TMS320C4x Parallel Runtime Support Library User's Guide

2617613–9761 revision A
October 1994

TEXAS INSTRUMENTS

Read This First

About This Manual

This manual describes the 'C4x parallel runtime support library of functions and macros. The library provides a standard method for programming the 'C4x digital signal processor (DSP) peripherals via the C programming language at both the register and bit levels and includes a set of high- and low-level functions for multiprocessing.

The high-speed interprocessor data communication peripherals of the 'C4x include six direct memory access (DMA) channels, byte-wide communication ports, and two 32-bit timers. The peripherals are controlled through memory-mapped registers that are accessed easily through assembly or C language. Because these peripherals can run concurrently with the operations of the high-performance floating-point central processing unit (CPU), the 'C4x can maintain the throughput as well as the numerical execution required for today's parallel systems.

This revision A incorporates the following changes and additions:

- Corrections to filenames in which functions and macros are defined.
- Description of the new CLOCK_PER_SEC implementation in version 4.6 and higher of the TMS320 C compiler, which affects the alarm, elapse, sleep, time_end, and the time_left functions.
- Addition of a glossary.

How to Use This Manual

This manual includes the following chapters:

Chapter 1:	***Library Files.*** Describes library file contents, invocation, and linking.
Chapter 2:	***Header Files.*** Describes how the header files declare functions, macros, and data structures.
Chapter 3:	***Summary of Parallel Runtime Support Functions and Macros.*** Lists Parallel Runtime Support Library functions in alphabetical order by category.
Chapter 4:	***Functions Reference.*** Presents an alphabetical reference of functions with examples.
Appendix A:	***Header Files Listing.*** Lists the code of the header files.
Appendix B:	***Glossary.*** Defines pertinent terms and acronyms.

Information About Cautions

This book contains cautions.

This is an example of a caution statement.

A caution statement describes a situation that could potentially damage your software or equipment.

The information in a caution is provided for your protection. Please read each caution carefully.

References

The following publications contain information regarding functions, operations, and applications of digital signal processing—in particular, image processing. They also refer to many related technical papers.

Andrews, H.C., and Hunt, B.R., *Digital Image Restoration.* Englewood Cliffs, NJ: Prentice-Hall, Inc., 1977.

Gonzales, Rafael C., and Wintz, Paul, *Digital Image Processing.* Reading, MA: Addison-Wesley Publishing Company, Inc., 1977.

Pratt, William K., *Digital Image Processing.* New York, NY: John Wiley and Sons, 1978.

Related Documentation From Texas Instruments

TMS320 Floating-Point DSP Assembly Language Tools User's Guide (lit. number SPRU035) describes the assembly language tools (assembler, linker, and other tools used to develop assembly code), assembler directives, macros, common object file format, and symbolic debugging directives for the TMS320C3x and TMS320C4x generations of devices.

TMS320 Floating-Point DSP Optimizing C Compiler User's Guide (lit. number SPRU034) describes the TMS320 floating-point C compiler. This C compiler accepts ANSI standard C source code and produces TMS320 assembly language source code for the TMS320C3x and TMS320C4x generations of devices.

TMS320C3x Peripheral Control Library User's Guide (lit. number SPRU086) describes the TMS320C3x peripheral control library, a collection of data structures and macros, for controlling the 'C3x bus control peripherals, DMA, serial ports, and timers via the C programming language. Because this library uses the same design methodology, this document can serve as an addendum to the *TMS320C4x Parallel Runtime Support Library User's Guide.*

TMS320C4x Technical Brief (lit. number SPRU076) provides an overview of the TMS320C4x 32-bit floating-point processor. The brief includes an architectural overview, mechanical descriptions, TMS320C4x C compiler description, parallel runtime support library functions, hardware development tools, a TIM-40 overview, and an alphabetical listing of third-party support products.

TMS320C4x User's Guide (lit. number SPRU063) describes the TMS320C4x 32-bit floating-point processor, developed for parallel-processing digital signal processing as well as general applications. Covered are its architecture, internal register structure, instruction set, pipeline, specifications, and operation of its DMA channels and communication ports. Software and hardware applications are included.

If You Need Assistance. . .

If you want to. . .	Do this. . .
Order Texas Instruments documentation	Call the Literature Response Center at **(800) 477–8924**
Obtain technical support, report suspected problems	Call the DSP hotline at **(713) 274–2320**, send a FAX to **(713) 274–2324** or to **+33–1–3070–1032** in Europe. You can also send email to **4389750@mcimail.com.**
Obtain TI product updates, application software	Dial the TMS320 Bulletin Board Service (BBS) at **(713) 274–2323** (24 hrs.). Set your modem to 8 bits,1 stop bit, no parity. Supported speeds are from 300 to 14400 bps. In Europe, dial **+44–2–3422–3248**
Access the TMS320 BBS from Internet	Connect via anonymous ftp to **ti.com** (192.94.94.5), subdirectory /pub/mirrors, or to **evans.ee.adfa.oz.au** (131.236.30.24), subdirectory /mirrors/tibbs
Report mistakes or offer suggestions regarding this document or any other TI documentation	Send your comments to: **Texas Instruments Incorporated** **Technical Publications Manager, MS 702** **P.O. Box 1443** **Houston, Texas 77251–1443** or send email to: **comments@books.sc.ti.com**

Contents

Tables

Chapter 1

Library Files

The source and header files of the TMS320C4x Parallel Runtime Support Library are stored in the prts40.src file. You must build the object library before linking the files. For example, the following steps build an object library for the small memory model, using the stack-passing parameter convention:

```
mk30 -v40 --h -o2 -mn prts40.src ; build the library
```

The –o2 option specifies a level 2 optimization. The – –h compiler option installs the header files after building the object library. You should include the corresponding header files in your program when you use the PRTS40 library functions. If your program is compiled with a particular compiler option, such as the large memory model (–mb) or the register-passing parameter convention (–mr), the entire PRTS40 object library must be recompiled with that particular compiler option. The following two examples show how to build a large memory model and a register-passing parameter convention object library.

```
mk30 -v40 --h -o2 -mn -mr prts40.src     ; build the register
                                         ; passing parameter
                                         ; convention library

mk30 -v40 --h -o2 -mn -mb prts40.src     ; build the large
                                         ; memory model
                                         ; library
```

You can also build the library as follows:

```
ar30 -x prts40.src                      ; extracts all files
cl30 -v40 -o2 -mn -c *.c *.asm          ; compiling
ar30 -a prts40.lib *.obj                ; build the object
                                        ; library
```

You can use many other compiler options to compile the PRTS40 library. For more information about the 'C4x C compiler, refer to the *TMS320 Floating-Point DSP Optimizing C Compiler User's Guide* (literature number SPRU034). For information about debugging C source code, refer to the *TMS320C4x C Source Debugger User's Guide* (literature number SPRU054).

