **Use offset when writing** parameter.

Offset — Number of memory locations added to the start address before starting to write data
`0` (default) | `Any integer value greater than 0`

An **Offset** instructs the block whether to write the first element of data to the start address of destination, or skip one or more locations at the destination before writing the output. The **Offset** defines how many values to skip in the destination before writing the first value to the destination.

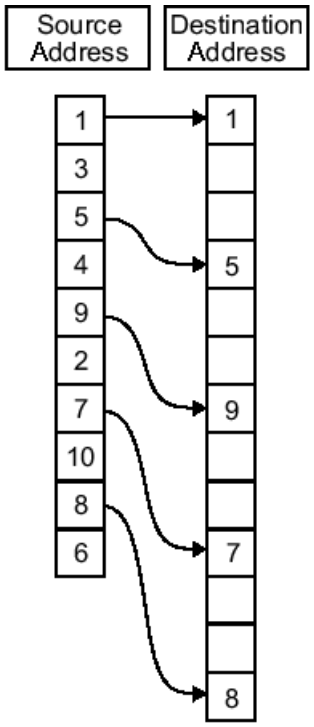
Dependencies

This parameter appears only when you select the **Use offset when writing** parameter.

Stride — Spacing between the elements when writing data
`1` (default) | `Any integer value greater than 1`

Specify the spacing between the elements when writing the output data. A stride value equal to one means that the generated code writes the input data sequentially to the destination in consecutive locations. When you add a stride value that is more than 1, the output data is stored not sequentially, but skips elements equal to the stride.

This figure shows a stride value of 3 applied to writing the input to an output location. You can specify a stride value for the input from the **Source** tab by using the **Stride** parameter. As shown in the figure, you can apply input and output stride at the same time to manipulate your memory more fully.



Input Stride = 2
Output Stride = 3
Number of Elements Copied = 5

Sample time — The rate, in seconds, at which memory copy operation occurs
`Inf` (default) | `-1` | `Any value greater than 0`

The default value `Inf` instructs the block to use a constant sample time. When you specify the **Sample time** as `-1`, the Memory Copy block inherits the sample time from the input. When there are no block inputs, the block inherits the sample time from the Simulink model.

Options

Set memory value at initialization — Specify a value and its source for initializing the destination memory location
off (default) | on

When you select this option, you direct the block to initialize the memory location to a specific value when you initialize your program at run time. After you select this option, use the **Specify initialization value source** parameter to set the value that you want. Alternately, you can direct the block to get the initial value from the block input.

If you select this parameter, set the destination **Copy to** parameter to either `Specified address` or `Specified source code symbol`. If you do not change the parameter value, you receive an error message that memory cannot be set at initialization when the destination is output port.

Specify initialization value source — Specify the source of the initial value
`Specify constant value` (default) | `Specify source code symbol`

- `Specify constant value` — Set a single value to use when your program initializes memory.

  The block reads the initialization value from the value that you specify in the **Initialization value (constant)** parameter.
- `Specify source code symbol` — Specifies a variable (a symbol) to use for the initial value.

  The block reads the initialization value from the symbol that you specify in the **Initialization value (source code symbol)** parameter. The block does not verify whether the symbol is valid. A symbol is valid only if it has an entry in the symbol table. If it is not valid, you get an error when you try to compile, link, and run your generated code.

Dependencies

This parameter appears only when you select the **Set memory value at initialization** parameter.

Initialization value (constant) — A constant value to initialize the memory location when you initialize your program at run time
`1` (default) | `Any value`

Dependencies

This parameter appears only when you select the **Set memory value at initialization** parameter and set the **Specify initialization value source** to `Specify constant value`.

Initialization value (source code symbol) — A defined symbol (variable) that contains initialization value
`myInitValueVariable` (default) | `Any defined symbol`

When you specify a symbol, the block does not verify whether the symbol is valid. A symbol is valid only if it has an entry in the symbol table. If it is not valid, you get an error when you try to compile, link, and run your generated code.

Dependencies

This parameter appears only when you select the **Set memory value at initialization** parameter and set the **Specify initialization value source** to `Specify source code symbol`.

Apply initialization value as mask — Specify the initialization value as mask
off (default) | on

The initialization value is treated as a string of bits for the mask.

The block performs the bitwise operation on the initialization value to manipulate register contents at the bit level. The operator that you specify in the **Bitwise operator** parameter determines the bitwise operation.

Note To use your initialization value as a mask, set the destination **Copy to** parameter to either `Specified address` or `Specified source code symbol`.

Dependencies

This parameter appears only when you select the **Set memory value at initialization** parameter.

Bitwise operator — Specify bitwise operator for manipulating memory value in the register
`bitwise AND` (default) | `bitwise OR` | `bitwise exclusive OR` | `left shift` | `right shift`

To use the initialization value as a mask, from the **Bitwise operator** list, select one of the entries listed in this table.

Bitwise Operator List Entry	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 to the 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 by 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 by 6 bits. In this case, the value must be a positive integer.

Dependencies

This parameter appears only when you select the **Apply initialization value as mask** parameter.

Set memory value at termination — Specify a value of the target memory after the write operation
off (default) | on

This parameter initializes memory when the program starts to access the target memory location and directs the program to set memory to a specific value when the program terminates.

Termination value — A constant value of the target memory after the write operation
1 (default) | `Any value`

This parameter appears only when you select the **Set memory value at termination** parameter.

Set memory value only at initialization/termination — Specify the periods of program execution at which the block performs copy operation
off (default) | on

This Memory Copy block performs operations at three periods during program execution: initialization, real-time operations, and termination. When you select this parameter, the block performs only the memory initialization and termination operations. It does not copy data during real-time operations.

Insert custom code before memory write — Specify a custom ANSI C code that you want to insert into the generated code before the block writes to a specified memory location
off (default) | on

The block inserts a custom ANSI C code that you specify in the **Custom code** parameter before the write operation.

You can use the custom code capability to lock and unlock registers before and after accessing them. For example, some processors have registers that you want to unlock with the EALLOW macro before your program accesses them.

Custom code — A custom ANSI C code that you insert into the generated code before the block writes to a specified memory location
`/* Custom Code Before Write*/` (default)

The code that you specify in this parameter is the code that appears in the generated code.

Dependencies

To enable this parameter, select the **Insert custom code before memory write** parameter.

Insert custom code after memory write — Specify a custom ANSI C code that you insert into the generated code after the block writes to the specified memory location
off (default) | on

The block inserts a custom ANSI C code that you specify in the **Custom code** parameter after the write operation.

You can use the custom code capability to lock and unlock registers before and after accessing them. For example, some processors have registers that you want to lock with the EDIS macro after your program accesses them.

Custom code — A custom ANSI C code that you insert into the generated code after the block writes to the specified memory location
`/* Custom Code After Write*/` (default)

The code that you specify in this parameter is the code that appears in the generated code.

Dependencies

This parameter appears only when you select the **Insert custom code after memory write** parameter.

More About

Symbol Table

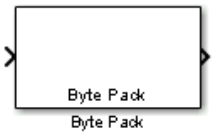
A Symbol table is a data structure that contains entries for all the symbols (variables) that you define in your code.

Version History

Introduced in R2016b

Byte Pack

Convert input signals to 8-, 16-, or 32-bit vector



Libraries:
Simulink Support Package for Arduino Hardware / Utilities
Embedded Coder Support Package for STMicroelectronics Discovery Boards/Utilities
Simulink Coder Support Package for STMicroelectronics Nucleo Boards/ Utilities
Embedded Coder Support Package for Texas Instruments C2000 Processors/ Target Communication
Embedded Coder Support Package for Infineon AURIX TC4x Microcontrollers/Utilities

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.

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

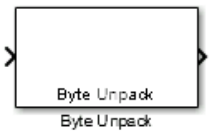
Introduced in R2016b

See Also

Byte Unpack | Byte Reversal

Byte Unpack

Unpack 8-, 16-, or 32-bit input vector to multiple output vectors



Libraries:
Simulink Support Package for Arduino Hardware / Utilities
Embedded Coder Support Package for STMicroelectronics Discovery Boards/Utilities
Simulink Coder Support Package for STMicroelectronics Nucleo Boards/ Utilities
Embedded Coder Support Package for Texas Instruments C2000 Processors/ Target Communication
Embedded Coder Support Package for Infineon AURIX TC4x Microcontrollers/Utilities

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

Unpacked Signals				
Dimension	**Size**	**Data Type**	**Dec Value**	**Hex Value**
			4	04
			–3	FD
Vector	2	`int16`	218	00DA
			–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

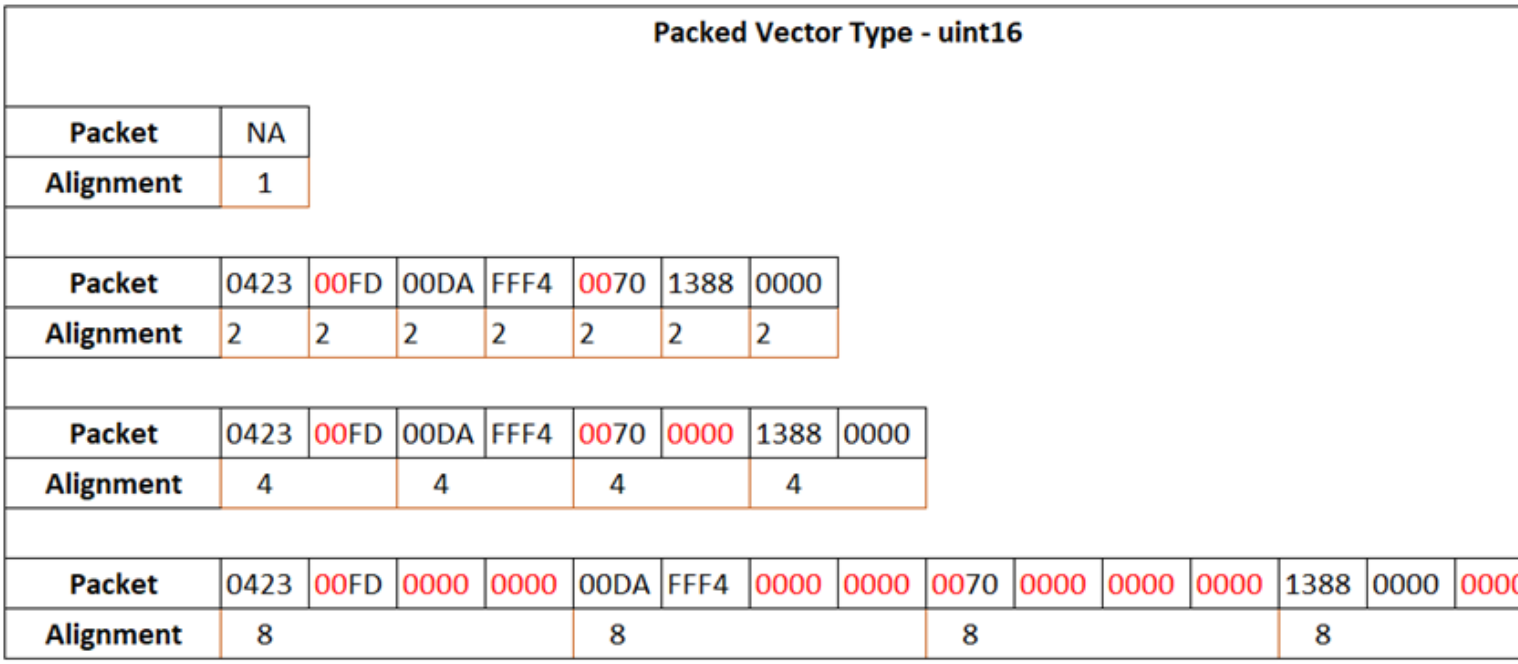
Packet	0023	0004	00FD	0000	00DA	0000	00F4	00FF	0070	0000	0088	0013	0000	0000
Alignment	2		2		2		2		2		2		2	

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

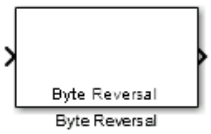
Introduced in R2016b

See Also

Byte Pack | Byte Reversal

Byte Reversal

Reverse little-endian data for big-endian processor



Libraries:
Simulink Support Package for Arduino Hardware / Utilities
Embedded Coder Support Package for STMicroelectronics Discovery Boards/Utilities
Simulink Coder Support Package for STMicroelectronics Nucleo Boards/ Utilities
Embedded Coder Support Package for Infineon AURIX TC4x Microcontrollers/Utilities

Description

The Byte Reversal block changes the order of the bytes in the input data. Use this block when your process communicates between processors that use different endianness. For example, use this block for communication between Intel processors that are little-endian and others that are big-endian. Processors compatible with the Intel 80x86 processors and Texas Instruments processors are little-endian. In this case, insert a Byte Reversal block before the Byte Pack block and another Byte Reversal block just after the Byte Unpack block.

Using the parameter of this block, you specify the number of block inputs.

Input/Output Ports

Input

Port_1 — First of *N* input ports
scalar | vector | matrix

Number of input ports specified in the **Number of inputs** parameter. The block can have 1 to *N* input ports.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Output

Port_1 — Output port containing reversed data
scalar | vector

Output ports from 1 to *N* as specified in the **Number of inputs** parameter. The number of output ports adjusts to match the number of input ports.

Data Types: `single` | `double` | `int8` | `int16` | `int32` | `uint8` | `uint16` | `uint32` | `Boolean`

Parameters

Number of inputs — Number of block inputs
1 (default)

The number of input ports and output ports adjusts to match the value specified in this parameter.

When you use more than one input port, each input port maps to the matching output port. Data entering input port 1 leaves through output port 1, and so on. Reversing the bytes does not change the data type.

Example

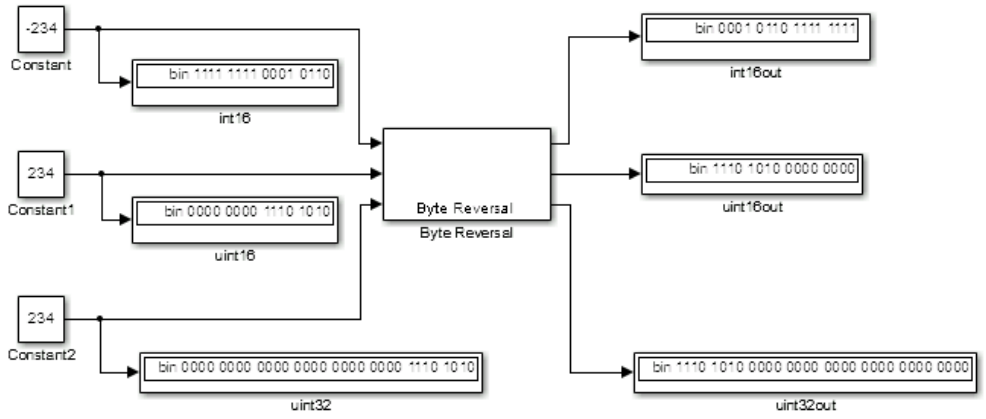
Suppose that you are sending a signal. The signals have the following attributes:

Constant value	Type	Byte structure
-234	`int16`	`1111 1111 0001 0110`
234	`uint16`	`0000 0000 1110 1010`
234	`uint32`	`0000 0000 0000 0000 0000 0000 1110 1010`

1. Set the number of inputs to 3.

   The block creates three input and three output ports.
2. Connect the signals to the Byte Reversal block.

The following model shows byte reversal. In the first signal, with value -234, the low-order byte `0001 0110` is swapped with the high-order byte `1111 1111`. Therefore, the signal outputs to `0001 0110 1111 1111`.



Version History

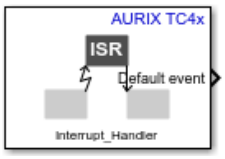
Introduced in R2016b

See Also

Byte Pack | Byte Unpack

Hardware Interrupt

Trigger downstream function-call subsystem from an interrupt service routine



Libraries:
Embedded Coder Support Package for Infineon AURIX TC4x Microcontrollers
SoC Blockset Support Package for Infineon AURIX Microcontrollers

Description

Trigger the downstream function-call subsystem from an interrupt service routine.

Ports

Input

Default event task — Function-call event input simulation
`scalar`

The simulation-only message input acts as pass-through with the output emitted on the block in simulation.

Dependencies

To enable this port, select the **Enable simulation port** parameter.

Output

Default event — Generate interrupt request
`scalar`

The output of this block is a function-call.

Parameters

Simulink task priority — Set priority of selected ISR
`50` (default) | `0-255`

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.

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 peripheral simulation ports in the block
`off` (default) | `on`

Select this parameter to enable peripheral simulation capability.

Note If you do not have a SoC Blockset license, the **Enable simulation port** parameter is visible but you cannot enable it.

Port Type — Specify the type of peripheral simulation port
`SoC Data (Message based)` (default) | `Signal`

Select this parameter to configure either the SoC Blockset compatible or the Simulink signal based simulation ports to enable peripheral simulation capability.

Dependencies

To view this parameter, select the **Enable simulation port** parameter.

Version History

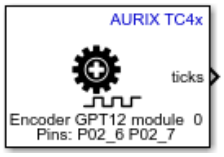
Introduced in R2022b

See Also

Hardware Mapping | “Map Tasks and Peripherals Using Hardware Mapping”

Encoder

Measures rotation of motor in ticks



Libraries:
Embedded Coder Support Package for Infineon AURIX TC4x Microcontrollers
SoC Blockset Support Package for Infineon AURIX Microcontrollers

Description

Measures the rotation of the motor in ticks.

The ticks port outputs the encoder ticks count as an `int32` value.

The **dir** port, when enabled, outputs the direction of the encoder.

- `0` - clockwise direction.
- `1` - counter-clockwise direction.

The Encoder block is developed based on GPT12 (general purpose timer) module available in the AURIX TC4x devices, where timer T3 & T5(slow speed calculation) is used.

Ports

Input

msg — SoC Blockset simulation message
`scalar`

The block converts this input SoC Blockset message into encoder position and outputs it in **ticks** port during simulation.

Dependencies

To view this port, select the **Enable simulation port** parameter and set the port type as **SoC Data (Message based)**.

sig — Simulink based simulation signal
`scalar` | `vector`

The simulation-only Simulink signal input acts as pass-through with the output of the Encoder block during simulation.

Dependencies

To enable this port, select the **Enable simulation port** parameter and set the port type as **Signal**.

offset_in — Enable to add motor offset
`scalar`

The offset_in port enables you to dynamically add the offset in raw position with respective to the motor if required.

Dependencies

To enable this port, select the **Enable offset inport** parameter.

Data Types: `int32`

rate_in — Enable to change data rate
`scalar`

The rate_in ports enables you to change the rate at which the data is sampled at run time in seconds.

Dependencies

To enable this port, select the **Enable refresh rate inport** parameter.

Data Types: `Float32`

Output

ticks — Encoder position in count
`scalar`

The ticks port outputs the encoder position in count as an `int32` value.

Data Types: `int32`

dir — Current motor direction
`scalar`

The dir port outputs the current direction of the motor as one of the following:

- `0` - Clockwise direction
- `1` - Counter clockwise direction

Dependencies

To enable this port, select the **Enable direction port** parameter.

Data Types: `Boolean`

speed — Mechanical speed of motor
`scalar`

The speed port outputs the mechanical speed (in rad/s) of the motor as an `float32` value.

Dependencies

To enable this port, select the **Enable speed port** parameter.

Data Types: `float32`

offset_out — Motor position offset
`scalar`

The offset_out port outputs the offset in raw position with respective to the motor as an **int32** value.

Dependencies

To enable this port, select the **Enable offset outport** parameter.

Data Types: `int32`

rate — Encoder update period
`scalar`

The rate port outputs the rate at which the data is sampled (in seconds) as an `float32` value.

Dependencies

To enable this port, select the **Enable rate outport** parameter.

Data Types: `float32`

Parameters

Inputs

Enable offset inport — Select to enable offset input port
`off` (default) | `on`

Enable this parameter to dynamically add the offset in raw position with respective to the motor.

Enable refresh rate inport — Select to enable refresh rate inport
`off` (default) | `on`

Enable this parameter to refresh the rate at which the data is sampled.

Enable simulation port — Enable peripheral simulation ports in the block
`off` (default) | `on`

Select this parameter to enable peripheral simulation capability.

Note If you do not have a SoC Blockset license, the **Enable simulation port** parameter is visible but you cannot enable it.

Port Type — Specify the type of peripheral simulation port
`SoC Data (Message based)` (default) | `Signal`

Select this parameter to configure either the SoC Blockset compatible or the simulink signal based simulation ports to enable peripheral simulation capability.

Dependencies

To view this parameter, select the **Enable simulation port** parameter.

Outputs

Enable direction port — Select to enable direction output port
`off` (default) | `on`

Enable this parameter to output the current direction of the motor.

Enable speed port — Select to enable speed output port
`off` (default) | `on`

Enable this parameter to output the mechanical speed (in rad/s) of the motor.

Enable offset outport — Select to enable offset output port
`off` (default) | `on`

Enable this parameter to output the offset in raw position with respective to the motor.

Enable refresh rate outport — Select to enable rate output port
`off` (default) | `on`

Enable this parameter to output the rate at which the data is sampled in seconds.

Sample time — Frequency at which block samples the input pin
`0.0001` (default) | `scalar` | `vector`

Specify in seconds how often the block should read the pin value.

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

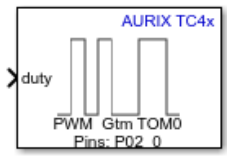
Introduced in R2022b

See Also

Encoder Peripheral Configuration

PWM

Generate pulse width modulated waveforms



Libraries:
Embedded Coder Support Package for Infineon AURIX TC4x Microcontrollers
SoC Blockset Support Package for Infineon AURIX Microcontrollers

Description

The block input controls the duty cycle of the square waveform for the corresponding channel. An input value of `0` produces a `0` percent duty cycle and an input value of `100` produces a `100` percent duty cycle. It accepts `N x 1` values of duty cycles corresponding to the **N** channels selected.

Ports

Input

duty — Input value which determines duty cycle
`Float range between [0-100]`

The value sent to the block input determines the duty cycle of the square wave that the board outputs on the specified PWM pin specified in PWM Peripheral Configuration.

Data Types: `Float`

freq — Input port determines frequency of PWM signal
`scalar`

The freq port determines the frequency of PWM signal for the selected channels. When you enable the frequency port, the port accepts a scalar input value and sets it as the frequency of the timer unit.

Dependencies

To enable this port, select the **Enable frequency input** parameter.

phase — Phase of selected channel
`scalar` | `vector`

The phase port accepts an input value in the range of `0 to 2`pi and sets the phase for the corresponding channel. It accepts `N x 1` values corresponding to the **N** channels selected.

Dependencies

To enable this port, select the **Enable phase input** parameter.

Data Types: `single` | `double`

dtm_rise — Rising dead time module
`scalar` | `vector`

The dtm_rise port accepts the rising dead time values in seconds. The timer unit selected in the PWM peripheral app should have DTM sub module for this feature. It accepts `N x 1` values corresponding to the **N** channels selected.

Dependencies

To enable this port, select the **Enable dead time properties input** parameter.

Data Types: `Float` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

dtm_fall — Falling dead time module
`scalar` | `vector`

The dtm_fall port accepts the falling dead time values in seconds. The timer unit selected in the PWM peripheral app should have DTM sub module for this feature. It accepts `N x 1` values corresponding to the **N** channels selected.

Dependencies

To enable this port, select the **Enable dead time properties input** parameter.

Data Types: `Float` | `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

Output

Status — Status of the PWM module
`scalar`

The block outputs the status of the PWM module as one of these values:

- `255`–Unknown state
- `0`– Initialized
- `1`– Running
- `2`– Stopped
- `3`– Error

Dependencies

To view this port, select the **Enable status output** parameter.

Data Types: `uint8`

msg — SoC Blockset simulation message
`bus`

The port outputs the PWM bus object in the SoC Blockset simulation message format during simulation. The PWM bus object contains the information received by the block at the input ports.

Dependencies

To view this port, select the **Enable simulation port** parameter and set the port type to **SoC Data (Message based)**.

sig — Simulink based simulation signal
`scalar` | `vector`

The port outputs the PWM bus object during simulation. The PWM bus object contains the information received by the block at the input ports.

Dependencies

To view this port, select the **Enable simulation port** parameter and set the port type to **Signal**.

Parameters

Basic

Enable frequency input — Select to enable input frequency port
`off` (default) | `on`

Select this parameter to set the PWM frequency.

When you select the **Enable frequency input** parameter, the block configures an input port, `freq`.

Enable phase input — Select to enable input phase port
`off` (default) | `on`

Select this parameter to set the phase of selected channel.

When you select the **Enable phase input** parameter, the block configures an input port, `phase`.

Enable status output — Select to configure the status port
`off` (default) | `on`

When you select the **Enable status output** parameter, the block configures `Status` output port.

Enable simulation port — Enable peripheral simulation ports in the block
`off` (default) | `on`

Select this parameter to enable peripheral simulation capability.

Note If you do not have a SoC Blockset license, the **Enable simulation port** parameter is visible but you cannot enable it.

Port Type — Specify the type of peripheral simulation port
`SoC Data (Message based)` (default) | `Signal`

Select this parameter to configure either the SoC Blockset compatible or the Simulink signal based simulation ports to enable peripheral simulation capability.

Dependencies

To view this parameter, select the **Enable simulation port** parameter.

Advanced

Number of channels — Select required channels
`1` (default) | `2` | `3` | `...`

Select the required number of PWM channels to send the duty cycles. The number of channels determines the size of the input ports `phase`, `dtm_rise`, and `dtm_fall`.

The number of channels selected depends on the timer module selection in PWM peripheral configuration app.

- `TOM`—16 channels are available
- `ATOM`—8 channels are available

Enable dead time properties input — Select to enable rising and falling input ports
`off` (default) | `on`

Select this parameter to determine the PWM dead time value in seconds.

For more information on dead time module, see “Dead-Time Module Properties” on page 8-30.

Note Enabling the dead time properties parameter in the block configures the `dtm_rise` and `dtm_fall` input ports. To use the dead time functionality, go to PWM Peripheral Configuration in the Hardware Mapping window and configure the corresponding dead time parameters in **Module** and **Channel** tabs.

Version History

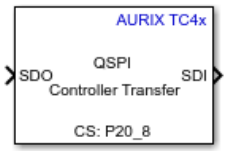
Introduced in R2022b

See Also

PWM Peripheral Configuration

QSPI

Write data to and read data from an SPI peripheral device



Libraries:
Embedded Coder Support Package for Infineon AURIX TC4x Microcontrollers
SoC Blockset Support Package for Infineon AURIX Microcontrollers

Description

Write data to and read data from an SPI peripheral device

The Queued Synchronous Peripheral Interface (QSPI) block accepts 1-D array of data type `int8`, `uint8`, `int16`, `uint16`, `int32`, and `uint32`. The block outputs an array of the same size and data type as the input values.

Ports

Input

SDO — Serial data out
`scalar` | `vector`

The port inputs the data to be communicated with peripheral device. The datatype is based on the block mask configuration.

Dependencies

To enable this port, set the **Transfer mode** parameter to either `SPI Transmit and Receive` or `SPI Transmit`.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

SDImsg — SoC Blockset simulation message
`scalar`

If the **Transfer mode** is set to either `SPI Transmit and Receive` or `SPI Receive`, the QSPI block converts the SoC Blockset message **SDImsg** into output data at the **SDI** output port during simulation.

Dependencies

To enable this port, select the **Enable simulation port** parameter and set the port type as **SoC Data (Message based)**.

SDIsig — Simulink based simulation signal
`scalar` | `vector`

If the **Transfer mode** is set to either `SPI Transmit and Receive` or `SPI Receive`, this input port acts as pass-through with the **SDI** output of QSPI block during simulation.

Dependencies

To enable this port, select the **Enable simulation port** parameter and set the port type as **Signal**.

Output

SDI — Serial data in
`scalar` | `vector`

The port outputs the data received from the peripheral device. The output datatype is based on the block mask configuration.

Dependencies

To enable this port, set the **Transfer mode** parameter to either `SPI Transmit and Receive` or `SPI Receive`.

Data Types: `int8` | `uint8` | `int16` | `uint16` | `int32` | `uint32`

status — Status of QSPI
`scalar` | `vector`

The port outputs the status of the data transfer.

- `0` — QSPI transfer completed
- `1` — QSPI transfer in progress

Dependencies

To enable this port, select the **Enable status port** parameter.

Data Types: `Boolean`

SDOmsg — SoC Blockset simulation message
`scalar`

If the **Transfer mode** is set to either `SPI Transmit and Receive` or `SPI Transmit`, this port outputs the data received at **SDO** port in message form during simulation.

Dependencies

To enable this port, select the **Enable simulation port** parameter and set the port type as **SoC Data (Message based)**.

SDOsig — Simulink based simulation signal
`scalar` | `vector`

If the **Transfer mode** is set to either `SPI Transmit and Receive` or `SPI Transmit`, this port outputs the data received at **SDO** port during simulation.

Dependencies

To enable this port, select the **Enable simulation port** parameter and set the port type as **Signal**.

Parameters

Basic

Transfer mode — Select data type transfer
`SPI Transmit and Receive` (default) | `SPI Transmit` | `SPI Receive`

Select the QSPI transfer mode.

- `SPI Transmit and Receive` - In this transfer mode, the algorithm waits until the data is received before continuing processing/further processing.

  When you select this transfer mode, the block configures both output and input ports, `SDO` and `SDI`.
- `SPI Transmit` - In this transfer mode, you can transmit the data without waiting to receive. Before initiating the data transfer, ensure the status is `1` i.e. not in progress from the QSPI Receive block.

  When you select this transfer mode, the block configures an input port, `SDO`.

  Note QSPI Receive block is responsible to handle the receive data from the peripheral device. In other words both QSPI Transmit block and QSPI Receive block must be used to complete the data transfer.
- `SPI Receive` - In this transfer mode, you can receive the data from peripheral device.

  When you select this transfer mode, the block configures an output port, `SDI`.

  Note The data should be transmitted using QSPI Transmit block before receiving the data. In other words, both QSPI Transmit and QSPI Receive block must be used to complete the data transfer.

Data bits — Select data bits
2 (default) | 5 | 6 | `...`

Data bits parameter provides the frame length of each data element in the QSPI communication. If the selected data bits is in the range:

- `2 to 8` - the valid data types are `int8` or `unit8`.
- `9 to 16` - the valid data types are `int16` or `unit16`.
- `17 to 32` - the valid data types are `int32` or `unit32`.

Enable status port — Enable to configure output status port
`off` (default) | `on`

Select this parameter to output status of data transfer.

When you select the Output Status parameter, the block configures an output port, Status.

Sample time — Frequency at which block reads input data
`-1` (default) | `scalar` | `vector`

Specify in seconds how often the block should read the input data.

When you specify this parameter as `-1`, Simulink determines the best sample time for the block based on the block context within the model.

Dependencies

To enable this parameter, set the **Transfer mode** to the `SPI Receive`

Enable simulation port — Enable peripheral simulation ports in the block
`off` (default) | `on`

Select this parameter to enable peripheral simulation capability.

Note If you do not have a SoC Blockset license, the **Enable simulation port** parameter is visible but you cannot enable it.

Port Type — Specify the type of peripheral simulation port
`SoC Data (Message based)` (default) | `Signal`

Select this parameter to configure either the SoC Blockset compatible or the Simulink signal based simulation ports to enable peripheral simulation capability.

Dependencies

To view this parameter, select the **Enable simulation port** parameter.

Advanced

Output data length — Specify data length to be received
`1` (default) | `positive integer`

Specify the data length to be received from the peripheral device.

Note The data length specified must match the data transmitted using QSPI Transmit block.

Dependencies

To enable this parameter, set the **Transfer mode** to the `SPI Receive`

Output data type — Select output data type
`1` (default) | `positive integer`

Select the supported data types based on the data bits selected.

Dependencies

To enable this parameter, set the **Transfer mode** to either `SPI Receive` or `SPI Transmit and Receive`

Version History

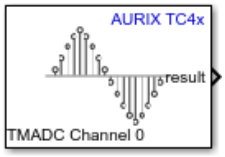
Introduced in R2022b

See Also

QSPI Peripheral Configuration

TMADC

Measure voltage of an analog input pin



Libraries:
Embedded Coder Support Package for Infineon AURIX TC4x Microcontrollers
SoC Blockset Support Package for Infineon AURIX Microcontrollers

Description

Measure the voltage of an analog input pin.

The Time Multiplexed Analog to Digital Converter (TMADC) block outputs the voltage as a 12-bit value and the output port is an array `N x 1` where **N** is defined based on the number of channels selected.

Ports

Input

trigger — Port to trigger the conversion
`scalar`

Triggers the TMADC input pin for conversion via software trigger.

From trigger port when mode is set trigger only enables you to provide the sample time.

- `1`– TMADC start of conversion is triggered
- `0`–TMADC start of conversion is not triggered

Dependencies

To view this port, set the **Mode** parameter to `Trigger only`.

Data Types: `Boolean`

msg — SoC Blockset simulation message
`scalar`

This port receives SoC Blockset message from the interface blocks and the TMADC block converts this message and outputs it at the **result** port during simulation.

Dependencies

To view this input port, set the **Mode** to `Trigger and read` or `Read results only` and select the **Enable simulation port** parameter and set the port type as **SoC Data (Message based)**.

sig — Simulink based simulation signal
`scalar` | `vector`

The simulation-only Simulink signal input port acts as a pass-through for the output of the TMADC block during simulation.

Dependencies

To view this port, set **Mode** to `Trigger and read` or `Read results only` and select the **Enable simulation port** parameter and set the port type to **Signal**.

Output

result — Voltage as 12-bit value
`scalar` | `vector`

The block outputs the voltage as a 12-bit value and the output port is an array `N x 1` where **N** is defined based on the number of channels selected.

Dependencies

To enable this port, set the **Mode** parameter to `Trigger and read` or `Read results only`.

Data Types: `uint16`

status — Status of read operation
`scalar`

The block outputs the status of read operation at this port as one of these values:.

- `0`—Able to read the result after successful trigger for the corresponding channel
- `1`—Unable to read the result after successful trigger for the corresponding channel

Dependencies

To enable this port, set the **Mode** to `Read results only` and select the **Enable status port** parameter.

Data Types: `Boolean`

timestamp — Status of 16-bit counter value
`scalar` | `vector`

The port outputs the status of the 16-bit counter value of timestamp counter.

Dependencies

To enable this port, select the **Enable timestamp** parameter.

Data Types: `Boolean`

event — Event message to start conversion
`scalar`

This port outputs an event message at each trigger to start the conversion during simulation.

Dependencies

To view this port, set the **Mode** to `Trigger only` and select the **Enable simulation port** parameter, and set the port type to **SoC Data (Message based)**.

triggersig — Simulink based trigger signal for simulation
`scalar` | `vector`

This port outputs the data received at **trigger** input port as a signal during simulation.

Dependencies

To view this port, set the **Mode** to `Trigger only` and select the **Enable simulation port** parameter, and set the port type to **Signal**.

Parameters

Mode — Operation mode of the block
`Trigger and read` (default) | `Trigger only` | `Read results only`

Select one of these operation modes:

- `Trigger and read`– Select this option to trigger conversion at every sample time. The block waits until the conversion is complete and outputs the results. This option enables the output port **result**.
- `Trigger only`– Select this option to trigger the selected channel via a software trigger. This option enables the input port **trigger**. As the block does not show the conversion results in this mode, you have to use another TMADC block with the **Mode** parameter set to `Read results only` to display the conversion results.
- `Read results only`– Select this option for the block to read the result at the selected channel. The block outputs `0` at the **status** port if it is able to read the result and `1` otherwise. This option enables the output port **result**.

Number of channels — Select required channels
`1` (default) | `2` | `3` | `...`

If you set the **Mode** to `Trigger and read`, then select the required number of channels ranging between `1` through `16`. The number of channels are limited to 1 if you set the **Mode** to `Trigger only` or `Read results only`.

Enable timestamp — Enable timestamp
`off` (default) | `on`

Select to enable the timestamp.

When you select the `Enable timestamp` parameter, the block configures an additional output port, timestamp similar to result port to read the timestamp.

Note

- TMADC module has 16-bit timestamp counter which resets after every counter overflow.
- Enable timestamp parameter is not available when mode is set to `Trigger only`.

Enable status port — Configure status port
`off` (default) | `on`

When you select this parameter, the block configures the **status** output port. The **status** port outputs the status of the conversion result.

Dependencies

To enable this port, set the **Mode** parameter to `Read results only`.

Sample time — Frequency at which block reads input analog pin(s)
`-1` (default) | `scalar` | `vector`

Specify in seconds how often the block should read the analog pin(s).

When you specify this parameter as `-1`, Simulink determines the best sample time for the block based on the block context within the model.

Dependencies

To enable the **Sample time** parameter, set the **Mode** parameter to either `Trigger and read` or `Read results only`.

Enable simulation port — Enable peripheral simulation ports in the block
`off` (default) | `on`

Select this parameter to enable peripheral simulation capability.

Note If you do not have a SoC Blockset license, the **Enable simulation port** parameter is visible but you cannot enable it.

Port Type — Specify the type of peripheral simulation port
`SoC Data (Message based)` (default) | `Signal`

Select this parameter to configure either the SoC Blockset compatible or the Simulink signal based simulation ports to enable peripheral simulation capability.

Dependencies

To view this parameter, select the **Enable simulation port** parameter.