You can inspect or modify library functions by using the archiver to extract the appropriate source file from the prts40.src file, as shown above. For more information about the archiver, refer to the *TMS320 Floating-Point DSP Assembly Language Tools User's Guide* (literature number SPRU035).

During program linking, the PRTS40 object library must be specified as an input file to the linker so that references to the parallel runtime support functions can be resolved. Libraries are usually specified last on the linker command line because the linker searches for unresolved references when it encounters a library on the command line. When a library is linked, the linker includes only those library members required to resolve undefined references. For more information about the linker, refer to the *TMS320 Floating-Point DSP Assembly Language Tools User's Guide* (literature number SPRU035).

Because prts40.lib contains references to routines defined in rts40.lib, **you should specify prts40.lib before rts40.lib**, to avoid unresolved references. This is shown in the example linker.cmd file in Example 1–1.

Example 1–1. Linker Command File Example (linker.cmd)

```
main.obj
    .
    .
    .
< other user-defined libraries >
    .
    .
    .
prts40.lib
rts40.lib
```

Another way to avoid unresolved references is to use the linker –x option to force the linker to reread all libraries until references are resolved.

Chapter 2

Header Files

The functions in the Parallel Runtime Support library are categorized as communication port, DMA, interrupt, multiprocessor, and timer functions. Each category has its own header file.

- compt40.h
- dma40.h
- intpt40.h
- mulpro40.h
- timer40.h

Each parallel runtime support function is declared in a header file; the file declares:

- A set of related functions (or macros)
- Any data types required to use the functions
- Any macros required to use the functions
- Any function definitions required for using the inline function option

This chapter explains how to use header files and describes the contents of each file:

Topic **Page**

2.1 How Header Files Work

To use a parallel runtime support function, **you must first use the #include preprocessor directive to include the header file that declares the function**. For example, since the elapse() function is declared by the timer40.h header file, you must include the timer40.h header file, as shown, before you use the elapse function.

```
#include    <timer40.h>
     .
     .
     .
  tim0 = elapse();
```

The header file can be included in any order. However, it must be included before you refer to any of the functions or objects that it declares. The source code of these header files is included in Appendix A.

Header files declare macros that use #define to perform macro substitution to improve readability. For example, to assign *dma_ptr to point to DMA channel # 5, use the macro DMA_ADDR(n):

```
DMA_REG   *dma_ptr = DMA_ADDR(5);
```

In general, the names of the macros and data structures are in uppercase, and the function names are in lowercase.

All the header files except the intpt40.h file define several data structures for the control registers of the 'C4x peripherals. These data structures provide easy readable methods of controlling 'C40 peripheral functions through C. The example below illustrates the structure convention by showing how the data structure COMPORT_REG can be used to halt the input FIFO of communication port channel # 2:

```
COMPORT_REG *cp_ptr=COMPORT_ADDR(2); /* Point to comm port 2   */
cp_ptr->gcontrol_bit.ich=1;        /* Halt the input FIFO     */
```

The peripheral control registers typically contain bit fields that control different aspects of the peripheral. The data structures provide two methods of accessing the control register. The first method is through bit-field structures, the other is by integer assignment. In the previous example, if both the input and output FIFO need to be halted, the following bit-field assignment statement can be added to the statements of the previous example:

```
cp_ptr->gcontrol_bit.och = 1; /* Halt the output FIFO        */
```

Alternately, by integer assignment, this statement:

```
cp_ptr->gcontrol = 0x18; /* halt the input/output FIFO       */
```

replaces the cp_ptr–>gcontrol_bit.ich = 1; and cp_ptr–>gcontrol_bit.och = 1; statements. Refer to Appendix A for complete information regarding the structure names.

2.2 Communication Port Functions (compt40.h)

The compt40.h header file declares three kinds of communication port functions: synchronous transfer, asynchronous transfer, and communication port control.

- **Synchronous transfer functions**—The synchronous communication port transfer functions use the CPU to transfer data between memory and the six 'C4x communication ports. They support byte, halfword, and word-wide data transfer. If byte- or halfword-wide data transfer is used, the functions handle the data packing and unpacking.
  - out_msg8()
  - in_msg8()
  - out_msg16()
  - in_msg16()
  - out_word()
  - in_word()
  - out_msg()
  - in_msg()
- **Asynchronous transfer functions**—The asynchronous communication port transfer functions use DMA autoinitialization and communication port flag synchronization mode to perform the data transfer. This allows concurrent data I/O along with CPU computation.
  - send_msg()
  - receive_msg()
- **Communication port control functions**—The communication port control functions configure and determine the status of the communication ports.
  - cp_in_level()
  - cp_out_level()
  - cp_in_halt()
  - cp_out_halt()
  - cp_in_release()
  - cp_out_release()

The compt40.h header also declares three macros and two data types. The three macros are COMPORT_ADDR, COMPORT_IN_ADDR, COMPORT_OUT_ADDR. These macros are used to set up communication port channel pointers. The two data types are:

- COMPORT_CONTROL, a union data type that unionizes an unsigned-long and a structure data type that defines the bit-field functions of the communication port control register, and
- COMPORT_REG, a structure data type that describes the communication port registers.

2.3 DMA Functions (dma40.h)

The DMA functions set up the DMA channels for different transfer tasks, such as unified mode, split mode, autoinitialization mode, synchronization mode, etc., and also enable external signal triggers and complex FFT bit-reversed DMA transfers. The dma40.h header file declares three kinds of DMA functions: high-level, user-customized, and DMA control.

High-level DMA functions provide one-step high-level DMA for data transfer. You don't need hardware knowledge to implement the DMA data transfer. The high-level DMA functions are:

- ❑ dma_move()
- ❑ dma_int_move()
- ❑ dma_cmplx()

User-customizable DMA functions provide an easy way for you to design your own DMA data transfers. These DMA functions include DMA setup functions and DMA start functions.