Version History

Introduced in R2022b

See Also

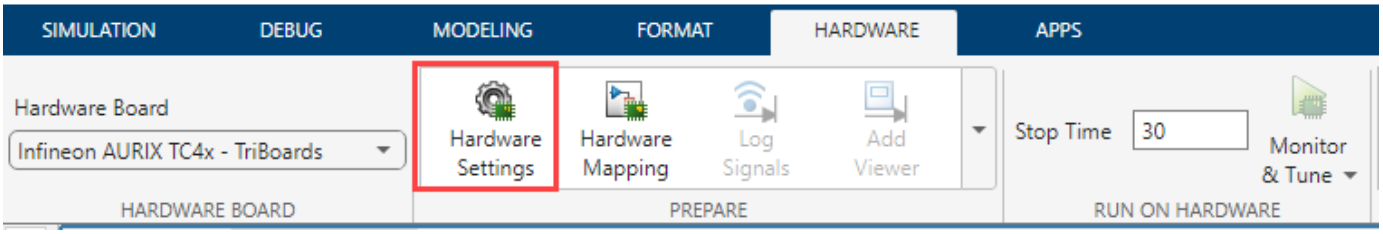
TMADC Peripheral Configuration

Coder Target Context Sensitive Help

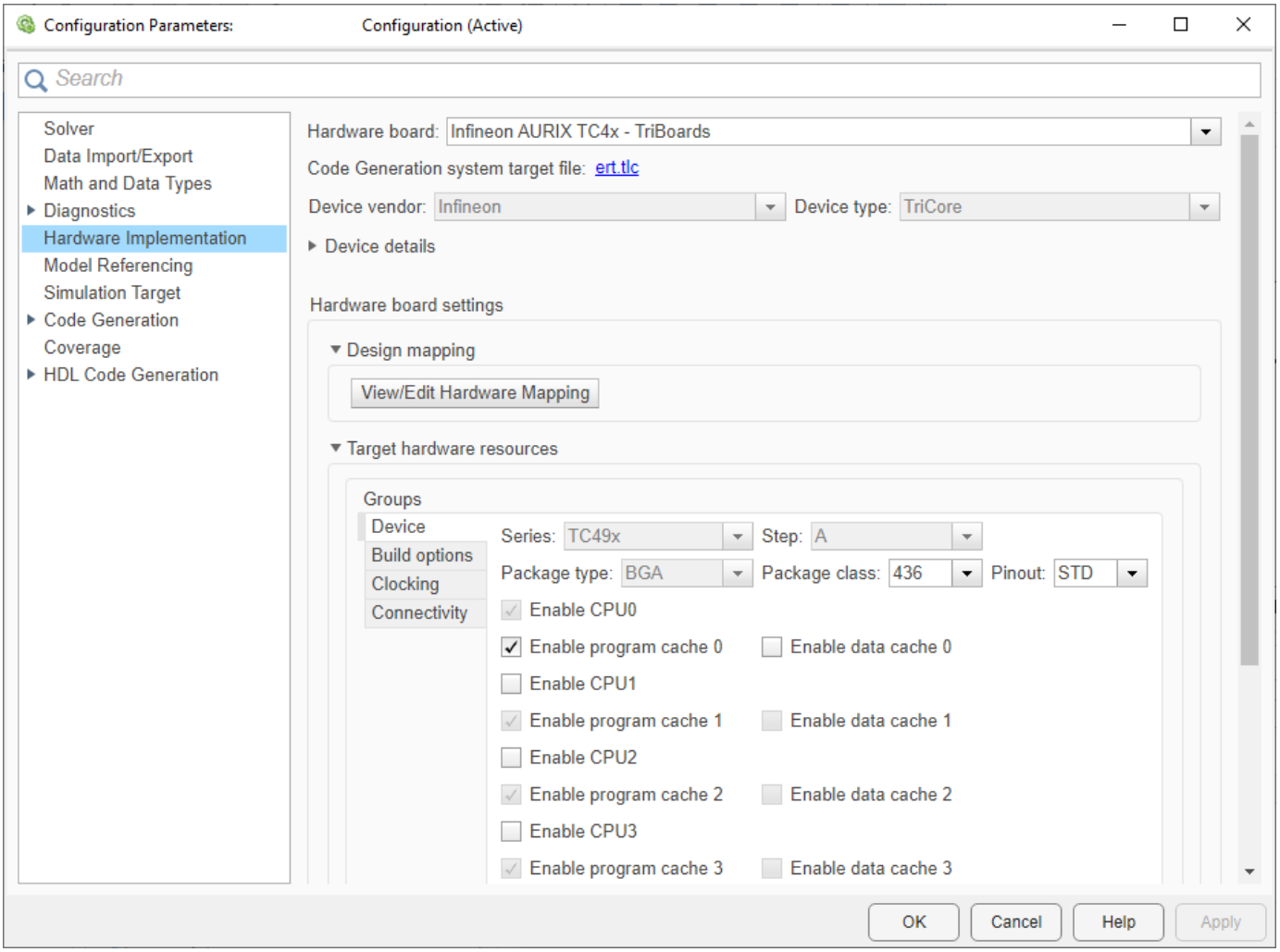
Model Configuration Parameters

Update the configuration parameters for a Simulink model that you create, before simulating or deploying the model to the controller.

In the Simulink window, click **Hardware Settings** in the **HARDWARE** tab to open the Configuration Parameters dialog box and select the target hardware in the **Hardware board** field. You can also press **Ctrl+E** to open the Configuration Parameters dialog box.



Hardware Implementation Pane Overview



Configure hardware board to run Simulink models.

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 `Infineon AURIX TC4x - TriBoards`.
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. To apply the changes, click **Apply**.

Design Mapping

The **Hardware Mapping** tool allows you to configure the software tasks and peripherals on the selected hardware board.

- In this tool, you can map the tasks in your software model to the available event sources and interrupts. The sources of events or interrupts depend on the choice of hardware board and peripherals available in the model.
- In this tool, you can configure the peripheral by setting hardware specific parameters. The available parameters depend on the selected hardware board for the model and the peripheral.

For more information, see

Parameter	Description	Default Value
View/Edit Hardware Mapping	Map tasks and peripherals in a model to hardware board configurations	not applicable

Target hardware resources

Device

Parameter	Description	Default Value
	Displays the device series for the selected hardware board.	TC49x
	Displays the device step for the selected hardware board.	A
	Displays the device packaging type for the selected hardware.	BGA
	Displays the number of programmable pins for the selected hardware board/ microcontrollers. You can set the package class to 436 or 292.	436
	Displays the pinout type for the selected hardware board. You can set the package class to STD or COM.	STD
	Select the CPU.	disabled
	Select the program cache.	on
	Select the data cache.	off

Build Options

Parameter	Description	Default Value
	Define how Embedded Coder responds when you build your model.	Build, load, and run
	Select to compile the generated code and driver source codes in parallel order for faster build and deployment speed.	off
	Select to force rebuild of the static driver library.	off
	Option to specify if the application has to load to the RAM or ROM.	RAM
	Indicates that the custom linker command file must be used during the build action.	enabled
	The path to the memory description file required during linking.	off

Clocking

Parameter	Description	Default Value
	Oscillator frequency used in the processor.	`20`
	CPU clock frequency of the microcontrollers on the target hardware	`400`

Connectivity

Parameter	Description	Default Value
	Select the type of communication interface for simulation.	`Serial (ASCLIN0)`
	Select the serial communication interface module.	`empty`
	Select the baud rate of the serial communication port.	`921600`

Design Mapping

View/Edit Hardware Mapping

Open the tool to map tasks in the model to available hardware interrupt sources for the selected hardware board.

Open the tool to map the simulation parameters of interface blocks to deployed configuration values of peripherals on the selected hardware board.

Device

Series

Displays the device series for the selected hardware board.

Default: TC49x

Step

Displays the device step for the selected hardware board.

Default: A

Package type

Displays the device packaging type for the selected hardware.

Default: BGA

Package class

Displays the number of programmable pins for the selected hardware board/microcontrollers. You can set the package class to 436 or 292.

Settings

Default: 436

Valid options: 292

Pinout

Displays the pinout type for the selected hardware board. You can set the package class to STD or COM.

- STD - Standard pinout supporting legacy designs.
- COM - Communication pinout supporting high speed communication.

Settings

Default: STD

Valid options: COM

Enable CPU#

Select the CPU.

Enable program cache

Select the program cache.

Enable data cache

Select the data cache.

Build Options

Build action

Default: `Build, load and run`

`Build, load and run`

Press `Ctrl+B` or click **Build, load and run**.

1. Generates code from the model.
2. Compiles and links the code into an executable with libraries.
3. Loads the executable and libraries into the hardware board.
4. Runs the executable on the hardware board.

`Build`

Press `Ctrl+B` or click **Build**.

1. Generates code from the model.
2. Compiles and links the code into an executable with libraries.

This option does not load and run the executable on the Infineon AURIX TC4x - TriBoard.

Disable parallel build

Select to compile the generated code in parallel order for faster build.

- `on` – When you select this option, the support package compiles generated code in a sequential order.
- `off` – When you clear the option, the support package compiles generated code parallely. Parallel execution reduces the time taken to build the model.

Default: `off`

Enable force rebuild of static library

Select to force rebuild of the static driver library.

- on – When you select this option, Infineon iLLD library files are built each time you compile the Simulink® model, thus resulting in a slower build.

  Note Ensure to select the **Enable force rebuild of static library** parameter, when you set **Build configurations** to `Specify` and modify `Tool options` in **Configuration Parameters** > **Code Generation** > **Toolchain settings**.

- off – When you clear the selection, Infineon iLLD library files are saved in a static library file. This library file is created only when the Simulink model is compiled for the first time. For all

consecutive compilations, the support package uses the same library file, resulting in a faster build.

Default: `off`

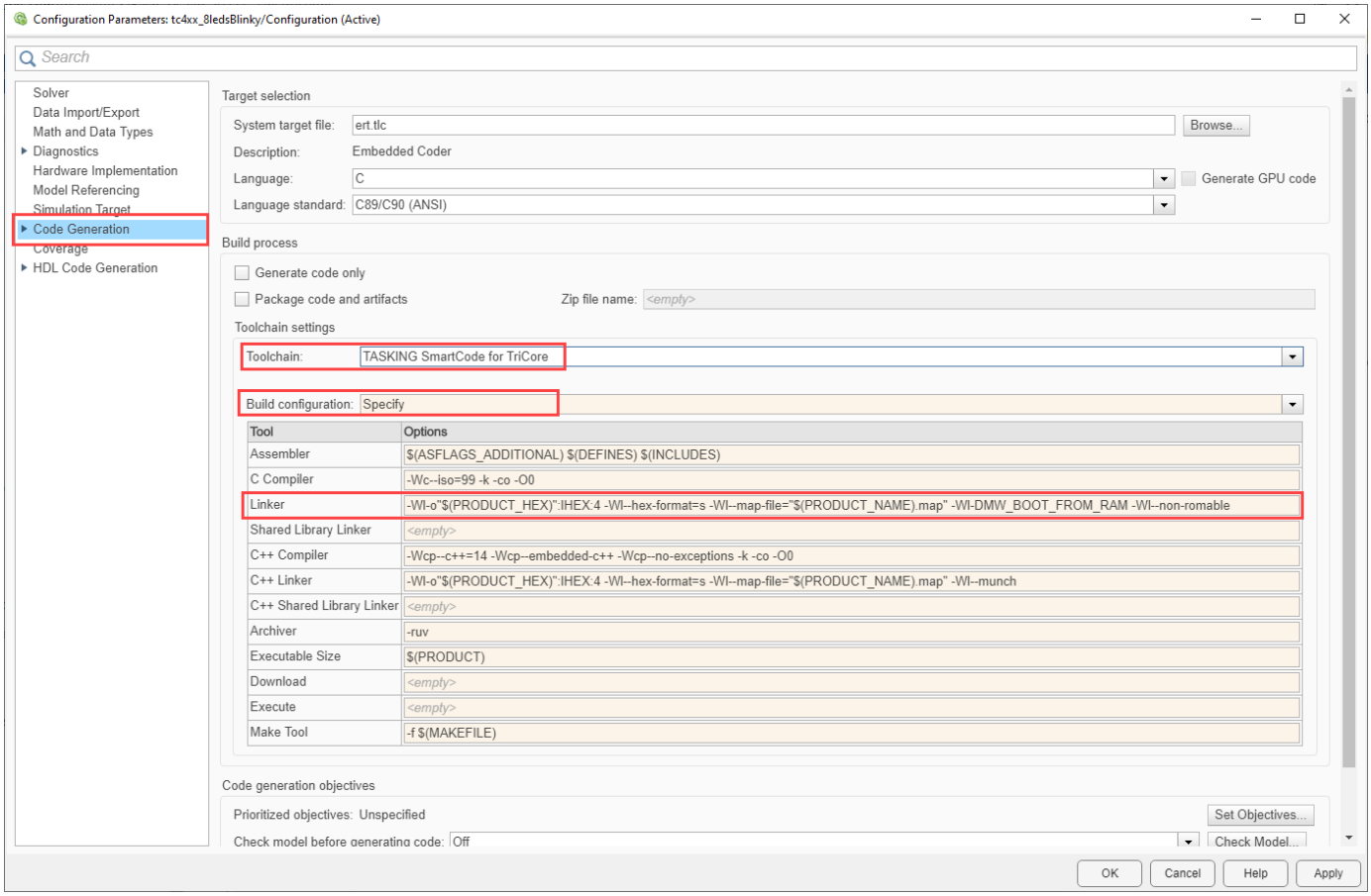
Boot From

How to configure the model to boot from RAM

The option to specify if the application has to load to the RAM. By default, the code is generated for boot from ROM/Flash. To deploy generated code on your hardware, configure the model to boot from RAM.

1. In the Configuration Parameter dialog box, click **Code Generation**.
2. Set the **Build configuration** to `Specify`.
3. In the **Linker** field, add the following linker flag.

   ```
   -Wl-DMW_BOOT_FROM_RAM -Wl--non-romable
   ```



4. Click **Apply** and **OK**.

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 file

For each family of Infineon AURIX TC4x processor selected under **Target hardware resources**, one linker command file is selected automatically.

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.

Clocking

External oscillator (MHz)

Oscillator frequency used in the processor.

CPU Clock (MHz)

This option is for the CPU clock frequency of the microcontrollers on the target hardware.

Connectivity

Connectivity interface

Select the type of communication interface for PIL simulation.

Port

COM port number of the USB Serial Port showing in your host computer. To find the CPM port, browse to **Device Manager > Ports (COM & LPT)**.

Baudrate

Select the baud rate of the serial communication port.

Tools

codertarget.aurix_target.setiLLDPrecompiledFolder()

Set folder path to store precompiled iLLD library

Syntax

```
codertarget.aurix_target.setiLLDPrecompiledFolder(validpath)
codertarget.aurix_target.setiLLDPrecompiledFolder()
```

Description

`codertarget.aurix_target.setiLLDPrecompiledFolder(validpath)` sets the path where precompiled iLLD library will be created.

`codertarget.aurix_target.setiLLDPrecompiledFolder()` if the folder path is not provided, the precompiled iLLD library is created in the OS specific`temporary directory`.

Examples

Create library files in specified folder path

Create a precompiled iLLD library in the specified path, when you click **Build** in Simulink. For more information, see

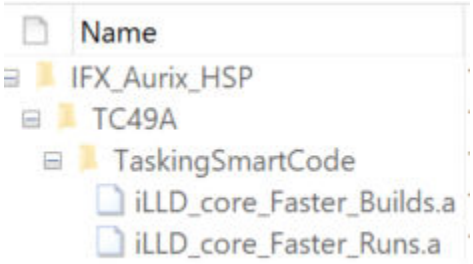
Specify the valid folder path with write privileges where you want the precompiled iLLD files to be placed.

Consider that you want the library in the following path `C:\Users\<username>\Desktop\Infineon_library`.

To do this, enter the following command at the MATLAB command prompt:

```
codertarget.aurix_target.setiLLDPrecompiledFolder(C:\Users\<username>\
Desktop\Infineon_library)
```

To build the Simulink model, press **Ctrl+B** or click **Build**. Once you have built the model, the files are copied to the folder you specified in MATLAB. The folder structure, for example, looks like this.



- The folder name `TC49A` represents the device **Series** and **Step** name provided in the Configuration Parameters (**Model Configuration Parameters > Hardware Implementation > Hardware board > Device**).
- The folder name `TaskingSmartCode` represents the **Toolchain** selected for code generation (**Model Configuration Parameters > Code Generation > Toolchain settings**).
- The file `iLLD_core_FASTER_Builds.a` represents the **Build configuration** option selected for code generation (**Model Configuration Parameters > Code Generation > Toolchain settings**).

Create library files in temporary directory

Create precompiled iLLD library in the temporary directory, when you click **Build** in Simulink.

If you don’t specify the folder path where you want the precompiled iLLD library files to be placed, then the library files are placed in the `tempdir`.

Consider that you did not provide the folder path, then the folder is placed in the following path: `C:\Users\<username>\AppData\Local\Temp`.

If this command is not executed, then the library files are placed in the `tempdir`.

You can also find the `tempdir` by executing the following command:

```
>>tempdir
```

To create library files in a temporary directory, enter the following command at the MATLAB command prompt:

```
codertarget.aurix_target.setiLLDPrecompiledFolder()
```

Input Arguments

`validpath` — Valid precompiled folder path

character vector

Valid path for placing the precompiled iLLD library.

Version History

Introduced in R2022b

See Also

Tasks

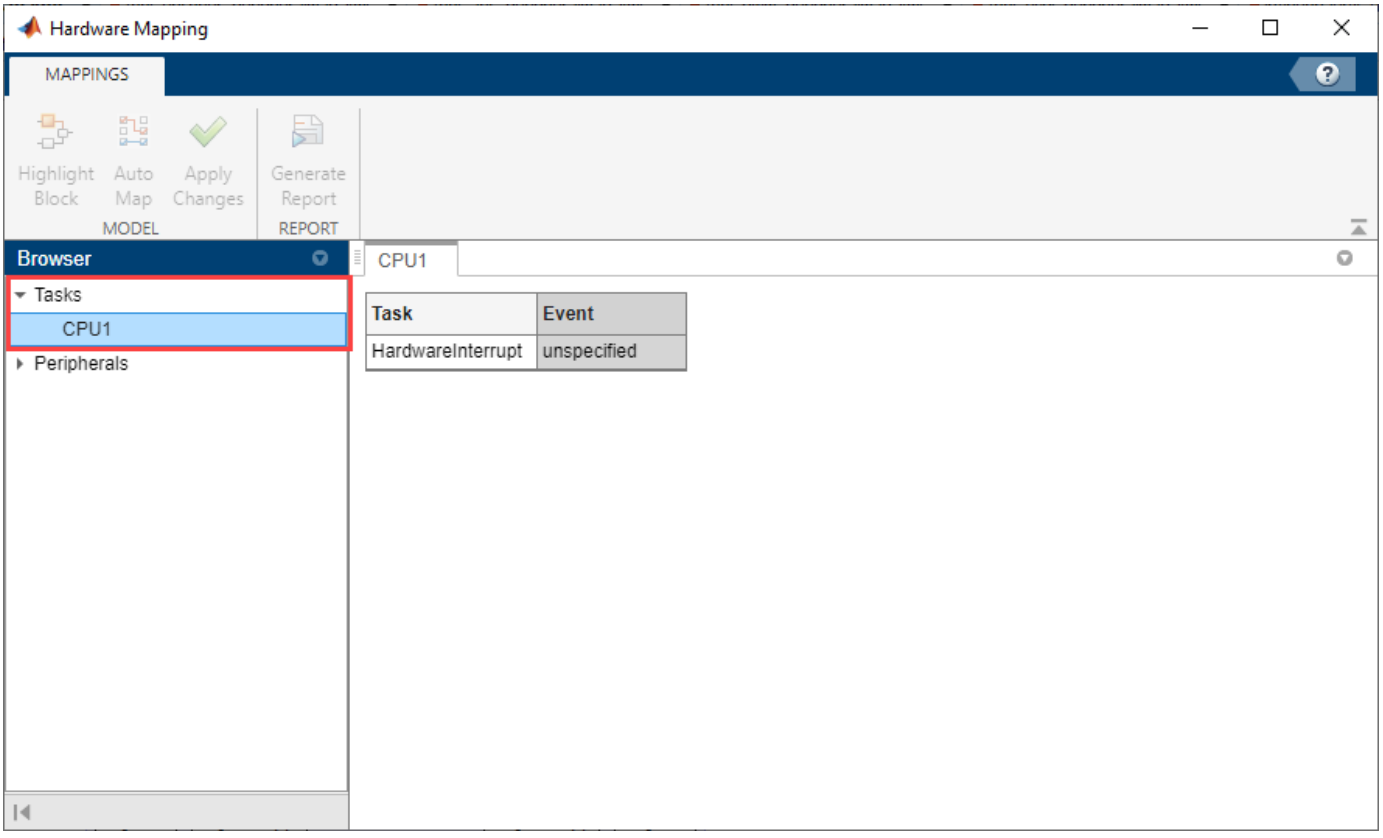
Map tasks in the Infineon AURIX to interrupt service routines on the hardware board

Description

View and edit the map of tasks in the interrupt service routines (ISRs) on the hardware board.