The DMA setup functions set up the DMA autoinitialization table control for unified mode and primary/auxiliary channel in split mode. The DMA control-word setup can be customized through a DMA_CONTROL data structure. The DMA setup functions are:

- ❑ set_dma_auto()
- ❑ set_pri_auto()
- ❑ set_aux_auto()

The DMA start functions provide different ways to start the DMA function that has been set up by DMA setup functions. The DMA start functions are:

- ❑ dma_go()
- ❑ dma_auto_go()
- ❑ dma_extrig()
- ❑ dma_prigo(), dma_auxgo()

DMA control functions allow you to set the DMA-related CPU registers, DIE and IIF, and check the status of the DMA channels. The DMA control functions are

- ❑ chk_pri_dma()
- ❑ chk_aux_dma()
- ❑ chk_dma()

The dma40.h header also declares nine macros and six data types. The nine macros are:

- DMA_ADDR
- DMA_RESET
- DMA_HALT
- DMA_HALT_B
- DMA_RESTART
- DMA_AUX_RESET
- DMA_AUX_HALT
- DMA_AUX_HALT_B
- DMA_AUX_RESTART

The DMA_ADDR macro is used to set up the different DMA channel pointers. The other macros are used to start and stop the DMA channels. The six data types are

- DMA_CONTROL, a union data type that unionizes an unsigned-long and a structure data type that defines the bit-field functions of the DMA global control register,
- DMA_REG, a structure data type that describes the 9 DMA registers,
- DMA_REGSET, a structure data type that describes the subset of the nine DMA registers,
- DMA_PRI_REG, a structure data type that describes the five DMA split-mode primary-channel registers,
- DMA_AUX_REG, a structure data type that describes the five DMA split-mode auxiliary-channel registers, and
- AUTOINIT, a structure data type that describes two sets of DMA autoinitialization tables for dma_int_move, dma_cmplx(), and dma_extrig() functions.

2.4 Interrupt Functions (intpt40.h)

Interrupt processing is the way in which DSP programs handle different tasks according to their priority. The interrupt functions in this module support the programming task of writing an interrupt-handling routine in the C by providing access to the vector table and to the CPU interrupt registers.

The intpt40.h header declares four kinds of interrupt functions: CPU-register setup, CPU-register check-out, general-purpose I/O, and vector-setup:

- **CPU-register setup** functions provide write access to the TMS320C40 CPU registers – DIE, IIE, and IIF—which can be accessed by either bit-field or integer assignment. The functions in this category are:

  load_die(), dma_sync_set(), load_iie(), set_iie(), reset_iie(), load_iif(), set_iif_flag(), reset_iif_flag(), and set_iiof().

- **CPU-register check-out** functions allow the 'C4x CPU registers to be read or checked. The functions in this category are

  chk_iie(), chk_iif_flag(), st_value(), die_value(), iie_value(), iif_value(), ivtp_value(), and tvtp_value().

- **General-purpose I/O** functions control IIOF pins of the 'C4x when they are configured for general-purpose rather than for interrupt. These functions are:

  iiof_in() and iiof_out().

- **Vector setup** functions provide a method to install and deinstall the interrupt (or trap) vector and vector table pointer. The functions in this category are:

  install_int_vector(), deinstall_int_vector(), set_ivtp(), reset_ivtp(), set_tvtp(), and reset_tvtp().

The intpt40.h header also declares fourteen macros. The macros are:

INT_ENABLE	CACHE_FREEZE	GET_IIE
INT_DISABLE	CACHE_DEFROST	GET_IIF
CACHE_ON	CPU_IDLE	GET_IVTP
CACHE_OFF	GET_ST	GET_TVTP
CACHE_CLEAR	GET_DIE	

The CPU_IDLE macro is an in-line assembly function for an idle instruction. The INT_ENABLE, INT_DISABLE, CACHE_ON, CACHE_OFF, CACHE_CLEAR, CACHE_FREEZE, and CACHE_DEFROST macros are used to set up the CPU status register. The rest of the macros are used to read the CPU register values.

2.5 Multiprocessor Functions (mulpro40.h)

Multiprocessor systems often use semaphores to arbitrate for shared memory. Processor identification is also useful in many parallel-processing systems. The mulpro40.h header defines a processor identification macro and declares two multiprocessor functions that allow you to set up a processor identification to use in the program and to set and release shared-memory semaphores.

The MY_ID() macro reads a processor-identification number from a predefined-memory location: 0x2FFF00 in internal RAM block 1. If location 0x2FFF00 conflicts with the memory use of your program, you can use the #define ID_ADDR preprocessor directive to change the processor-identification number location. **The rts40.lib should be rebuilt with mk30 for this change to take effect**.

The lock() and unlock() multiprocessor functions implement the interlock instructions for accessing a shared-memory semaphore.

The mulpro40.h header also declares two data types:

- BUS_CONTROL, a union data type that unionizes an unsigned-long and a structure data type that defines the bit-field functions of the external bus control register, and
- BUS_CTRL_REG, a structure data type that describes the local and global bus control registers.

2.6 Timer Functions (timer40.h)

The timer40.h header file declares three kinds of timer functions: high-level, low-level, and general-purpose I/O. The high-level and low-level functions facilitate setting up the timers. You need no knowledge of the 'C4x timer architectures to use the high-level functions. The general-purpose I/O functions allow you to use the timer pins as general-purpose I/O pins. The functions are:

- High-level timer functions
  - time_go()
  - time_run()
  - elapse()
  - time_end()
  - alarm()
  - time_left()
  - sleep()
- Low-level timer functions
  - time_start()
  - time_read()
  - time_stop()
  - count_down()
  - count_left()
  - time_delay()
- General-purpose I/O functions
  - out_timer()
  - in_timer

Note:

The timer40.h header declares as an external the CLOCK_PER_SEC global variable defined in timer40.c, that sets the default processor speed at 40 ns. You may need to modify this variable to change the default processor speed, such as when running a 'C4x at 50 ns (see example for elapse() function).

The timer40.h file also defines several macros. One of them is TIMER_ADDR, which provides the addresses for the 'C4x timers.

Two data types are also defined in timer40.h:

- TIMER_CONTROL, a union data type that unionizes an unsigned long and a structure data type that defines the bit-field functions of the timer control register, and
- TIMER_REG, a structure data type that describes the timer registers.

Chapter 3

Summary of Parallel Runtime Support Functions and Macros

This chapter lists and describes all of the parallel runtime support functions and macros by category. Chapter 4 describes each function and macro in detail, including the syntax and an example.

The topics covered in this chapter include:

Topic **Page**

3.1 Communication Port Functions and Macros

Table 3–1. Communication Port Functions and Macros

Macro	Description
COMPORT_REG *COMPORT_ADDR(int ch_no);	Sets up a structure pointer to communication port channel number register address.
long *COMPORT_IN_ADDR(int ch_no);	Sets up a pointer to communication port channel number input register address.
long *COMPORT_OUT_ADDR(int ch_no);	Sets up a pointer to communication port channel number output register address.
Function	**Description**
int cp_in_level(int ch_no);	Checks the communication port channel number input buffer level.
int cp_out_level(int ch_no);	Checks the communication port channel number output buffer level.
long in_word(int ch_no);	Reads one-word data from communication port channel number.
size_t in_msg(int ch_no, void *message, int step);	Reads data from communication port channel number to a word array that is pointed to by *message with the pointer step size (step).
size_t in_msg8(int ch_no, void *message);	Reads data from communication port channel number and unpacks it to byte-wide-data-array message.
size_t in_msg16(int ch_no, void *message);	Reads data from communication port channel number and unpacks it to 16-bit-wide-data-array message.
size_t unpack_byte(void *pack_msg, void *msg, size_t in_size);	Reads in_size 32-bit data from pack_msg and unpacks them to byte-wide data array message.
size_t unpack_halfword(void *pack_msg, void *msg, size_t in_size);	Reads in_size 32-bit data from pack_msg and unpacks them to 16-bit-wide data array message.
void cp_in_halt(int ch_no);	Halts the communication port channel number input port.
void cp_in_release(int ch_no);	Unhalts the communication port channel number input port.
void cp_out_halt(int ch_no);	Halts the communication port channel number output port.
void cp_out_release(int ch_no);	Unhalts the communication port channel number output port.
void out_msg(int ch_no, void *message, size_t message_size, int step);	Writes message_size words data from message with the pointer step size (step) to communication port channel number.
void out_msg8(int ch_no, void *message, size_t message_size);	Packs message_size bytes data to 32-bit data from message and writes them to communication port channel number.

Table 3–1. Communication Port Functions and Macros (Concluded)

Function	**Description**
void out_msg16(int ch_no, void *message, size_t message_size);	Packs message_size 16-bit data to 32-bit data from memory address *message and writes them to communication port channel number.
void out_word(int ch_no);	Writes one-word data to communication port channel number.
void pack_byte(void *message, void *pack_msg, size_t in_size);	Packs in_size bytes data to 32-bit data from memory address *message and writes them to memory location *pack_msg.
void pack_halfword(void *message, void *pack_msg, size_t in_size);	Packs in_size 16-bit data to 32-bit data from memory address *message and writes them to memory location *pack_msg.
void receive_msg(int ch_no, void *message, int step);	Asynchronously reads data from communication port channel number to message with a given pointer step size (step).
void send_msg(int ch_no, void *message, size_t message_size, int step);	Asynchronously writes message_size words data from memory addresss *message with a given pointer step size (step) to communication port channel number.

3.2 DMA Functions

Table 3–2. DMA Functions

Function	Description
int chk_aux_dma(int ch_no);	Checks if DMA channel number auxiliary channel is busy.
int chk_dma(int ch_no);	Checks if DMA channel number primary or auxiliary channel is busy.
int chk_pri_dma(int ch_no);	Checks if DMA channel number primary channel is busy.
void dma_auto_go(int ch_no, long ctrl, void *link_tab);	Starts DMA channel number with autoinitialization.
void dma_auxgo(int ch_no, DMA_AUX_REG *reg);	Starts DMA channel number auxiliary channel with DMA configuration by reg structure pointer.
void dma_cmplx(int ch_no, void *src, void *dest, size_t fft_size, int priority);	Sets up DMA channel number to transfer fft_size data from source to destination with complex bit-reverse address and configured CPU/DMA priority.
void dma_extrig(int ex_int, int ch_no, DMA_REG *reg);	Sets up DMA channel number with DMA configuration by the register structure pointer to be triggered by the external interrupt (ex_int).
void dma_go(int ch_no, DMA_REG *reg);	Starts DMA channel number with DMA configuration by the register structure pointer.
void dma_int_move(int ex_int, int ch_no, void *src, void *dest, size_t length);	Sets up DMA channel number to transfer length data from source to destination to be triggered by the external interrupt ex_int.
void dma_move(int ch_no, void *src, void *dest, size_t length);	Sets up DMA channel number to transfer length data from source to destination.
void dma_prigo(int ch_no, DMA_PRI_REG *reg);	Starts DMA channel number primary channel with DMA configuration by the register structure pointer.
void set_aux_auto(void *tab_addr, long ctrl, void *dest, int dest_idx, size_t length, void *next_tab);	Sets DMA auxiliary channel autoinitialization table.
void set_dma_auto(void *tab_addr, long ctrl, void *src, int src_idx, size_t length, void *dest, int dest_idx, void *next_tab);	Sets DMA unified mode autoinitialization table.
void set_pri_auto(void *tab_addr, long ctrl, void *src, int src_idx, size_t length, void *next_tab);	Sets DMA primary-channel autoinitialization table.

3.3 DMA Macros

Table 3–3. DMA Macros

Macro	**Description**
DMA_REG *DMA_ADDR(int ch_no);	Sets up a structure pointer to DMA channel number register address.
void DMA_AUX_HALT(int ch_no);	Halts DMA channel number auxiliary channel in read or write boundary.
void DMA_AUX_HALT_B(int ch_no);	Halts DMA channel number auxiliary channel in read/write boundary.
void DMA_AUX_RESET(int ch_no);	Resets DMA channel number auxiliary channel.
void DMA_AUX_RESTART(int ch_no);	Restarts DMA channel number auxiliary channel.
void DMA_HALT(int ch_no);	Halts DMA channel number primary channel in read or write boundary.
void DMA_HALT_B(int ch_no);	Halts DMA channel number primary channel in read/write boundary.
void DMA_RESET(int ch_no);	Resets DMA channel number primary channel.
void DMA_RESTART(int ch_no);	Restarts DMA channel number primary channel.