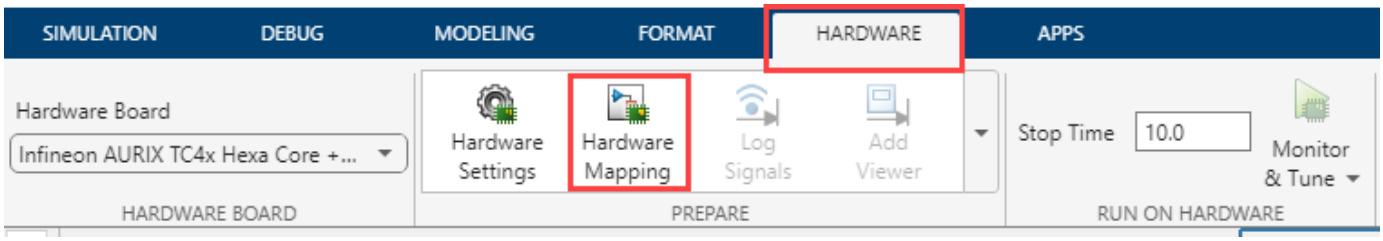
In this tool, you can map the tasks in your software model to the available event sources and interrupts:

- Manually select the task in **Browser** > **Tasks** > ***CPU1***. Select the desired event or interrupt. source. Click the **Apply Changes** button in the toolstrip.
- The sources of events or interrupts depend on the choice of hardware board and peripherals available in the model.



Open the Tasks

- In the **Hardware** tab, click **Hardware Mapping**.



Parameters

CPU1

`Task` — Hardware interrupt
`Hardware interrupt`

This parameter is read-only.

This parameter lists all the event source available for hardware interrupt peripheral.

`Event` — Event source or interrupts
`Unspecified` (default) | `Interrupt name`

The sources of events or interrupts depend on the choice of hardware board and peripherals available in the model. For example, the event `PWM_Periodinterrupt_GTMTOM_2` is available when you select **period event** in PWM peripheral block.

Version History

Introduced in R2022b

See Also

| “Map Tasks and Peripherals Using Hardware Mapping” | Hardware Mapping

TMADC Peripheral Configuration

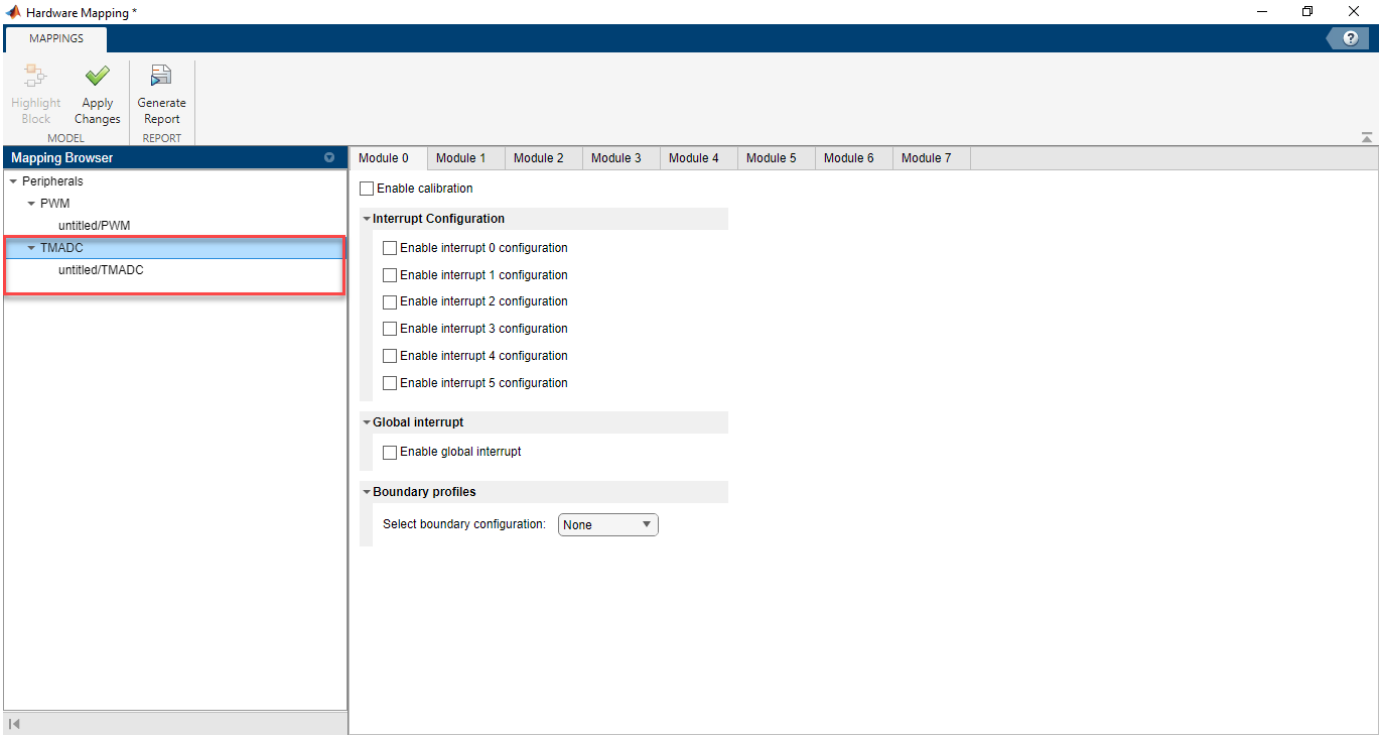
Map TMADC peripherals in the Infineon AURIX model to peripheral registers in the MCU

Description

View and edit the map of peripherals in the Infineon® AURIX™ model to the hardware peripherals.

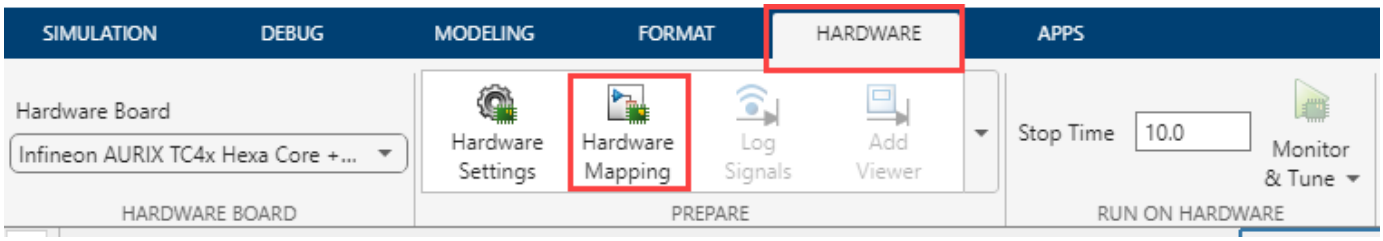
Using the **Peripheral Configuration** tool, you can:

- View and edit configuration parameters for TMADC peripheral block.
- Configure the global parameters. To set the group peripheral, select peripheral in **Browser** > **Peripherals** > ***TMADC***. For more, see
- Check for conflicts, if any between peripherals.



Open the TMADC Peripheral Configuration

- In the **Hardware** tab, click **Hardware Mapping**.



Parameters

Global parameters

`Enable calibration` — TMADC calibration
`off` (default) | `on`

Enables the TMADC calibration for the module.

`Enable interrupt # configuration` — Select interrupt configuration for TMADC
`off` (default) | `on`

Select the interrupt configuration for the TMADC module. This parameter enables the interrupt# for the module.

`Event operation` — Event operation for TMADC
`OR without wait-for-clear` (default) | `AND`

Select the event operation for TMADC module.

Dependencies

To enable the **Event operation** parameter, select either the **Enable interrupt # configuration** parameter or the **Enable global interrupt** parameter.

`Event selection` — Event selection for TMADC
`Result` (default) | `Boundary` | `Error`

Select the event selection for TMADC module.

Dependencies

To enable the **Event operation** parameter, select either the **Enable interrupt # configuration** parameter or the **Enable global interrupt** parameter.

`Interrupt Event source name` — Source name of interrupt event for TMADC channel
`TMADC#_Interrupt_#` | `TMADC_Global_Interrupt_#`

This read-only parameter displays the source name of the interrupt as `TMADCx_Interrupt_y`, where x is the **Module x** and y is the interrupt configuration you enable in the **Module x** tab of Hardware Mapping window.

Dependencies

To view the **Interrupt Event source name**, select either the **Enable interrupt # configuration** parameter or the **Enable global interrupt** parameter.

`Enable global interrupt` — Select global interrupt configuration for TMADC
`off` (default) | `on`

This parameter enables the global interrupt for the TMADC module.

`Select global service request` — Select global interrupt configuration for TMADC
`Global service request 0` (default) | `Global service request 1` | `Global service request 2` | `...`

Select the global service request for the TMADC module.

Dependencies

To enable the **Select global service request** parameter, select the **Enable global interrupt** parameter.

`Global event operation` — Event operation for global interrupt
`OR` (default) | `AND`

Select the event operation for global interrupt.

Dependencies

To enable the **Global event operation** parameter, select the **Enable global interrupt** parameter.

`Select boundary configuration` — Boundary profiles for TMADC
`None` (default) | `Profile-1` | `Profile-2` | `Both profiles`

Select boundary profiles for the TMADC module.

`Upper limit` — TMADC profile upper limit
`2000` (default)

Specify the upper limit for boundary profile.

Dependencies

To set the **Upper limit** parameter for profile #, set the **Select boundary configuration** parameter as either `Profile-#` or `Both profiles`.

`Lower limit` — TMADC profile lower limit
`1000` (default)

Specify the lower limit for boundary profile.

Dependencies

To set the **Lower limit** parameter for profile #, set the **Select boundary configuration** parameter as either `Profile-#` or `Both profiles`.

`Enable boundary flag` — Boundary flag for TMADC module
`off` (default) | `on`

Select this parameter to enable boundary flag for TMADC module.

Dependencies

To enable the **Enable boundary flag** parameter for profile #, set the **Select boundary configuration** parameter as either `Profile-#` or `Both profiles`.

`Boundary result` — Boundary result for TMADC module
`Result 0` (default) | `Result 1` | `Result 2` | `...`

Select the appropriate result register ranging between `Result 0` through `Result 15` for the boundary flag.

Dependencies

To set the **Boundary result** parameter, select the **Enable boundary flag** parameter.

Module

`Module` — TMADC module for conversion
`0` (default) | `1` | `2` | `3` | `...`

Select the TMADC module `0` through `7` on the hardware board.

`Channel(s) selected` — TMADC module selected channels
`1` (default) | `2` | `3` | `4` | `...`

Provides the number of channels selected for the TMADC module. The number of channels selected depends on the input provided on the TMADC block.

Channel group

`Enable channel group` — Select channel group
`Depends on the number of channels selected on the block mask` (default)

This parameter enables the group functionality for the available channels.

Note This parameter is enabled only when number of channels on the block mask is more than one.

`Operating mode for the group` — Operating mode for TMADC channel
`One shot` (default) | `Continuous`

Select the operating mode for TMADC channel group.

Dependencies

To enable the **Operating mode for the group** parameter, select the **Enable channel group** parameter.

`Trigger source for the group` — TMADC trigger source for channel
`Software trigger` (default) | `Hardware trigger #` | `Both hardware triggers`

Select the TMADC trigger source for channel group.

Dependencies

To enable the **Trigger source for the group** parameter, select the **Enable channel group** parameter.

`Source of hardware trigger # for the group` — TMADC hardware trigger source for group
`The default varies based on the selected module` (default) | `GTM-ATOM#` | `GTM-TOM#` | `EGTM-TOM#` | `EGTM-ATOM#`

Select the TMADC hardware trigger source for channel group.

Dependencies

To enable the **Source of hardware trigger # for the group** parameter, select the **Enable channel group** parameter and set **Trigger source for the group** to `Hardware trigger #` or `Both hardware triggers`.

`Hardware trigger # for the group` — TMADC hardware trigger for group
`The default varies based on the selected module` (default) | `ADC_TriggerSignal_#`

Select the TMADC hardware trigger for channel group.

Dependencies

To enable the **Source of hardware trigger # for the group** parameter, select the **Enable channel group** parameter and set **Trigger source for the group** to `Hardware trigger #` or `Both hardware triggers`.

`Hardware trigger # edge for the group` — TMADC hardware trigger edge for group
`Rising-edge` (default) | `Falling-edge` | `Both-edges`

Select the TMADC hardware trigger edge for channel group.

Dependencies

To enable the **Source of hardware trigger # for the group** parameter, select the **Enable channel group** parameter and set **Trigger source for the group** to `Hardware trigger #` or `Both hardware triggers`.

`Trigger delay for the group in nano seconds` — Trigger delay for TMADC channel
`0` (default)

Specify the trigger delay in nano seconds for the TMADC channel group.

Dependencies

To enable the **Trigger delay for the group in nano seconds** parameter, select the **Enable channel group** parameter.

`Select result register` — Result register for TMADC channel group
`Result 0` (default) | `Result 1` | `Result 2` | `...`

Select the result register ranging between `Result 0` through `Result 15` for TMADC channel group.

Dependencies

To enable the **Select result register** parameter, select the **Enable channel group** parameter.

`Enable wait-for-read for the result registers` — Specify if register should wait to read result
`off` (default) | `on`

Select this parameter to enable the register to wait and read the TMADC channel # result registers.

Dependencies

To enable this parameter, select the **Enable channel group** parameter and set the **Trigger source for the group** parameter to `Hardware trigger #` or `Both Hardware triggers`.

Channel

`Select pin` — TMADC pin
`Default value varies based on the module selected` (default) | `AN0`

Select the TMADC pin for conversion.

`Input mode` — Input mode for TMADC pin
`Tri-state` (default) | `Pull-up` | `Push-down`

Select the input mode for the pin(s).

`Speed` — Speed for TMADC pin
`Speed-1` (default) | `Speed-2` | `Speed-3`

Select the pin speed for TMADC.

`Voltage level` — Voltage level for TMADC pin
`Automotive` (default) | `TTL-5V` | `TTL-3.3V`

Select the pin voltage level.

`Select channel #` — Select input for channel
`Input 0` (default) | `Input 1` | `Input 2` | `...`

This parameter is read-only.

This read-only parameter indicates the channel corresponding to the pin selected.

`ADC sampling duration in nano seconds` — TMADC sampling duration
`50` (default)

Specify the sampling duration in nano seconds for the TMADC channel.

`SAR core` — TMADC SAR core for channel
`Core 0` (default) | `Core 1`

Select the Successive-approximation-register (SAR) core for TMADC channel.

`Operating mode` — Operating mode for TMADC channel
`One shot` (default) | `Continuous`

Select the operating mode for TMADC channel.

`Trigger source` — TMADC trigger source for channel
`Software trigger` (default) | `Hardware trigger #` | `Both hardware triggers`

If the **Mode** in block parameters is set to `Trigger and read`, select the TMADC trigger source for channel. If the **Mode** is set to `Trigger only`, then this is a read-only parameter set to `Software trigger`.

`Source of hardware trigger#` — TMADC source of hardware trigger
`The default varies based on the selected module` (default) | `GTM-ATOM#` | `GTM-TOM#` | `EGTM-TOM#` | `EGTM-ATOM#`

If the **Mode** in block parameters is set to `Trigger and read`, select the appropriate trigger source through which conversion initiates.

Dependencies

To enable the **Source of hardware trigger#** parameter, set the **Trigger source** to either `Hardware trigger #` or `Both hardware triggers`.

`Hardware trigger#` — Hardware trigger for TMADC channel
`ADC_TriggerSignal_0` (default) | `ADC_TriggerSignal_1` | `ADC_TriggerSignal_2` | `...`

If the **Mode** in block parameters is set to `Trigger and read`, select the appropriate trigger signal through which conversion initiates.

Dependencies

To enable the **Hardware trigger#** parameter, set the **Trigger source** to either `Hardware trigger #` or `Both hardware triggers`.

`Hardware trigger edge#` — Hardware trigger edge for TMADC channel
`Rising-edge` (default) | `Falling-edge` | `Both-edges`

If the **Mode** in block parameters is set to `Trigger and read`, select the appropriate trigger edge through which conversion initiates.

Dependencies

To enable the **Hardware trigger edge#** parameter, set the **Trigger source** to either `Hardware trigger #` or `Both hardware triggers`.

`Trigger delay in nano seconds` — Trigger delay for TMADC channel
`0` (default)

Specify the trigger delay in nano seconds for the TMADC channel.

Dependencies

To enable the **Trigger delay in nano seconds** parameter, set the **Mode** in block parameters as either `Trigger and read` or `Trigger only`.

`Select result register` — Result register for TMADC channel
`Result 0` (default) | `Result 1` | `Result 2` | `...`

Select the result register for TMADC channel.

Dependencies

To enable the result related parameter **Select result register**, set the **Mode** in block parameters as either `Read results only` or `Trigger and read`.

`Enable wait-for-read` — Specify if register should wait to read result
`off` (default) | `on`

Select this parameter to enable the register to wait and read the TMADC channel # result registers.

Dependencies

To enable this result related parameter, set the **Mode** in block parameters to`Trigger and read` or `Read only`. In the `Trigger and read` mode, set the **Trigger source** to `Hardware trigger #` or `Both Hardware triggers`.

`Result interrupt` — Result interrupt for TMADC channel
`Service request-None` (default) | `Service request-#` | `Global service request`

Select the result interrupt for the TMADC channel.

Dependencies

To enable this parameter, enable the interrupt configuration and set the **Event selection** parameter to `Result` in the global parameters.

`Error interrupt` — Error interrupt for TMADC channel
`Service request-None` (default) | `Service request-#` | `Global service request`

Select error interrupt for the TMADC channel.

Dependencies

To enable this parameter, enable the interrupt configuration and set the **Event selection** parameter to `Error` in the global parameters.

`Select boundary profile` — Boundary profile for TMADC channel result
`None` (default) | `Boundary profile #`

Select the boundary profile for TMADC channel # result.

Dependencies

To enable the **Select boundary profile** parameter, set the **Select boundary configuration** parameter in global parameters to either `Profile-#` or `Both profiles`.

`Boundary selection` — Boundary selection for TMADC channel result
`Disable` (default) | `Lower bound` | `Upper bound` | `Both bound`

Select the boundary type for TMADC channel.

Dependencies

To enable the **Boundary selection** parameter, set the **Select boundary profile** parameter as `Boundary profile-#`.

`Boundary mode` — Boundary mode for TMADC channel result
`Inside boundary` (default) | `Outside boundary`

Select the boundary mode for TMADC channel result.

To enable the **Boundary mode** parameter, set the **Select boundary profile** parameter as `Boundary profile-#`.

`Enable hysteresis` — Boundary hysteresis for TMADC channel result
`Off` (default) | `on`

Enables the hysteresis condition for boundary selection for TMADC channel result.

Dependencies

To enable the **Enable hysteresis** parameter, set the **Boundary selection** to `Both bound`.

`Boundary interrupt` — Boundary interrupt for TMADC channel result
`Service request-None` (default) | `Service request-#` | `Global service request`

Select the boundary interrupt for TMADC channel result.

Dependencies

To enable this parameter, enable the interrupt configuration and set the global parameter **Event selection** to `Boundary`.