3.4 Interrupt Functions

Table 3–4. Interrupt Functions

Function	Description
void deinstall_int_vector(int N);	Restores interrupt N vector from int_vect_buf[N] to memory location IVTP+N.
int die_value();	Reads DIE register value.
void dma_sync_set(int ch_no, int bit_value, int r_w);	Sets corresponding bits of DMA channel number synchronization mode in DIE register.
int iie_value();	Reads IIE register value.
int iif_value();	Reads IIF register value.
int iiof_in(int pin_no);	Sets IIOF pin number as general-purpose input pin and returns the IIOF pin value.
void iiof_out(int pin_no, int flag);	Sets IIOF pin number as general-purpose output pin and sets the IIOF pin number value.
void install_int_vector(void *isr, int N);	Sets isr address to memory location IVTP+N and saves the old vector.
int ivtp_value();	Reads IVTP register value.
int chk_iie(int bit_no);	Checks the status of the bit number of the IIE register.
int chk_iif_flag(int bit_no);	Checks the status of the bit number of the IIF register.
void load_die(unsigned long die_value);	Loads data into the DIE register.
void load_iie(unsigned long iie_value);	Loads data into the IIE register.
void load_iif(unsigned long iif_value);	Loads data into the IIF register.
void reset_iie(int bit_no);	Clears the bit number of the IIE register.
void reset_iif_flag(int bit_no);	Clears the bit number of the IIF register.
void reset_ivtp();	Restores IVTP value from global memory ivtp_buf.
void reset_tvtp();	Restores TVTP value from global memory tvtp_buf.
void set_iie(int bit_no);	Sets the bit number of the IIE register.
void set_iif_flag(int bit_no);	Sets the bit number of the IIF register.
void set_iiof(int ch_no, int iiof_value);	Loads data iiof_value to the IIOF channel number field of the IIF register.
void set_ivtp(*isr);	Sets IVTP point to isr address. The default is pointed to the vector section.
void set_tvtp(*isr);	Sets TVTP point to isr address.
int tvtp_value();	Reads TVTP register value.
int st_value();	Reads the status register value.

3.5 Interrupt Macros

Table 3–5. Interrupt Macros

Macro	Description
void CACHE_CLEAR();	Clears the 'C4x on-chip cache.
void CACHE_DEFROST();	Takes the 'C4x out of the cache freeze mode.
void CACHE_FREEZE();	Freezes the 'C4x on-chip cache function.
void CACHE_OFF();	Turns off the 'C4x on-chip cache function.
void CACHE_ON();	Turns on the 'C4x on-chip cache function.
void CPU_IDLE();	Sets 'C4x CPU idle to wait for interrupt.
void GET_DIE();	Loads DIE register to R0. It is used in the die_value function.
void GET_IIE();	Loads IIE register to R0. It is used in the iie_value function.
void GET_IIF();	Loads IIF register to R0. It is used in the iif_value function.
void GET_IVTP();	Loads IVTP register to R0. It is used in the ivtp_value function.
void GET_ST();	Loads ST register to R0. It is used in the st_value function.
void GET_TVTP();	Loads TVTP register to R0. It is used in the tvtp_value function.
void INT_DISABLE();	Disables the 'C4x CPU interrupt function globally.
void INT_ENABLE();	Enables the 'C4x CPU interrupt function globally.

3.6 Multiprocessor Functions and Macros

Table 3–6. Multiprocessor Functions and Macros

Macro	**Description**
int MY_ID();	Reads the processor-identification number.
Function	**Description**
int lock(int *semaphore);	Returns the value of the semaphore and sets it to one.
void unlock(int *semaphore);	Sets the semaphore to zero.

3.7 Timer Functions and Macros

Table 3–7. Timer Functions and Macros

Macro	**Description**
TIMER_REG *TIMER_ADDR(int ch_no);	Sets up a structure pointer to timer channel number register address.
Function	**Description**
float elapse();	Returns the elapsed time in seconds since the time_go function was executed.
float time_end();	Stops timer 0 and returns the elapsed time in seconds since the execution of the time_go function.
float time_left();	Returns the value of the difference between the timer 0 period and counter registers in seconds.
int count_left(int t);	Returns the value of the difference between timer t period and counter registers.
int in_timer(int t);	Returns the TCLK t value when it is configured as a general-purpose input pin.
int time_read(int t);	Returns the value in the timer t counter register.
int time_stop(int t);	Stops the timer t and returns the value in the counter register.
void alarm(float x);	Starts timer 0 with x seconds in the period register.
void count_down(int t, unsigned long x);	Starts timer t with x in the period register.
void c_int45();	Adds one to memory time_count when timer 0 interrupt occurs.
void install_int_vector(void *isr, int N);	Sets isr address to memory location IVTP+N.
void out_timer(int t, int flag);	Sets the TCLK t value when it is configured as a general-purpose output pin.
void sleep(float x);	Delays CPU operation for x seconds.
void time_delay(unsigned long x);	Delays CPU operation x cycles.
void time_run();	Starts timer 0 with a 64-bit counter size.
void time_start(int t);	Starts timer t with a period register equal to –1.
void wakeup();	Disable the timer 1 interrupt in the IIE register when timer 1 interrupt occurs.

Chapter 4

Functions Reference

This chapter is a reference of functions organized alphabetically, one function per page. Refer to the page indicated in the list below for details on a function.

Syntax

```
#include <timer40.h>
void alarm(float x);
```

Parameters

x — time (in seconds) before alarm (interrupts) activates

Defined in

alarm.c in prts40.src or in timer40.h (if INLINE option is used)

Description

The *alarm* function starts timer 0 with the period register equal to approximately x seconds. This function is designed for *alarm-clock*-type applications. The timer interrupt flag will be set after x seconds. A user-defined timer 0 interrupt active is executed if the GIE bit, IIE register, and interrupt vector are all configured properly.

Note:

The speed of the processor is target-specific. The default value of CLOCK_PER_SEC is set to 25000000.0 by prts40.src, corresponding to a 50-MHz-input-clock 'C4x. For a different processor speed, you must initialize in your program the global variable CLOCK_PER_SEC (defined in timer40.c) to the desired value. CLOCK_PER_SEC must be set to half the number of input system clocks per second. For example, for a 40-MHz input clock 'C4x:

If X2/XCLKIN = 40 MHz, then H1 = X2/XCLKIN ÷ 2 = 20 MHz
so CLOCK_PER_SEC = H1 = 20000000.0

Example

Set up timer 0 to set the flag every 1 millisecond with the device speed at 40 MHz clock input.

```
#include  <timer40.h>
extern    float  CLOCK_PER_SEC=20000000.0;
float     x = 0.001, y = 0.00001, z;

alarm(x);           /* start timer 0 function                */

sleep(y);           /* Delay CPU operation for y seconds     */

z = time_left();    /* Check the remaining time in seconds   */
```

Related Functions

count_down

Syntax

```
#include <timer40.h>
void c_int45(void);
```

Parameters None

Defined in time_int.c in prts40.src

Description The *c_int45* timer 0 interrupt service routine extends the timer counter to 64 bits via the time_run function. When the interrupt occurs, it adds 1 to the external global variable time_count. Therefore, a 64-bit timer counter can be obtained by combining the time_count variable and the timer 0 counter register.

Example Refer to the source code of the time_run function in the prts40.src file.

Related Functions time_run, set_ivtp, install_int_vector

Syntax

#include <intpt40.h>
void CACHE_CLEAR(void);

Parameters

None

Defined in

intpt40.h (as a macro)

Description

The *CACHE_CLEAR* macro sets bit 12 of the 'C4x status register, ST, clearing the 'C4x on-chip cache.

Example

Clear the 'C4x on-chip cache.

```
CACHE_CLEAR();        /* Clear the Cache      */
```

Related Macros

CASH_DEFROST, CACHE_FREEZE, CACHE_OFF, CACHE_ON

Syntax

#include <intpt40.h>
void CACHE_DEFROST(void);

Parameters

None

Defined in

intpt40.h (as a macro)

Description

The *CACHE_DEFROST* macro resets bit 10 of the 'C4x status register, ST, unfreezing the 'C4x on-chip cache.

Example

Take the 'C4x on-chip cache out of freeze mode.

```
CACHE_DEFROST();     /* Defrost the Cache    */
```

Related Macros

CACHE_CLEAR, CACHE_FREEZE, CACHE_OFF, CACHE_ON

Syntax #include <intpt40.h>
void CACHE_FREEZE(void);

Parameters None

Defined in intpt40.h (as a macro)

Description The *CACHE_FREEZE* macro sets bit 10 of the 'C4x status register, ST, freezing the 'C4x on-chip cache.

Example Freeze the 'C4x on-chip cache function.

```
CACHE_FREEZE();       /* Freeze the Cache     */
```

Related Macros CACHE_CLEAR, CACHE_DEFROST, CACHE_OFF, CACHE_ON

Syntax

#include <intpt40.h>
void CACHE_OFF(void);

Parameters

None

Defined in

intpt40.h (as a macro)

Description

The *CACHE_OFF* macro resets bit 11 of the 'C4x status register, ST, disabling the 'C4x on-chip cache.

Example

Turn off the 'C4x on-chip cache.

```
CACHE_OFF();     /* Turns off the Cache     */
```

Related Macros

CACHE_CLEAR, CACHE_DEFROST, CACHE_FREEZE, CACHE_ON

Syntax	#include <intpt40.h>
	void CACHE_ON(void);

Parameters	None
Defined in	intpt40.h (as a macro)
Description	The *CACHE_ON* macro sets bit 11 of the 'C4x status register, ST, enabling the 'C4x on-chip cache.
Example	Turn on the 'C4x on-chip cache.

```
CACHE_ON();       /* Turns on the Cache       */
```

Related Macros CACHE_CLEAR, CACHE_DEFROST, CACHE_FREEZE, CACHE_OFF

Syntax

#include <dma40.h>
int chk_aux_dma(int ch_no);

Parameters

ch_no — DMA channel number (0–5)

Defined in

chk_aux_.c in prts40.src or in dma40.h (if INLINE option is used)

Description

The *chk_aux_dma* function checks whether a specified DMA auxiliary channel is being used. If the return value equals 1, the DMA auxiliary channel is used. If the return value equals 0, the DMA auxiliary channel is free.

Example

Check if the auxiliary channel of DMA #3 is busy before initializing channel. You must initialize DMA_CTRL, dest, dest_indx, and size values.

```
DMA_AUX_REG tab;

     .
     .
     .

set_aux_auto(&tab, DMA_CTRL, dest, dest_indx, size, 0);

while(chk_aux_dma(3)); /* Check if auxiliary channel of
                          the DMA # 3 is busy             */
dma_auxgo(3, &tab);    /* Start DMA #3 auxiliary transfer */
```

Related Functions

chk_dma, chk_pri_dma, dma_auxgo, set_aux_auto

Syntax

#include <dma40.h>
int chk_dma(int ch_no);

Parameters

ch_no — DMA channel number (0–5)

Defined in

chk_dma.c in prts40.src or in dma40.h (if INLINE option is used)

Description

This *chk_dma* function checks whether a specified DMA channel is in use in either the primary or auxiliary channel. If the return value equals 1, the DMA is in use. If the return value equals 0, the DMA is not in use.

Example

Check if DMA #2 is busy. You need to initialize DMA_CTRL, src, idxs, size, dest, and idxd values.

```
DMA_REG   tab;
     .
     .
     .
set_dma_auto(&tab, DMA_CTRL, src, idxs, size, dest, idxd, 0);

while(chk_dma(2));      /* Check if the DMA # 2 is busy      */
dma_go(2, &tab);        /* Start DMA #2 transfer             */
```

Related Functions

chk_aux_dma, chk_pri_dma, dma_auxgo, set_aux_auto

Syntax

#include <intpt40.h>
int chk_iie(int bit_no);

Parameters

bit_no — IIE register bit number

Defined in

chk_iie.c in prts40.src

Description

The *chk_iie* function checks whether a specified bit number of the IIE register is set. If the return value equals 1, the bit is set in IIE register. If the return value equals 0, the bit is not set in the IIE register.

Example

Check if ICFULL0 bit (bit 1) is set.

```
if (chk_iie(ICFULL0))   /* Check if ICFULL0 bit, bit1,is set */
       {
           .
           .
           .
       };
```

Related Functions

set_iie, reset_iie

Syntax

```
#include <intpt40.h>
int chk_iif_flag(int bit_no);
```

Parameters

bit_no — IIF register bit number

Defined in

chk_iif.c in prts40.src

Description

The *chk_iif_flag* function checks whether a specified bit number of the IIF register is set. If the return value equals 1, the bit is set in the IIF register. If the return value equals 0, the bit is not set in the IIF register.

Example

Check if the DMA0 flag bit (bit 25) is set.

```
if (chk_iif_flag(DMA0_FLAG))  /* Check if DMA0 flag bit,
    {                            bit25, is set        */
        .
        .
        .
    };
```

Related Functions

reset_iif_flag, set_iif_flag

Syntax

#include <dma40.h>
int chk_pri_dma(int ch_no);

Parameters

ch_no — DMA channel number (0–5)

Defined in

chk_pri.c in prts40.src or in dma40.h (if INLINE option is used)

Description

The *chk_pri_dma* function checks whether a specified DMA primary channel is being used. If the return value equals 1, the DMA primary channel is in use. If the return value equals 0, the DMA primary channel is free.