Version History

Introduced in R2022b

See Also

TMADC | “Map Tasks and Peripherals Using Hardware Mapping” | Hardware Mapping

Encoder Peripheral Configuration

Map encoder peripherals in the Infineon AURIX model to peripheral registers in the MCU

Description

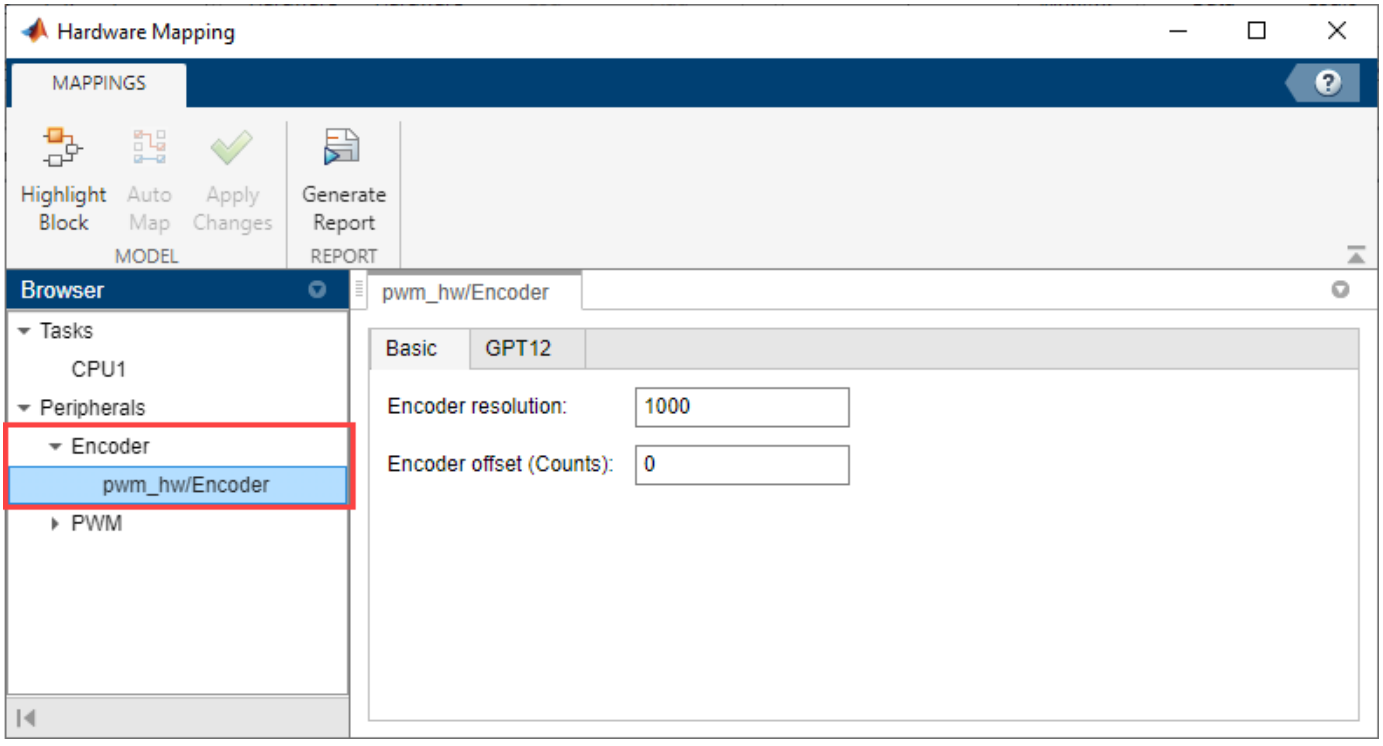
View and edit the map of peripherals in the Infineon AURIX model to the hardware peripherals.

Encoder block is developed based on GPT12 (general purpose timer) module available in the AURIX TC4x devices.

Using the **Peripheral Configuration** tool, you can:

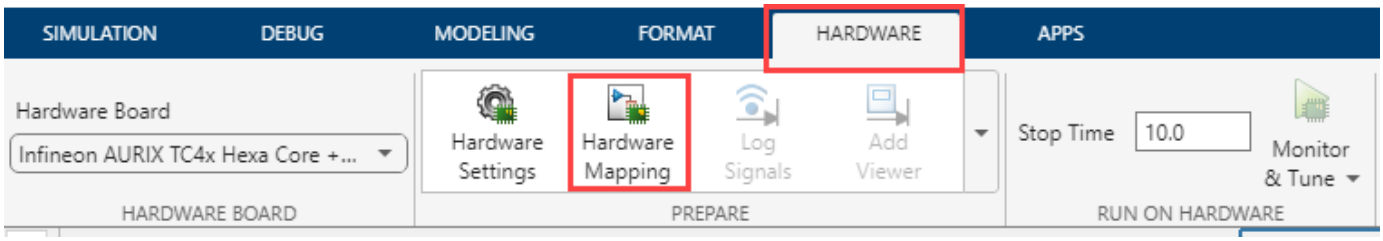
- View and edit the configuration parameters for the Encoder peripheral block.
- Configure the module specific parameters in **Browser** > **Peripherals** > ***Encoder***. For more, see Hardware Mapping
- Check for conflicts, if any between peripherals.

In this workflow, we have used **pwm_hw** as the project name and hence peripheral block name is named as **pwm_hw/Encoder**.



Open the Encoder Peripheral Configuration

- In the Simulink toolstrip, go to **Hardware** tab and click **Hardware Mapping**.



Parameters

Global parameters

`GPT1 prescaler` — Encoder GPT1 prescaler
`8` (default) | `4` | `16` | `32`

Select the general purpose timer (GPT) prescaler for GPT1.

Basic

`Encoder resolution` — Resolution of encoder
`1000` (default) | `positive scalar integer`

Specify the encoder resolution.

`Encoder offset (Counts)` — Encoder offset in counts
`0` (default) | `positive scalar integer`

Specify the encoder offset in counts .

GPT12

`Module` — General purpose timer encoder module
`0` (default) | `1` | `2` | `3`

Select the GPT12 module.

`GPT12 resolution` — GPT12 resolution
`Two-Fold` (default) | `Four-Fold`

Select the GPT12 resolution.

- `Two-Fold` - Considers two edges of encoder sensor signal A.
- `Four-Fold` - Considers both the edges of both encoder sensor signals A and B.

`PinA` — Encoder pin A
`default varies based on the module selected` (default) | `P10_4`

Select the encoder pin A.

`PinB` — Encoder pin B
`default varies based on the module selected` (default) | `P10_7`

Select the encoder pin B.

`Enable zero signal` — Select zero signal for encoder
`on` (default) | `off`

Select this parameter if encoder has Z signal.

`PinZ` — Encoder pin Z
`default varies based on the module selected` (default) | `P10_8`

Select the encoder pin Z.

Dependencies

To enable the **PinZ** parameter, select the **Enable zero signal** parameter.

`Enable slow speed detection` — Select slow speed detection
`off` (default) | `on`

Enable this parameter to measure slower speeds.

Note Enable this parameter, if the application requires to measure the speed of the encoder at a rate faster than the encoder signals.

`GPT2 prescaler` — Encoder GPT2 prescaler
`8` (default) | `4` | `16` | `32`

Select the general purpose timer (GPT) prescaler for GPT2.

Dependencies

To enable the **GPT2 prescaler** parameter, select the **Enable slow speed detection** parameter.

`Speed mode threshold(RPM)` — Speed mode threshold in RPM
`2000` (default)

Slow speed threshold calculation is enabled when the threshold goes below the default value.

Dependencies

To enable the **Speed mode threshold(RPM)** parameter, select the **Enable slow speed detection** parameter.

`Enable interrupt on zero signal` — Enable interrupt on zero signal for encoder
`off` (default) | `on`

Select this parameter to enable the interrupt on zero signal.

Enabling this parameter, will enable the event source name. The event source name is displayed and can be mapped in the tasks(CPU1)

Dependencies

To enable the **Enable interrupt on zero signal** parameter, select the **Enable zero signal** parameter.

`Input mode` — PinA/B/Z input mode
Tri-state (default) | `Pull up` | `Pull down`

Select the input mode for the pinA/B/Z.

Speed — Pin speed
`Speed-1` (default) | `Speed-2` | `Speed-3`

Select the pin speed.

`Voltage level` — Pin voltage level
`Automotive` (default) | `TTL-5V` | `TTL-3.3V`

Select the pin voltage level.

Version History

Introduced in R2022b

See Also

Encoder | “Map Tasks and Peripherals Using Hardware Mapping” | Hardware Mapping

PWM Peripheral Configuration

Map PWM peripherals in theInfineon AURIX model to peripheral registers in the MCU

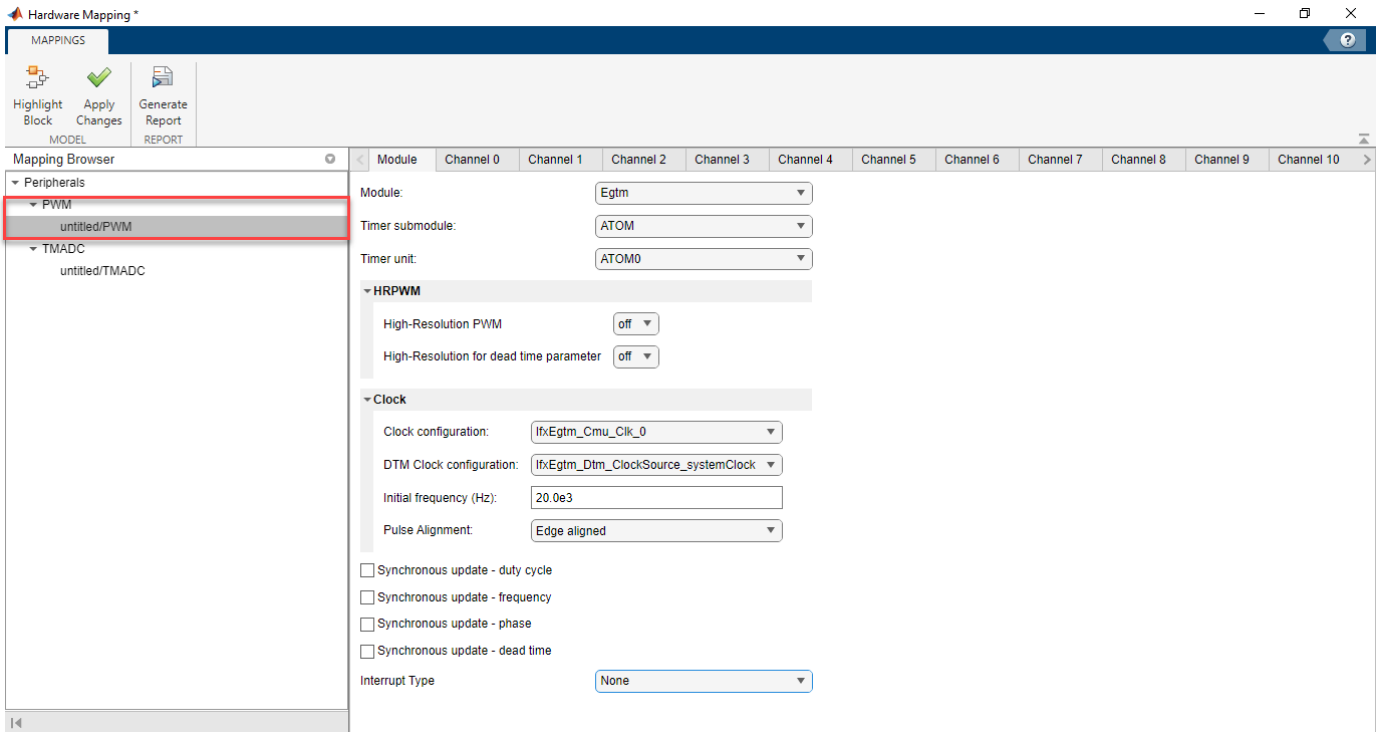
Description

View and edit the module and channel settings of PWM peripherals in the Infineon AURIX model.

Using the **Peripheral Configuration** tool, you can:

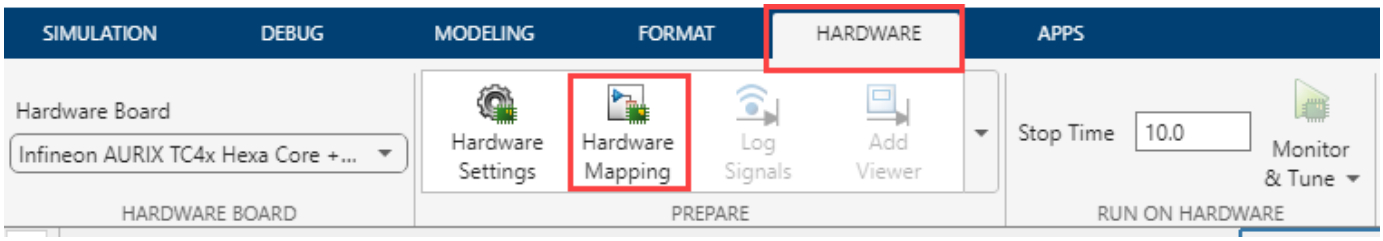
- View and edit the module and channel of PWM peripherals.
- Configure the global parameters. To set the group peripheral, select peripheral in **Browser** > **Peripherals** > ***PWM***.

  The PWM Global parameters are available only if more than two PWM blocks are used in the model. For more, see “Map Tasks and Peripherals Using Hardware Mapping”.
- Check for any conflicts between peripherals.



Open the PWM Peripheral Configuration

- In the Simulink toolstrip, go to **Hardware** tab and click **Hardware Mapping**.



Parameters

Global parameters

`Enable sync group functionality` — Select PWM sync group functionality
`off` (default) | `on`

Select to enable the PWM sync functionality between two PWM blocks.

Note

- Enable sync group functionality parameter is displayed only if more than one PWM block is available in the model.
- Enable this parameter only when both the PWM blocks belong to the same module (either Gtm or Egtm).

When you select this option, the dialog box displays the **Drive Group** and **Trigger Group** options. Two or more PWM blocks in the model enable these global parameters, since these parameters are implied for sync functionality.

`Drive Group` — PWM drive group
`Lists available PWM block` (default)

The drive group lists all the PWM blocks used in the model. Select the PWM block from the list.

Dependencies

To enable the **Drive Group** parameter, select the **Enable sync group functionality** parameter.

`Trigger Group` — PWM trigger group
`Lists available PWM block` (default)

The trigger group lists all the PWM blocks used in the model. Select the PWM block from the list.

The PWM generated from the trigger block syncs with the PWM generated from the drive block.

Dependencies

To enable the **Trigger Group** parameter, select the **Enable sync group functionality** parameter.

Module

`Module` — Select the PWM module
`Gtm` (default) | `Egtm`

Select the generic timer module (Gtm) or enhanced generic timer (Egtm) module.

`Timer submodule` — Select timer submodule for PWM
`TOM` (default) | `ATOM`

Select the Timer Output Module (TOM) or ARU-connected Timer Output Module (ATOM) timer submodule. .

`Timer Unit` — Select timer unit for PWM
`TOM0` (default) | `TOM1` | `...` | `ATOM0` | `ATOM1` | `...`

Select the timer unit for PWM.

For example, `TOM0`, 0 represents the cluster number.

Dependencies

The Timer Unit options depends on the selection of Timer submodule. If TOM is set as Timer submodule, then the Timer unit list TOM# options. If ATOM is set as Timer submodule, then the Timer unit list ATOM# options.

`High-Resolution PWM` — High resolution PWM
`off` (default) | `on`

Select `On` to enable the high resolution PWM.

Dependencies

High-Resolution PWM parameter is disabled and Read-only. You can enable this parameter when the **Module** parameter is set to `EGTM` and **Timer submodule** is set to `ATOM`.

`High-Resolution for dead time parameter` — High resolution for dead time parameter
`off` (default) | `on`

Select this parameter to enable high resolution for the dead-time parameter.

For more information about configuring a dead-time module, see .

Dependencies

To enable this parameter set **Module** to `EGTM` and **Timer submodule** to `ATOM`.

`Clock configuration` — Select PWM clock configuration
`IfxGtm_Cmu_Fxclk_0` (default) | `IfxGtm_Cmu_Fxclk_1` | `IfxGtm_Cmu_Fxclk_2` | **…**

Select the PWM clock configuration. The PWM clock configuration parameters lists all the fixed clock and CMU clock options.

For example, if you select Module as `Gtm` and Timer submodule as `TOM`, the clock configuration divider is represented as `IfxGtm_Cmu_Fxclk`. When you select Timer submodule as `ATOM`, the clock configuration divider is represented as `IfxGtm_Cmu_clk`. Similarly, for Module Egtm, the clock configuration divider is either `IfxEgtm_Cmu_Fxclk` or `IfxEgtm_Cmu_clk`.

For TOM modules there are five different divide values based on the selection and for ATOM there are eight different selections of clock configuration.

- `TOM` - FXCLK Divider 0 (2^0), FXCLK Divider 1 (2^4), FXCLK Divider 2 (2^8), FXCLK Divider 3 (2^{12}), and FXCLK Divider 4 (2^{16})
- `ATOM` - CMUCLK Divider 0 (2^0), CMUCLK Divider 1 (2^1), CMUCLK Divider 2 (2^2), CMUCLK Divider 3 (2^3), CMUCLK Divider 4 (2^4), CMUCLK Divider 5 (2^5), CMUCLK Divider 6 (2^6), and CMUCLK Divider 7 (2^7).

Dependencies

Clock configuration depends on the selection made in **Module** and **Timer submodule**.

- If TOM submodule is selected, then fixed clocks are displayed.
- If ATOM is selected, then configurable CMU clocks are displayed.

`Initial frequency (Hz)` — Initial frequency (Hz) for PWM
`20000` (default) | `Float`

Specify the initial frequency (Hz) for PWM.

`Pulse alignment` — Select pulse alignment for PWM
`Edged aligned` (default) | `Center aligned`

Select the pulse alignment for PWM.

- Select edge aligned if you want to generate edge aligned PWM pulse.
- Select center aligned if you want to generate symmetric center aligned PWM pulse.

`Synchronous update - duty cycle` — Enable synchronous update duty cycle for PWM
`On` (default) | `off`

Enable this parameter to synchronously update duty for all sync channels.

`Synchronous update - frequency` — Enable synchronous update frequency for PWM
`On` (default) | `off`

Enable this parameter to synchronously update frequency for all sync channels.

Dependencies

To enable this parameter, select **Enable frequency input** parameter in the PWM block.

`Synchronous update - phase` — Enable synchronous update phase for PWM
`On` (default) | `off`

Enable this parameter to synchronously update phase for all sync channels.

Dependencies

To enable this parameter, select **Enable phase input** parameter in the PWM block.

`Synchronous update - dead time` — Enable synchronous update dead time for PWM
`On` (default) | `off`

Enable this parameter to synchronously update dead time for all sync channels. This parameter is read-only.