Example

Check if the primary channel of DMA #3 is busy. You need to initialize DMA_CTRL, src, src_indx, and size.

```
DMA_PRI_REG tab;
    .
    .
    .
set_pri_auto(&tab, DMA_CTRL, src, src_indx, size, 0);

while(chk_pri_dma(4));   /* Check if primary channel of
                            DMA # 3 is busy                 */
dma_prigo(4, &tab);      /* Start DMA #4 primary transfer   */
```

Related Functions

chk_aux_dma, chk_dma, dma_prigo, set_pri_auto

Syntax

#include <compt40.h>
COMPORT_REG *COMPORT_ADDR(int ch_no);

Parameters

ch_no — Communication port channel number (0–5)

Defined in

compt40.h (as a macro)

Description

The *COMPORT_ADDR* macro sets up the communication port register memory location.

Example

Set up a pointer to communication port channel 3.

```
COMPORT_REG *cp_ptr = COMPORT_ADDR(3);
```

Related Macros

DMA_ADDR, TIMER_ADDR

Syntax

```
#include <compt40.h>
long *COMPORT_IN_ADDR(int ch_no);
```

Parameters

ch_no — Communication port channel number (0–5)

Defined in

compt40.h (as a macro)

Description

The *COMPORT_IN_ADDR* macro sets up a communication port-input pointer.

Example

Set up the cp_ptr pointer to point to communication port channel 5.

```
long *cp_ptr = COMPORT_IN_ADDR(5);
```

Related Macros

COMPORT_OUT_ADDR

COMPORT_OUT_ADDR *Sets Up Communication Port Output Register Memory Location*

Syntax

```
#include <compt40.h>
long *COMPORT_OUT_ADDR(int ch_no);
```

Parameters

ch_no — Communication port channel number (0–5)

Defined in

compt40.h (as a macro)

Description

The *COMPORT_OUT_ADDR* macro sets up the communication port output-register memory location.

Example

Set up the cp_ptr point to communication port channel 0 output-register memory location (for example, 0x100042).

```
long *cp_ptr = CP_OUT_ADDR(0);
```

Related Macros

COMPORT_IN_ADDR

Syntax

#include <timer40.h>
void count_down(int t, unsigned long x);

Parameters

t	—	Timer channel number (0, 1)
x	—	Number of cycles for timer period register

Defined in

cnt_down.c in prts40.src or in timer40.h (if INLINE option is used)

Description

The *count_down* function starts the timer with the period register equal to x cycles. *t* defines which timer is used. This function can be used as an *alarm-clock*-type application. The timer interrupt flag is set after the counter reaches the period register value. Your interrupt service routine is executed if the GIE bit, IIE register, and interrupt vectors are set up properly.

Example

Start timer 1 with period register equal to 1000 cycles

```
count_down(1, 10000);   /* Start timer 1 with period = 10000 */
time_delay(100);        /* Delay CPU for 100 cycles          */
i = count_left(1);      /* Check the remain cycles time      */
```

Related Functions

alarm

Syntax	#include <timer40.h>
	int count_left(int t);

Parameters	t — Timer channel number (0, 1)
Defined in	cnt_left.c in prts40.src or in time40.h (if INLINE option is used)
Description	The *count_left* function returns the remaining cycle time for the timer to set the timer interrupt flag (or reach the period time). It returns the difference between the *timer* and *register-counter register* of the timer without changing the status of the timer. *t* defines whether timer 0 or timer 1 is used.
Example	See the count_down function example.
Related Functions	time_left, time_start

Syntax

#include <compt40.h>
void cp_in_halt(int ch_no);

Parameters

ch_no — Communication port channel number (0–5)

Defined in

cpinhlt.c in prts40.src or compt40.h (if INLINE option is used)

Description

The *cp_in_halt* function halts a specified communication port input channel.

Example

Halt communication port 3 input channel.

```
cp_in_halt(3);   /* Halt comm port 3 input channel    */
```

Related Functions

cp_in_release, cp_out_halt

cp_in_level *Returns the Input Buffer Level of a Specified Communication Port Channel*

Syntax

#include <compt40.h>
int cp_in_level(int ch_no);

Parameters

ch_no — Communication port channel number (0–5)

Defined in

cpinbuf.c in prts40.src or in compt40.h (if INLINE option is used)

Description

The *cp_in_level* function returns the input-buffer level (number of words) of a specified communication port channel.

Example

Return the input buffer level of communication port 2.

```
while (!cp_in_level(2));   /* Wait for input data from
                              comm port 2             */
```

Related Functions

cp_in_halt, cp_in_release, cp_out_level

Syntax

#include <compt40.h>
void cp_in_release(int ch_no);

Parameters

ch_no — Communication port channel number (0–5)

Defined in

cpingo.c in prts40.src or in compt40.h (if INLINE option is used)

Description

The *cp_in_release* function starts a specified communication port input channel.

Example

Start communication port 1 input channel.

```
cp_in_release(1);   /* Unhalt comm port 1 input channel  */
```

Related Functions

cp_in_halt, cp_in_level, cp_out_release

Syntax	#include <compt40.h>
	void cp_out_halt(int ch_no);

Parameters	ch_no — Communication port channel number (0–5)
Defined in	cpouthlt.c in prts40.src or in compt40.h (if INLINE option is used)
Description	The *cp_out_halt* function halts a specified communication port output channel.
Example	Halt communication port 0 output channel.

```
cp_in_halt(0);   /* Halt comm port 0 output channel   */
```

Related Functions	cp_in_level, cp_in_release, cp_out_halt

Syntax

#include <compt40.h>
int cp_out_level(int ch_no);

Parameters

ch_no — Communication port channel number (0–5)

Defined in

cpoutbuf.c in prts40.src or in compt40.h (if INLINE option is used)

Description

The *cp_out_level* function returns the output-buffer level (number of words) of a specified communication port channel.

Example

Return output buffer level for communication port 4.

```
while (cp_out_level(4));   /* Wait for comm port 4
                              output buffer to be empty      */
```

Related Functions

cp_in_halt, cp_in_level, cp_in_release

Syntax

```
#include <compt40.h>
void cp_out_release(int ch_no);
```

Parameters

ch_no — Communication port channel number (0–5)

Defined in

cpoutgo.c in prts40.src or in compt40.h (if INLINE option is used)

Description

The *cp_out_release* function starts a specified communication port output channel.