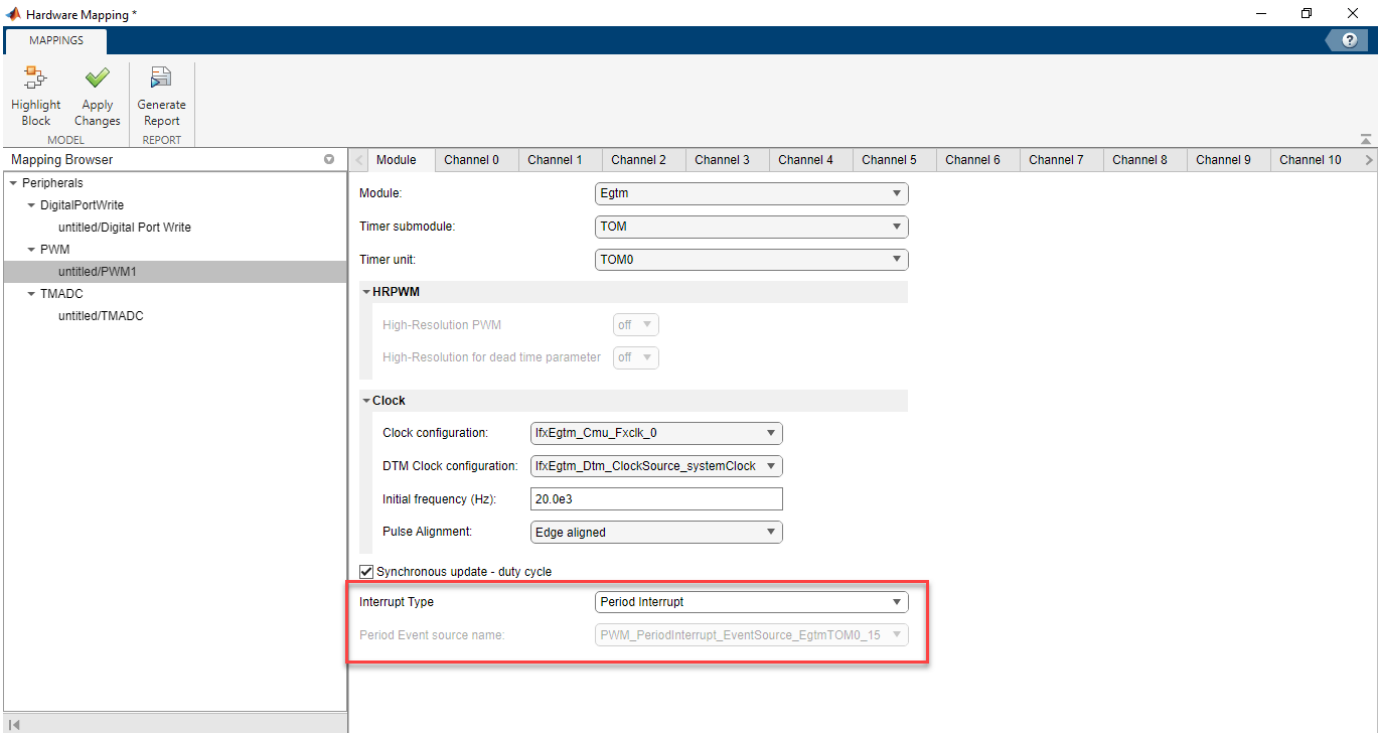
Dependencies

To enable this parameter, select **Enable dead time properties input** parameter in the PWM block.

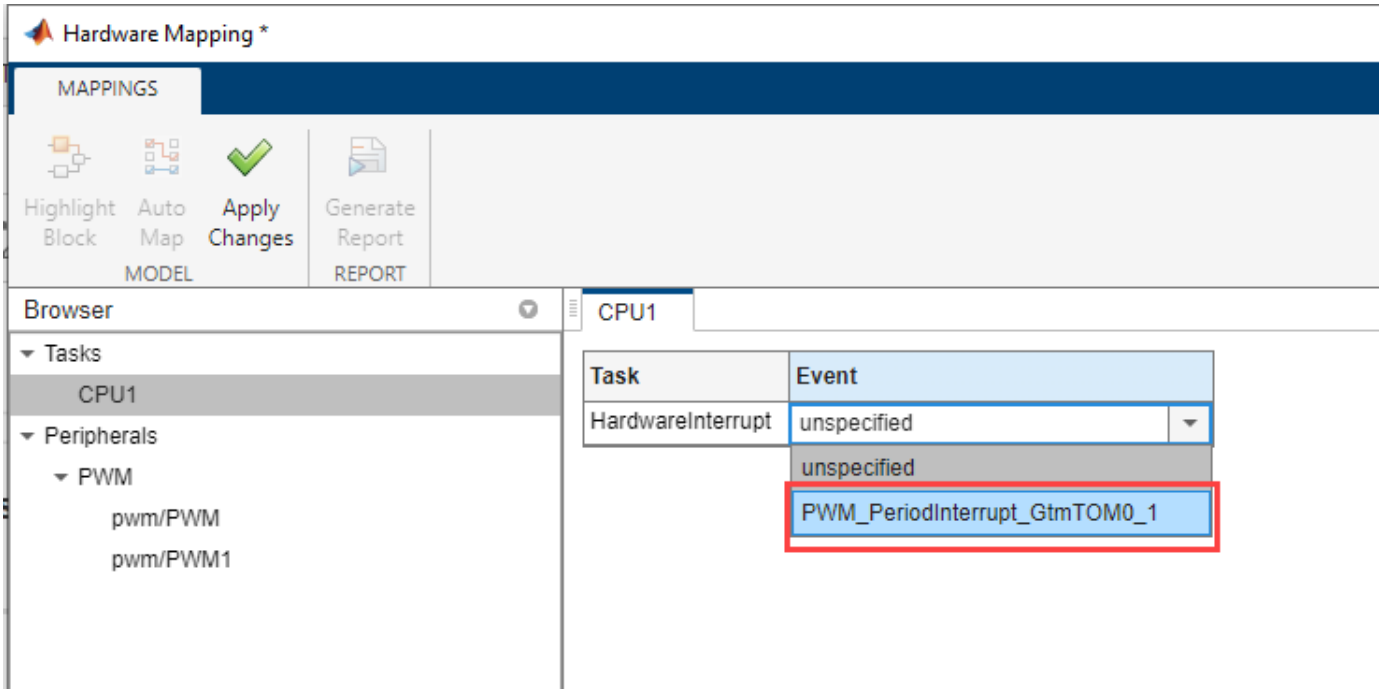
`Interrupt Type` — Interrupt event for PWM
`none` (default) | `Period Interrupt` | `Duty Interrupt`

Specify the type of interrupt event for the PWM as one of these options.

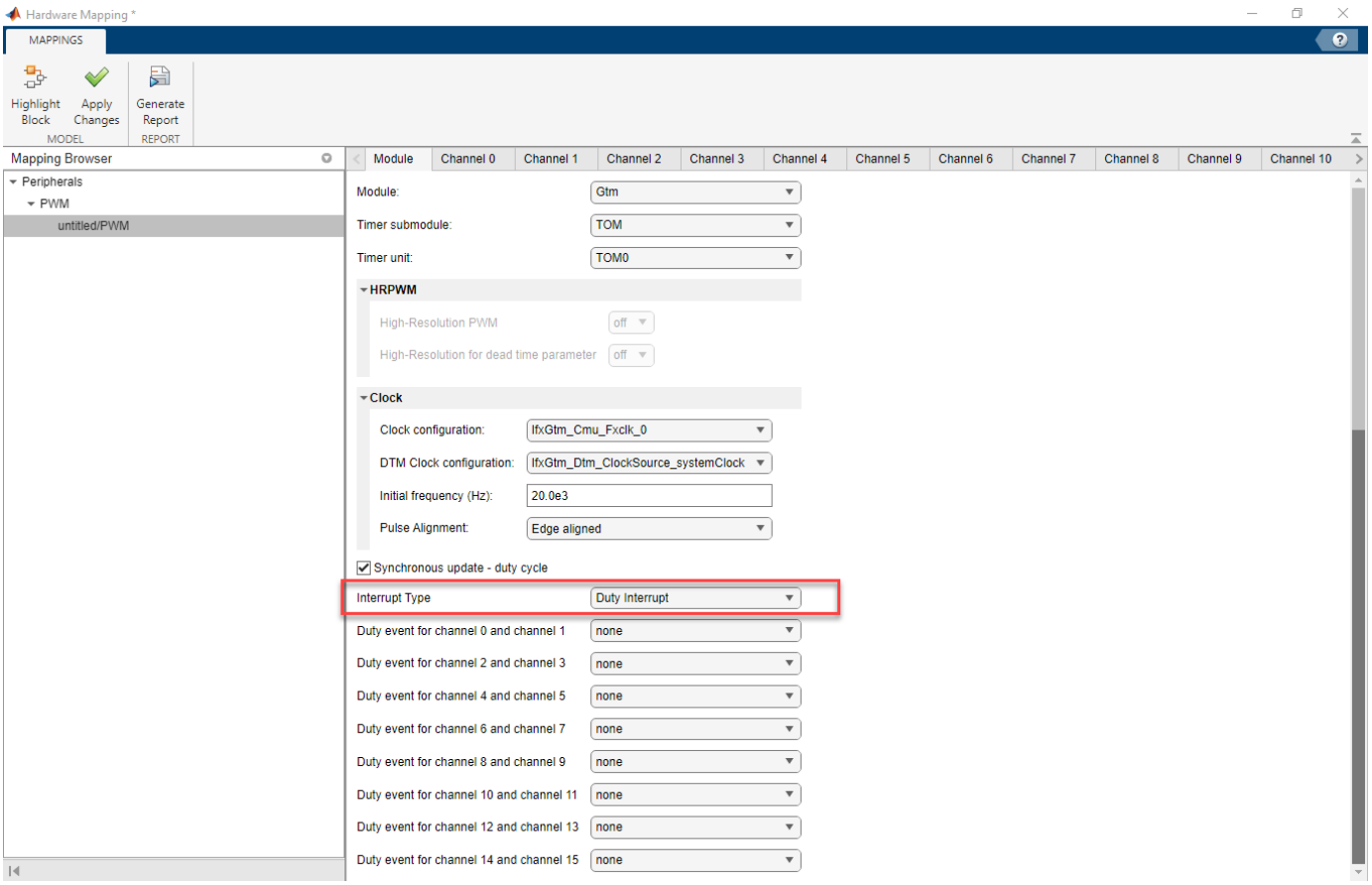
- `None`—Select this option when you do not want to specify an interrupt event.
- `Period Interrupt`—Select this option to specify an interrupt event at the end of every period.



Selecting this option also enables the read-only parameter **Period Event source name**, which provides information on the source of the period interrupt. The source depends on**Module**, **Timer submodule**, and **Timer unit** parameters. You can map the name in the tasks (CPU1).



- `Duty Interrupt`—Select this option to enable an interrupt at the start of every duty cycle.



`Duty event for channel # and channel #` — Duty event for channel pairs
`none` (default) | `Options vary based on the module selected and the channel pair`

Specify the duty event based on the module and channel pair.

Channel pairs such as channel 0 and channel 1, channel 2 and channel 3, ... , channel 14 and channel 15, share same duty event sources. For example, if you select the `Gtm`**Module** and the channel 0 and channel 1 pair, the available duty options are:

- `IfxGtm_Pwm_SubModule_Ch_0`
- `IfxGtm_Pwm_SubModule_Ch_1`

If you select the `IfxGtm_Pwm_SubModule_Ch_0` event, the block enables the read-only **Duty Event source name** parameter in **Channel 0** tab.

Dependencies

To enable this parameter, set **Interrupt Type** to `Duty Interrupt`.

Channel

`Channel #` — Select PWM channel
`Default varies based on the module selected` (default) | `IfxGtm_Pwm_SubModule_Ch_1` | `IfxGtm_Pwm_SubModule_Ch_2` | …

The number of channel(s) selected in the PWM block will appear in their respective channel tabs of the PWM peripheral. Each of these channels must be configured respectively.

Dependencies

The channel # list depends on

- Module selected in the module tab (Gtm or Egtm)
- Timer Submodule (TOM or ATOM)
- Timer unit (TOM# or ATOM#)

`Output pin #` — Select PWM output pin channel
`P00_9` (default) | `P01_9` | `P02_0` | `...`

Select the PWM output pin channel.

Dependencies

The Output pin # list depends on

- Module selected in the module tab (Gtm or Egtm)
- Timer Submodule (TOM or ATOM)
- Timer unit (TOM# or ATOM#)
- Channel

`Output mode` — PWM output mode
`Push pull` (default) | `Open drain`

Select the output mode for the PWM pin(s).

Speed — Pin speed
`Speed-1` (default) | `Speed-2` | `Speed-3`

Select the pin speed.

Voltage level — Pin voltage level
`Automotive` (default) | `TTL-5V` | `TTL-3.3V`

Select the pin voltage level.

Signal polarity — Signal polarity of PWM
`Active High` (default) | `Active Low`

Select the signal polarity of PWM channel.

Initial Duty cycle (%) — PWM initial duty cycle in percentage
`50` (default)

Specify the PWM initial duty cycle in percentage. Range varies from 0-100%.

Initial offset (rad) — PWM initial offset in radians
`0` (default)

Specify the initial offset in radians.

QSPI Trigger Signal — QSPI trigger from PWM channel
`None` (default) | `Options vary based on choice of source of hardware trigger`

Select the signal to trigger the QSPI module from the PWM channel.

To trigger QSPI module, set the **Trigger source** parameter to `Hardware trigger` and **Source of hardware trigger** to `GTM` or `eGTM` in the corresponding **Module** tab of QSPI Hardware Mapping window.

For more information on QSPI trigger signals corresponding to the `GTM` module, see .

For more information on QSPI trigger signals corresponding to the `eGTM` module, see .

The **QSPI Trigger Signal** list depends on:

- Source of hardware trigger in the QSPI module tab (`GTM` or `eGTM`)
- Timer Submodule in the PWM module tab (`TOM` or `ATOM`)
- Timer unit in the PWM module tab
- Channel

Note You must have a QSPI block in the model to trigger the QSPI module. You can trigger multiple QSPI modules by configuring them in different QSPI blocks.

You can trigger multiple QSPI modules by configuring them in different QSPI blocks.

Enable complementary output — Enable PWM complementary output
`off` (default) | `on`

Enables the complementary output for PWM.

`Complementary output pin` — Select PWM complementary output pin
`Default varies based on the module, timer unit and channel selected` (default) | `P00_8` | `P01_13` | `...`

Select the PWM complementary output pin.

Dependencies

To enable the **Complementary output pin** parameter, select the **Enable complementary output** parameter.

`Signal polarity of complementary pin` — Select signal polarity of complementary pin
`Active Low` (default) | `Active High`

Select the signal polarity of complementary pin.

Dependencies

To enable the **Signal polarity of complementary pin** parameter, select the **Enable complementary output** parameter.

`Enable Dead time insertion` — Flag to specify dead time
`off` (default) | `on`

Enables the dead time insertion.

For more information on dead time module, see .

Dependencies

To enable the **Enable Dead time insertion** parameter, select the **Enable complementary output** parameter.

`Dead-time for rising edge(s)` — PWM dead time for rising edge(s)
`0` (default) | `positive scalar integer`

Specify the PWM dead time for rising edge(s) in seconds.

For more information on dead time module, see .

Dependencies

To enable the **Dead-time for rising edge(s)** parameter, select the **Enable Dead time insertion** parameter.

`Dead-time for falling edge(s)` — PWM dead time for falling edge(s)
`0` (default) | `positive scalar integer`

Specify the PWM dead time for falling edge(s) in seconds.

For more information on dead time module, see .

Dependencies

To enable the **Dead-time for falling edge(s)** parameter, select the **Enable Dead time insertion** parameter.

`ADC trigger` — Enable ADC trigger for PWM channel
`off` (default) | `on`

Enable this parameter to trigger ADC conversion at the end of every period.

Note You must have an ADC block in the model to trigger ADC conversion.

`Number of trigger channels` — Select ADC trigger channels
`1` (default) | `2` | `3` | `4`

Select the number of ADC trigger channels.

Dependencies

To enable the **Number of trigger channels** parameter, select the **ADC trigger** parameter.

`Trigger channel#` — Select ADC trigger signal
`ADC_TriggerSignal_0` (default) | `ADC_TriggerSignal_1` | `ADC_TriggerSignal_2` | `...`

Select the ADC trigger signal channel.

Dependencies

- To enable the **Trigger channel#** parameter, select the **ADC trigger** parameter.
- Number of trigger channels list depends on the timer module selected.

More About

QSPI Trigger Signals Corresponding to GTM Module

If you set the **Source of hardware trigger** parameter in QSPI **Module** tab to `GTM`, then the signal you use to trigger the QSPI module depends on the timer unit and PWM channel you select in the **Timer unit** and **channel** parameters, respectively. For example, the combination (A#, C#) represents (`ATOM #`, `IfxGtm_Pwm_SubModule_Ch_#`) and (T#, C#) represents (`TOM #`, `IfxGtm_Pwm_SubModule_Ch_#`).

- To set the **QSPI Trigger Signal** parameter to `IfxGtm_cfg_QspiTriggerSignal_0`, you must have one of these timer unit and channel combinations: (A0, C7), (A1, C7), (A2, C7), (A0, C0), (A1, C0), (A2, C0), (A3, C7), (A4, C7), (A5, C7), (A5, C7), (A6, C7), (A7, C7), (A8, C7), (A9, C7), (A10, C7), (A11, C7), (A3, C0), (A4, C0), (A5, C0), (A6, C0), (T0, C5), (T1, C5), (T0, C8), (T1, C8), (T2, C15), (T3, C15), (T4, C15), (T5, C15), (T2, C 8), (T3, C8), (T4, C8), (T5, C8).
- To set the **QSPI Trigger Signal** parameter to `IfxGtm_cfg_QspiTriggerSignal_1`, you must have one of these timer unit and channel combinations: (A0, C16), (A1, C16), (A2, C16), (A0, C1), (A1, C1), (A2, C1), (A3, C6), (A4, C6), (A5, C6), (A6, C6), (A7, C6), (A8, C6), (A9, C6), (A10, C6), (A11, C6), (A3, C1), (A4, C1), (A5, C1), (A6, C1), (T0, C14), (T1, C14), (T0, C9), (T1, C9), (T2, C14), (T3, C14), (T4, C14), (T5, C14), (T2, C 9), (T3, C9), (T4, C9), (T5, C9).
- To set the **QSPI Trigger Signal** parameter to `IfxGtm_cfg_QspiTriggerSignal_2`, you must have one of these timer unit and channel combinations: (A0, C5), (A1, C5), (A2, C5), (A0, C2), (A1, C2), (A2, C2), (A3, C5), (A4, C5), (A5, C5), (A6, C5), (A7, C5), (A8, C5), (A9, C5), (A10, C5), (A11, C5), (A3, C2), (A4, C2), (A5, C2), (A6, C2), (T0, C13), (T1, C13), (T0, C10), (T1, C10), (T2, C13), (T3, C13), (T4, C13), (T5, C13), (T2, C10), (T3, C10), (T4, C10), (T5, C10)

- To set the **QSPI Trigger Signal** parameter to `IfxGtm_cfg_QspiTriggerSignal_3`, you must have one of these timer unit and channel combinations: (A0, C4), (A1, C4), (A2, C4), (A0, C3), (A1, C3), (A2, C3), (A3, C4), (A4, C4), (A5, C4), (A6, C4), (A7, C4), (A8, C4), (A9, C4), (A10, C4), (A11, C4), (A3, C3), (A4, C3), (A5, C3), (A6, C3), (T0, C12), (T1, C12), (T0, C11), (T1, C11), (T2, C12), (T3, C12), (T4, C12), (T5, C12), (T2, C11), (T3, C11), (T4, C11), (T5, C11).
- To set the **QSPI Trigger Signal** parameter to `IfxGtm_cfg_QspiTriggerSignal_4`, you must have one of these timer unit and channel combinations: (A0, C3), (A1, C3), (A2, C3), (A0, C4), (A1, C4), (A2, C4), (A3, C3), (A4, C3), (A5, C3), (A6, C3), (A7, C3), (A8, C3), (A9, C3), (A10, C3), (A11, C3), (A3, C4), (A4, C4), (A5, C4), (A6, C4), (T0, C11), (T1, C11), (T0, C11), (T1, C11), (T2, C11), (T3, C11), (T4, C11), (T5, C11), (T2, C12), (T3, C12), (T4, C12), (T5, C12).
- To set the **QSPI Trigger Signal** parameter to `IfxGtm_cfg_QspiTriggerSignal_5`, you must have one of these timer unit and channel combinations: (A0, C2), (A1, C2), (A2, C2), (A0, C5), (A1, C5), (A2, C5), (A3, C2), (A4, C2), (A5, C2), (A6, C2), (A7, C2), (A8, C2), (A9, C2), (A10, C2), (A11, C2), (A3, C5), (A4, C5), (A5, C5), (A6, C5), (T0, C10), (T1, C10), (T0, C13), (T1, C13), (T2, C10), (T3, C10), (T4, C10), (T5, C10), (T2, C13), (T3, C13), (T4, C13), (T5, C13).
- To set the **QSPI Trigger Signal** parameter to `IfxGtm_cfg_QspiTriggerSignal_6`, you must have one of these timer unit and channel combinations: (A0, C1), (A1, C1), (A2, C1), (A0, C6), (A1, C6), (A2, C6), (A3, C1), (A4, C1), (A5, C1), (A6, C1), (A7, C1), (A8, C1), (A9, C1), (A10, C1), (A11, C1), (A3, C6), (A4, C6), (A5, C6), (A6, C5), (T0, C9), (T1, C9), (T0, C14), (T1, C14), (T2, C9), (T3, C9), (T4, C9), (T5, C9), (T2, C14), (T3, C14), (T4, C14), (T5, C14).
- To set the **QSPI Trigger Signal** parameter to `IfxGtm_cfg_QspiTriggerSignal_7`, you must have one of these timer unit and channel combinations: (A0, C0), (A1, C0), (A2, C0), (A0, C7), (A1, C7), (A2, C7), (A3, C0), (A4, C0), (A5, C0), (A6, C0), (A7, C0), (A8, C0), (A9, C0), (A10, C0), (A11, C0), (A3, C7), (A4, C7), (A5, C7), (A6, C7), (T0, C8), (T1, C8), (T0, C15), (T1, C15), (T2, C8), (T3, C8), (T4, C8), (T5, C8), (T2, C15), (T3, C15), (T4, C15), (T5, C15).

Note The **QSPI Trigger Signal** parameter remains disabled in the block dialog box if you do not select one of these timer unit and channel combinations.

QSPI Trigger Signals Corresponding to eGTM

If you set the **Source of hardware trigger** in QSPI **Module** tab to `eGTM`, then the signal you use to trigger the QSPI module depends on the timer unit and PWM channel you select in the **Timer unit** and **channel** parameters, respectively. For example, the combination (A#, C#) represents (`ATOM #`, `IfxEgtm_Pwm_SubModule_Ch_#`) and (T#, C#) represents (`TOM #`, `IfxEgtm_Pwm_SubModule_Ch_#`).

- To set the **QSPI Trigger Signal** parameter to `IfxEgtm_cfg_QspiTriggerSignal_0`, you must have one of these timer unit and channel combinations: (A0, C7), (A1, C7),(A2, C7), (A0, C0), (A1, C0), (A2, C0), (T0, C15), (T1, C15), (T0, C8), (T1, C8).
- To set the **QSPI Trigger Signal** parameter to `IfxEgtm_cfg_QspiTriggerSignal_1`, you must have one of these timer unit and channelcombinations: (A0, C6), (A1, C6),(A2, C6), (A0, C1), (A1, C1), (A2, C1), (T0, C14), (T1, C14), (T0, C9), (T1, C9).
- To set the **QSPI Trigger Signal** parameter to `IfxEgtm_cfg_QspiTriggerSignal_2`, you must have one of these timer unit and channel combinations: (A0, C5), (A1, C5),(A2, C5), (A0, C2), (A1, C2), (A2, C2), (T0, C13), (T1, C13), (T0, C10), (T1, C10).
- To set the **QSPI Trigger Signal** parameter to `IfxEgtm_cfg_QspiTriggerSignal_3`, you must have one of these timer unit and channel combinations: (A0, C4), (A1, C4),(A2, C4), (A0, C3), (A1, C3), (A2, C3), (T0, C12), (T1, C12), (T0, C11), (T1, C11).

- To set the **QSPI Trigger Signal** parameter to `IfxEgtm_cfg_QspiTriggerSignal_4`, you must have one of these timer unit and channel combinations: (A0, C3), (A1, C3),(A2, C3), (A0, C4), (A1, C4), (A2, C4), (T0, C11), (T1, C11), (T0, C12), (T1, C12).
- To set the **QSPI Trigger Signal** parameter to `IfxEgtm_cfg_QspiTriggerSignal_5`, you must have one of these timer unit and channel combinations: (A0, C2), (A1, C2),(A2, C2), (A0, C5), (A1, C5), (A2, C5), (T0, C10), (T1, C10), (T0, C13), (T1, C13).
- To set the **QSPI Trigger Signal** parameter to `IfxEgtm_cfg_QspiTriggerSignal_6`, you must have one of these timer unit and channel combinations: (A0, C1), (A1, C1),(A2, C1), (A0, C6), (A1, C6), (A2, C6), (T0, C9), (T1, C9), (T0, C14), (T1, C14).
- To set the **QSPI Trigger Signal** parameter to `IfxEgtm_cfg_QspiTriggerSignal_7`, you must have one of these timer unit and channel combinations: (A0, C0), (A1, C0),(A2, C0), (A0, C7), (A1, C7), (A2, C7), (T0, C8), (T1, C8), (T0, C15), (T1, C15).

Note The **QSPI Trigger Signal** parameter remains disabled in the block dialog box if you do not select one of these timer unit and channel combinations.

Dead-Time Module Properties

Follow these steps to use the dead-time functionality:

- Select the **Enable dead time properties input** parameter in the block to configure the `dtm_rise` and `dtm_fall` input ports.
- Select the **Enable complementary output** parameter Hardware Mapping window corresponding to that channel you have selected to configure the **Enable dead time insertion**, **Dead-time for rising edge(s)**, and **Dead-time for falling edge(s)** parameters.
- To enable the High-Resolution for dead time parameter, in the **Module** tab of the Hardware Mapping window, set the **Module** parameter to `EGTM` and the **Timer submodule** parameter to `ATOM`.

Version History

Introduced in R2022b

See Also

PWM | “Map Tasks and Peripherals Using Hardware Mapping” | Hardware Mapping

QSPI Peripheral Configuration

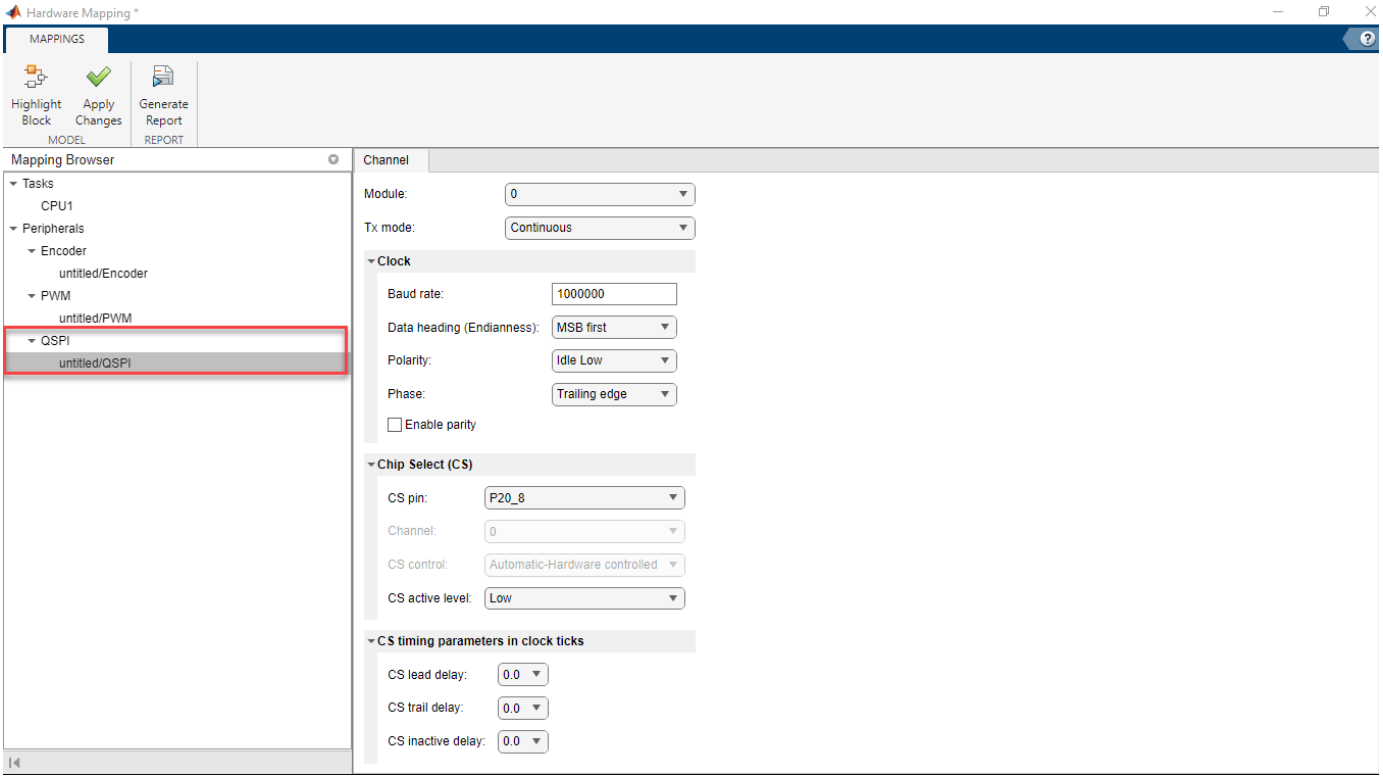
Map QSPI peripherals in the Infineon AURIX model to peripheral registers in the MCU

Description

View and edit the map of peripherals in the Infineon AURIX model to the hardware peripherals.

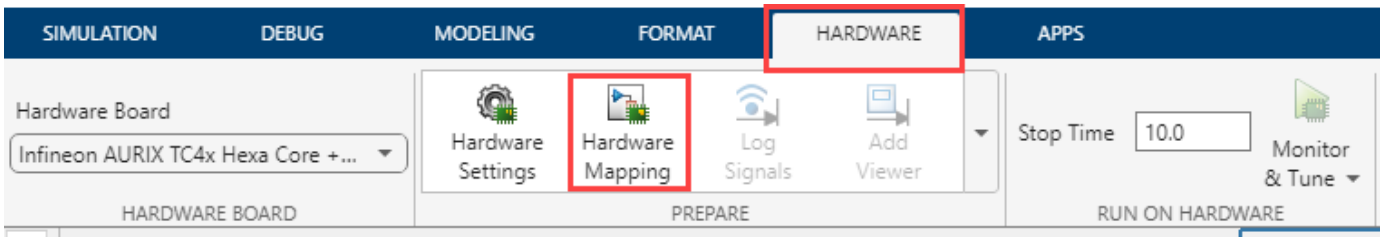
Using the **Peripheral Configuration** tool, you can:

- View and edit configuration parameters for QSPI peripheral block.
- Configure the global parameters. To set the group peripheral, select peripheral in **Browser** > **Peripherals** > ***QSPI***. For more, see “Map Tasks and Peripherals Using Hardware Mapping”
- Check for conflicts, if any between peripherals.



Open the QSPI Peripheral Configuration

- In the Simulink toolstrip, go to **Hardware** tab and click **Hardware Mapping**.



Parameters

Global parameters

`Enable Tx FIFO event` — QSPI transmit FIFO event
`off` (default) | `on`

Enables the QSPI transmit FIFO event.

When you select this option, the dialog box displays the **Tx FIFO Mode Selection** options.

Note

- Enabling this parameter, expect that the data is handled through interrupts. Therefore it is mandatory to use QSPI block (Transfer mode as SPI Transmit) during the events.
- Global parameters for enable Tx FIFO event and Rx FIFO event are not applicable if the QSPI block with **Transfer mode** is set to `SPI transmit and receive`.

`Enable Rx FIFO event` — QSPI receive FIFO event
`off` (default) | `on`

Enables the QSPI receive FIFO empty event.

When you select this option, the dialog box displays the **Rx FIFO Mode Selection** options.

Note

- Enabling this parameter, expect that the data is handled through interrupts. Therefore it is mandatory to use QSPI block (Transfer mode as SPI Receive) during the events.
- Global parameters for enable Tx FIFO event and Rx FIFO event are not applicable if the QSPI block with **Transfer mode** is set to `SPI transmit and receive`.

`Enable error event` — QSPI error event
`off` (default) | `on`

Enables the QSPI error event.

Note Enabling this parameter, expects that the data is handled through interrupts. Therefore it is recommended to use QSPI block (SPI receive or transmit as transfer mode) during the events.

`Tx FIFO mode` — QSPI transmit FIFO mode selection
`Batch mode-1` (default)

This parameter is read-only.

This read-only parameter indicates the Tx FIFO events occur based on Batch mode-1.

Dependencies

To enable this parameter, select the **Enable Tx FIFO event** parameter.

`Tx FIFO threshold` — QSPI transmit FIFO threshold
`0` (default) | `3` | `4` | `...`

Enables the QSPI transmit FIFO threshold limit.

Dependencies

To enable this parameter, select the **Enable Tx FIFO event** parameter.

`Rx FIFO mode` — QSPI receive FIFO mode selection
`Batch mode-1` (default)

This parameter is read-only.

This read-only parameter indicates the Rx FIFO events occur based on Batch move-1.

Dependencies

To enable this parameter, select the **Enable Rx FIFO event** parameter.

`Trigger source` — QSPI trigger source
`Software trigger` (default) | `Hardware trigger`

Select the type of trigger source for the QSPI.

`Source of hardware trigger` — Source of hardware trigger selection
`GTM` (default) | `eGTM`

Select the source of hardware trigger for the QSPI.

Dependencies

To enable this parameter, set the **Trigger source** parameter to `Hardware trigger`.

`Rx FIFO threshold` — QSPI receive FIFO threshold
`0` (default) | `3` | `4` | `...`

Enables the QSPI receive FIFO threshold limit.

Dependencies

To enable this parameter, select the **Enable Rx FIFO event** parameter.

`Serial Clock pin (SCK)` — QSPI serial clock pin selection
`The default varies based on the module selected` (default) | `P22_7` | `...`

Select the QSPI serial clock pin selection. The list varies based on the module selected.

`Serial Data Out pin (SDO)` — QSPI serial data out pin selection
`The default varies based on the module selected` (default) | `P20_14` | `P22_10` | `...`

Select the QSPI serial data out pin selection. The list varies based on the module selected.

`Serial Data Input pin (SDI)` — QSPI serial data input pin selection
`The default varies based on the module selected` (default) | `P22_6` | `P22_9` | `...`

Select the QSPI serial data input pin selection. The list varies based on the module selected.

Channel

`Module` — QSPI Module
`0` (default) | `1` | `2` | `...`

Select the QSPI module `0` through `7` on the hardware board.

`Tx mode` — QSPI transmit mode
`Continuous` (default) | `Single transfer`

Select the QSPI transmit mode.

- `Continuous` - This option activates the chip select signal till the data transfer completes.
- `Single-transfer` - This option deactivates the chip select signal for every data element involved in the data transfer.

`Baud rate` — QSPI clock baud rate
`1000000` (default) | `positive scalar integer`

Specifies the rate of data communication between the peripherals connected (clock period).

`Data heading (Endianness)` — QSPI data heading in binary
`MSB first` (default) | `LSB first`

Select the QSPI data heading in binary numbers.

- `MSB first` - the bit furthest to the left (msb) is moved first from SDO pin followed by the subsequent left bits.
- `LSB first` - the bit furthest to the right (lsb) is moved first from SDO pin followed by the subsequent right bits.

`Polarity` — QSPI clock polarity
`Idle low` (default) | `Idle High`

Select the QSPI clock polarity in idle state.

`Phase` — QSPI clock phase
`Trailing edge` (default) | `Leading edge`

Select the QSPI clock phase.

`Enable parity` — Enable QSPI clock parity
`off` (default) | `on`

Enables the QSPI clock parity.

When you select the **Enable parity** option, the dialog box displays the **Parity** option.

`Parity` — QSPI clock parity
`Odd parity` (default) | `Even parity`

Select the QSPI clock parity.

Dependencies

To enable **Parity** parameter, select the **Enable parity** parameter.

`CS pin` — QSPI chip select pin
`The default varies based on the module selected` (default)

Select the QSPI chip select pin.

`Channel` — QSPI channel
`The default varies based on the module selected` (default) | `1` | | `...`

This parameter is read-only.

Select the QSPI channel `0` through `13`. This read-only parameter indicates the channel corresponding to the CS pin selected.

`CS control` — QSPI chip select control
`Automatic-Hardware controlled` (default) | `Software controlled`

This parameter is read-only.

This parameter is read-only. Select the QSPI chip select control.

`CS active level` — QSPI chip select active level
`Low` (default) | `High`

Select the QSPI chip select active level.

`CS lead delay` — QSPI CS lead delay
`0.0` (default)

Introduces the selected delay between the active edge of the CS pin and the first shift clock edge.

`CS trail delay` — QSPI CS trail delay
`0.0` (default)

Introduces the selected delay between shift clock period of a data block and is followed either by the deactivating edge of CS pin, or a new data block in continuous mode.

`CS inactive delay` — QSPI CS inactive delay
`0.0` (default)

Introduces the selected delay between the end of the last TRAIL phase of a frame.

Version History

Introduced in R2022b

See Also

QSPI | “Map Tasks and Peripherals Using Hardware Mapping” | Hardware Mapping

Digital Port Read Peripheral Configuration

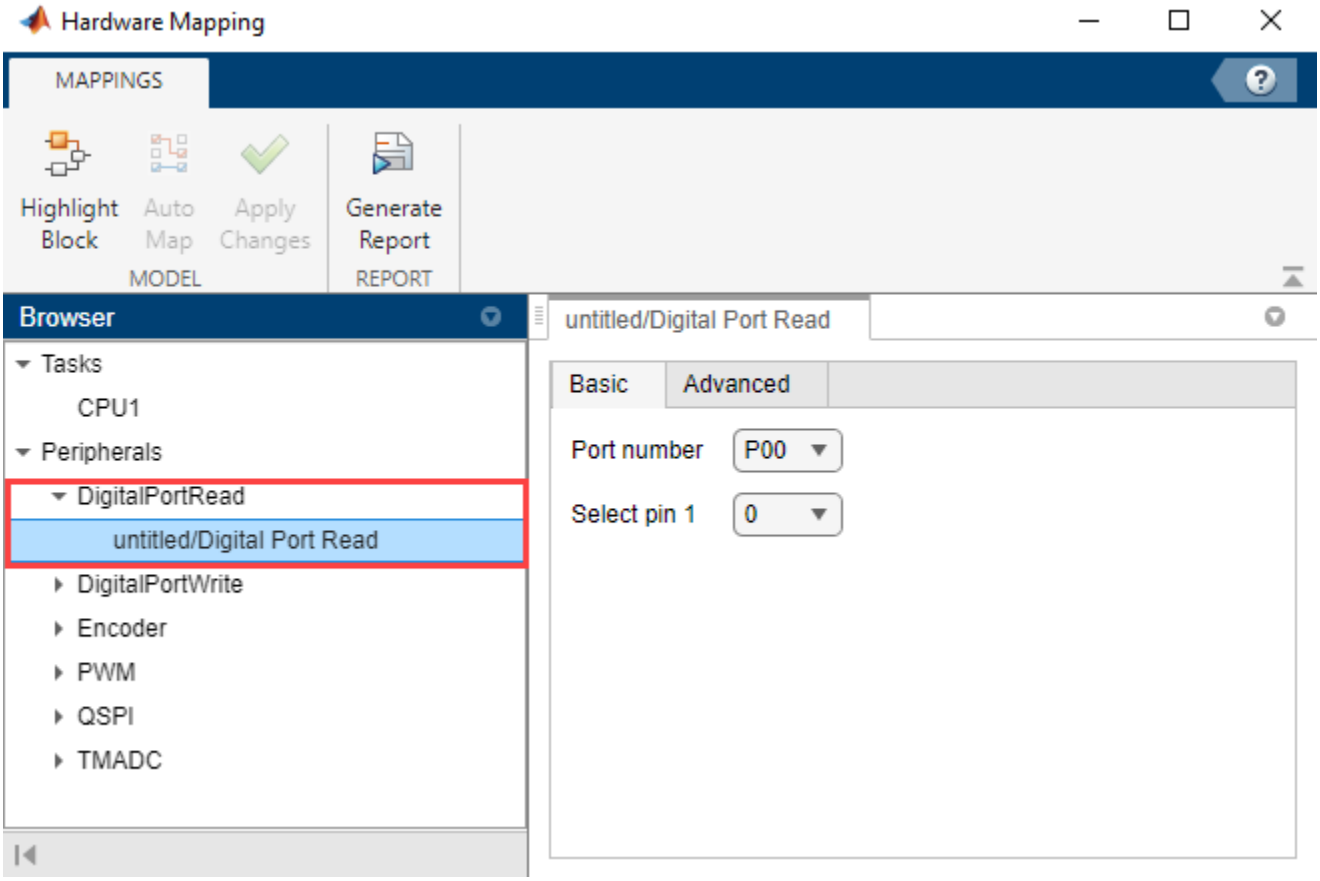
Map Digital Port Read peripherals in the Infineon AURIX model to peripheral registers in the MCU

Description

View and edit the assignment of ports and pins in the Infineon AURIX model to the hardware peripherals.

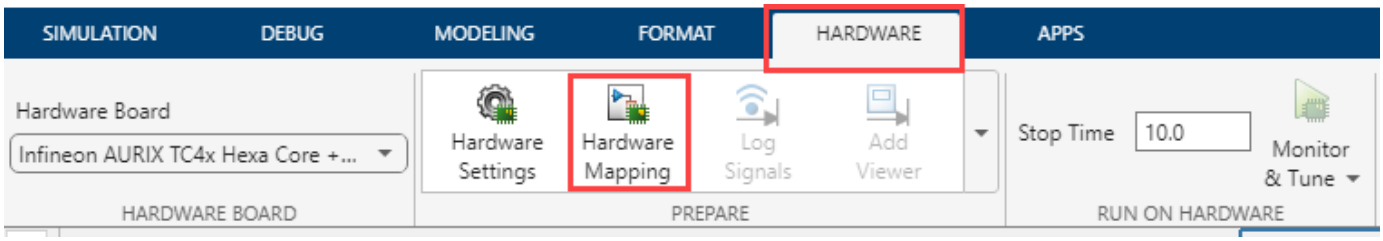
Using the **Peripheral Configuration** for Digital Port Read, you can:

- View and edit the assignment of ports and pins.
- Check your model for any conflicts between peripherals.



Open the Digital Port Read Peripheral Configuration

- In the Simulink toolstrip, go to **Hardware** tab and click **Hardware Mapping**.



Parameters

Basic

`Port number` — Port number of digital read peripheral
`P00` (default) | `P01` | `P02` | `...`

Select the port number of digital read peripheral `P00` through `P41` on the hardware board. The available ports depends on the package class and pinout selection in the Configuration Parameters.

`Select pin` — Digital read pin
`0` (default) | `1` | `2` | `...`

Select the pin number for the selected digital read port ranging between `0` through `15` on the hardware board.

The **Select pin #** is the number of pins configured in the respective digital read block.

Advanced

`Input mode` — Digital port read input mode
`Tri-state` (default) | `Pull-up` | `Push-down`

Select the input mode for the pin(s).

Note The configuration made in Advanced tab is applicable for all the pins selected in the Basic tab.

`Speed` — Digital port read speed
`Speed-1` (default) | `Speed-2` | `Speed-3`

Select the pin speed.

Note The configuration made in Advanced tab is applicable for all the pins selected in the Basic tab

`Voltage level` — Digital port voltage level
`Automotive` (default) | `TTL-5V` | `TTL-3.3V`

Select the pin voltage level.

Note The configuration made in Advanced tab is applicable for all the pins selected in the Basic tab

Version History

Introduced in R2022b

See Also

Digital Port Read | “Map Tasks and Peripherals Using Hardware Mapping” | Hardware Mapping

Digital Port Write Peripheral Configuration

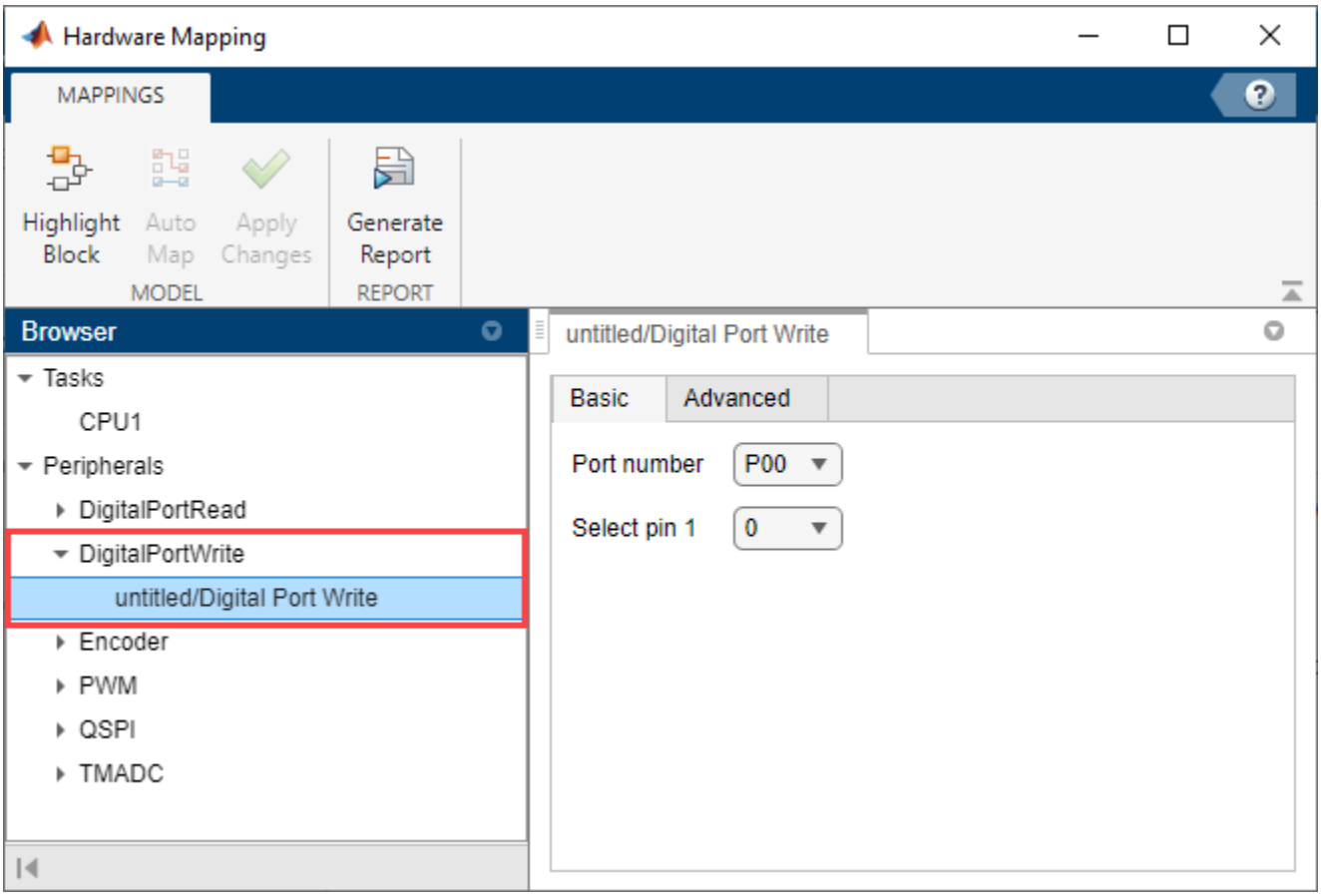
Map Digital Port Write peripherals in the Infineon AURIX model to peripheral registers in the MCU

Description

View and edit the assignment of ports and pins in the Infineon AURIX model to the hardware peripherals.

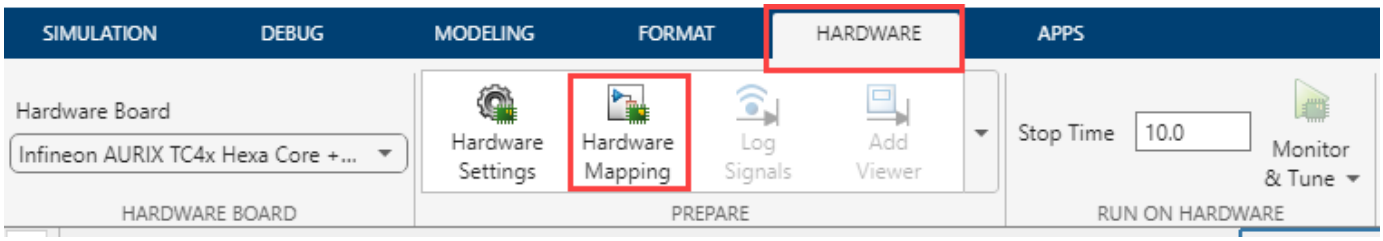
Using the **Peripheral Configuration** for Digital Port Write, you can:

- View and edit the assignment of ports and pins.
- Check your model for any conflicts between peripherals.



Open the Digital Port Write Peripheral Configuration

- In the Simulink toolstrip, go to **Hardware** tab and click **Hardware Mapping**.



Parameters

Basic

`Port number` — Port number of digital write peripheral
`P00` (default) | `P01` | `P02` | `...`

Select the port number of digital write peripheral `P00` through `P41` on the hardware board. The available ports depends on the package class and pinout selection in the Configuration Parameters.

`Select pin` — Digital write pin
`0` (default) | `1` | `2` | `...`

Select the pin number for the selected of digital write peripheral ranging between `0` through `15` on the hardware board.

The **Select pin #** is the number of pins configured in the respective digital write block.

Advanced

`Output mode` — Digital port write output mode
`Push pull` (default) | `Open drain`

Select the output mode for the pin(s).

Note The configuration made in Advanced tab is applicable for all the pins selected in the Basic tab

`Speed` — Digital port write speed
`Speed-1` (default) | `Speed-2` | `Speed-3`

Select the pin speed.

Note The configuration made in Advanced tab is applicable for all the pins selected in the Basic tab

`Voltage level` — Digital port voltage level
`Automotive` (default) | `TTL-5V` | `TTL-3.3V`

Select the pin voltage level.

Note The configuration made in Advanced tab is applicable for all the pins selected in the Basic tab

Version History

Introduced in R2022b

See Also

Digital Port Write | “Map Tasks and Peripherals Using Hardware Mapping” | Hardware Mapping

Hardware Mapping

Map tasks and peripherals in a model to hardware board configurations

Description

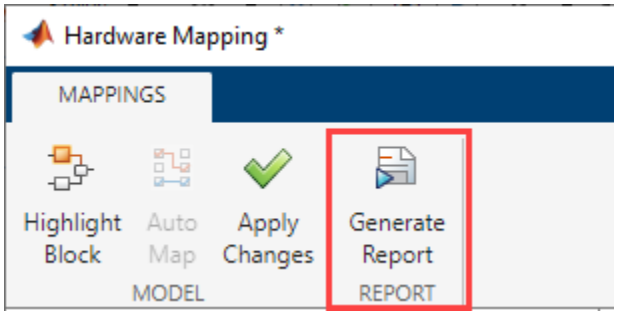
The **Hardware Mapping** tool allows you to configure the software tasks and peripherals on the selected hardware board.

- In this tool, you can map the tasks in your software model to the available event sources and interrupts:
  - Manually select the task in **Browser** > **Tasks** > ***`CPU name`***. Select the desired event or interrupt. source. Click the **Apply Changes** button in the toolstrip.

  The sources of events or interrupts depend on the choice of hardware board and peripherals available in the model.
- You can configure the global parameters. To set the group peripheral, select peripheral in **Browser** > **Peripherals** > ***`Peripheral group name`***. In the ***`Peripheral group name`*** pane, choose the appropriate settings to apply to the hardware peripheral when deployed to the hardware board. The available parameters depend on the selected hardware board for the model and the peripheral.
- In this tool, you can configure the peripheral by setting hardware specific parameters. To set the peripheral, select the peripheral in **Browser** > **Peripherals** > ***`Peripheral group name`*** > ***`Peripheral block name`***. In the *`Peripheral block name`* pane, choose the appropriate settings to apply to the hardware peripheral when deployed to the hardware board. The available parameters depend on the selected hardware board for the model and the peripheral.

After configuring all the parameters for peripherals in **Hardware Mapping**, click **Apply Changes**. Click **Generate Report** to view the hardware mapping summary report. The hardware mapping summary report with all the peripheral configurations are generated in a web page and saved in the following location:

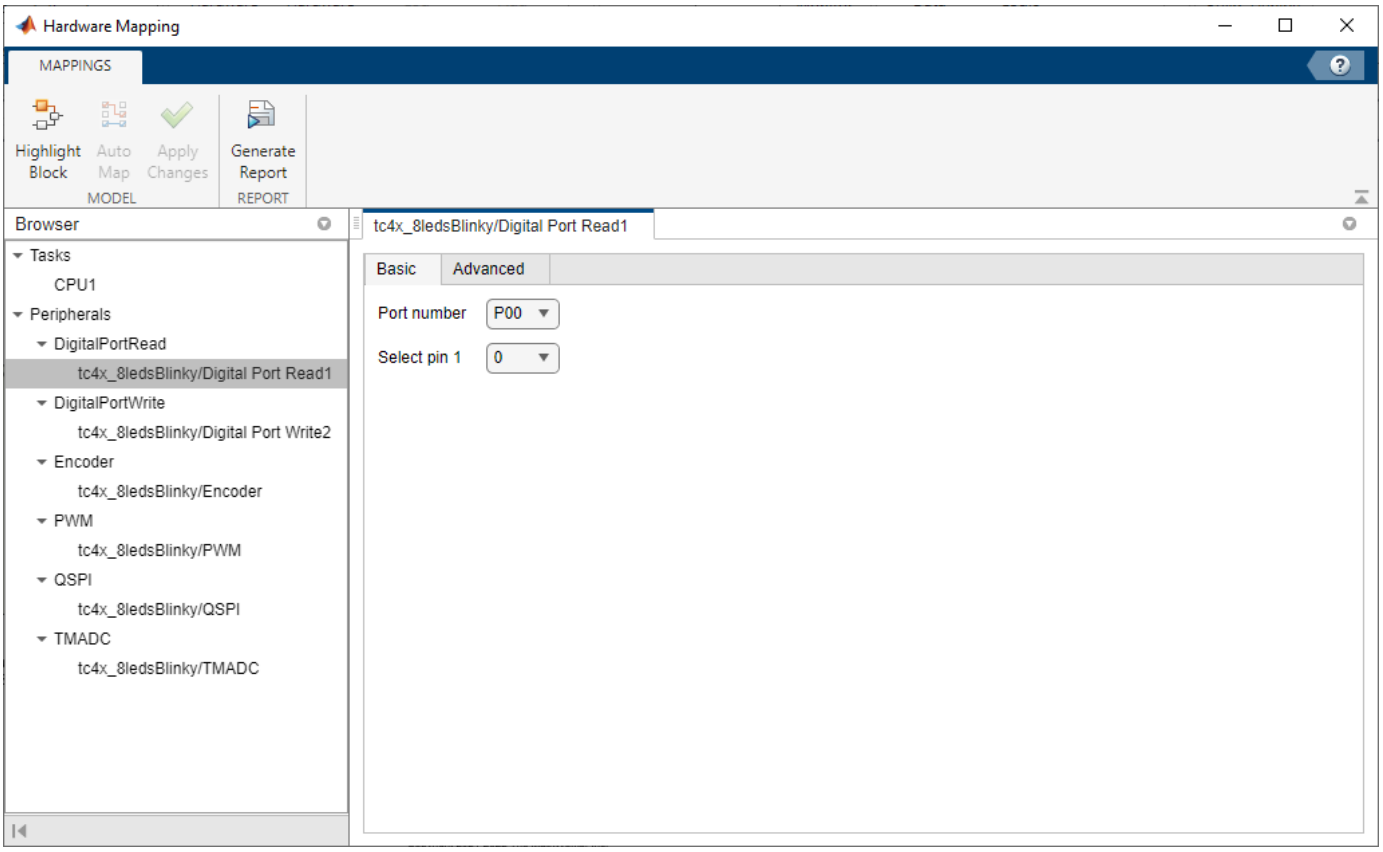
```
file:///C:/Users/<username>/AppData/Local/Temp/html/SummaryReport
```



For a complete set of parameters for available hardware boards, see:

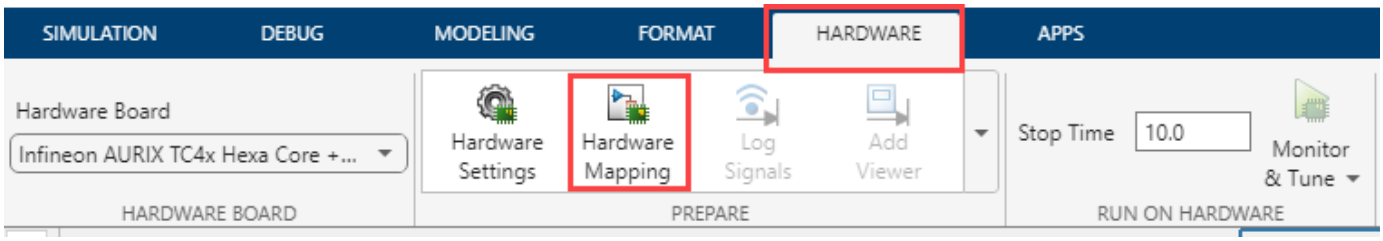
- **Digital Port Write Peripheral Configuration**
- **Digital Port Read Peripheral Configuration**

- **Encoder Peripheral Configuration**
- **PWM Peripheral Configuration**
- **QSPI Peripheral Configuration**
- **TMADC Peripheral Configuration**



Open the Hardware Mapping

- In the **Hardware** tab, click **Hardware Mapping**.



Version History

Introduced in R2022b

See Also

“Map Tasks and Peripherals Using Hardware Mapping” | Digital Port Write Peripheral Configuration | Encoder Peripheral Configuration | Digital Port Read Peripheral Configuration | PWM Peripheral Configuration | QSPI Peripheral Configuration | TMADC Peripheral Configuration