Example

Start communication port 0 output channel.

```
cp_out_release(0);  /* Unhalt comm port 0 output channel */
```

Related Functions

cp_in_release, cp_out_halt, cp_out_level

Syntax

#include <intpt40.h>
void CPU_IDLE(void);

Parameters

None

Defined in

intpt40.h (as a macro)

Description

The *CPU_IDLE* macro puts the 'C4x CPU in idle state to wait for the interrupt. This macro is used in the time_delay() and sleep() functions when the CPU is waiting for timer 1 interrupt to be waked up.

Example

Set the CPU to idle state to wait for the DMA1 interrupt to occur.

```
set_iie(DMA1);        /* Enable DMA1 interrupt   */
INT_ENABLE();         /* Enable GIE bit in ST    */
CPU_IDLE();           /* Set CPU to idle status  */
```

Related Functions/ Macros

install_int_vector, INT_ENABLE, set_iie, set_ivtp

Syntax #include <intpt40.h>
void deinstall_int_vector(int N);

Parameters N — The number of the interrupt vector location

Defined in set_vect.c in prts40.src

Description The *deinstall_int_vector* function is a counterpart of the install_int_vector function. It restores the data from int_vect_buf[N] to the memory location pointed to by the IVTP register plus the displacement N. Therefore, the old interrupt vector, which is modified by the install_int_vector function, can be restored.

Example The example below will restore the data in memory location 0x2FFE02 from int_vect_buf[2].

```
set_ivtp((void *)0x2ffe00);       /* set the IVTP = 0x2FFE00 */
install_int_vector((void *)c_int02, 2);
               .
               .
               .
deinstall_int_vector(2);
reset_ivtp();                   /* set the IVTP back to
                                  old location         */
```

Related Functions install_int_vector, reset_ivtp

Syntax

#include <intpt40.h>
int die_value(void);

Parameters

None

Defined in

val_die.c in prts40.src

Description

The *die_value* function returns the current data value of the 'C4x CPU register, DIE (DMA Interrupt Enable).

Example

The example below shows how to get the DIE register value in C program.

```
i = die_value();     /* Reads the DIE register value   */
```

Related Functions

dma_int_move

Syntax

```
#include <dma40.h>
DMA_REG *DMA_ADDR(int ch_no);
```

Parameters ch_no — DMA channel number (0–5)

Defined in dma40.h (as a macro)

Description The *DMA_ADDR* macro sets up the DMA register memory location.

Example Set up dma_ptr pointer to point DMA channel 0.

```
DMA_REG *dma_ptr = DMA_ADDR(0);
```

Related Macros COMPORT_ADDR, TIMER_ADDR

Syntax

```
#include <dma40.h>
void dma_auto_go(int ch_no, long ctrl, void *link_tab);
```

Parameters

ch_no	—	DMA channel number (0–5)
ctrl	—	DMA autoinitialization control word
*link_tab	—	DMA autoinitialization table linker pointer

Defined in

dmautogo.c in prts40.src or dma40.h (if INLINE option is used)

Description

The *dma_auto_go* function starts a specified DMA channel in unified-mode autoinitialization. The *link_tab pointer is loaded to the DMA link register first, and then the DMA control word is loaded into the DMA global control register to start the autoinitialization. The set_dma_auto function sets up the DMA autoinitialization link table.

The DMA channel function is overridden if the DMA is busy.

Example

Start DMA #1 autoinitialization. You must define DMA_CTRL, src, idxs, size, dest, idxd, and DMA_AUTO values.

```
DMA_REG   tab;
set_dma_auto(&tab, DMA_CTRL, src, idxs, size, dest, idxd, 0);
while(chk_dma(1));  /* Check if the DMA # 1 is busy        */
dma_auto_go(1, DMA_AUTO, &tab);  /* Start DMA #1 autoinit   */
```

Related Functions

chk_dma, dma_auxgo, dma_prigo, set_dma_auto

Syntax

```
#include <dma40.h>
void dma_auxgo(int ch_no, DMA_AUX_REG *register);
```

Parameters

ch_no	—	DMA channel number
*register	—	DMA auxiliary-channel register structure pointer

Defined in

dma_aux.c in prts40.src

Description

The *dma_auxgo* function starts a DMA split-mode auxiliary-channel data transfer with a specified DMA channel. The structure DMA_AUX_REG is defined in the header file. It contains the DMA auxiliary-channel register values for the DMA auxiliary-channel transfer function setup. The set_aux_auto function sets up the DMA auxiliary-channel register structure pointer.

The DMA channel function is overridden if the DMA is busy.

Example

See the chk_aux_dma function example.

Related Functions

chk_aux_dma, dma_auto_go, dma_prigo, set_aux_auto

Syntax

#include <dma40.h>
void DMA_AUX_HALT(int ch_no);

Parameters

ch_no — DMA channel number (0–5)

Defined in

dma40.h (as a macro)

Description

The *DMA_AUX_HALT* macro halts the specified DMA auxiliary-channel function at the first available read or write boundary (by setting the aux_start field of the control register to binary 01).

Example

Halt DMA auxiliary channel 4.

```
DMA_AUX_HALT(4); /* halt DMA auxiliary channel 4 with 01
                    in aux_start field of control register  */
```

Related Macros

DMA_AUX_HALT_B, DMA_AUX_RESTART

DMA_AUX_HALT_B *Halts the Specified DMA Auxiliary-Channel Function at Read/Write Boundary*

Syntax

#include <dma40.h>
void DMA_AUX_HALT_B(int ch_no);

Parameters

ch_no — DMA channel number

Defined in

dma40.h (as a macro)

Description

The *DMA_AUX_HALT_B* macro halts the specified DMA auxiliary-channel function at the read/write boundary (by setting the aux_start field of the control register to binary 10).

Example

Halt DMA auxiliary channel 2 with 10 in the aux_start field of the control register.

```
DMA_AUX_HALT_B(2); /* halt DMA auxiliary channel 2 with 10
                      in aux_start field of control register */
```

Related Macros

DMA_AUX_HALT, DMA_AUX_RESET, DMA_AUX_RESTART

Syntax

```
#include <dma40.h>
void DMA_AUX_RESET(int ch_no);
```

Parameters

ch_no — DMA channel number (0–5)

Defined in

dma40.h (as a macro)

Description

The *DMA_AUX_RESET* macro resets the specified DMA auxiliary-channel function (by setting the aux_start field of the control register to binary 00).

Example

Reset DMA auxiliary channel 0.

```
DMA_AUX_RESET(0);   /* reset DMA auxiliary channel 0 */
```

Related Macros

DMA_AUX_HALT, DMA_AUX_HALT_B, DMA_AUX_RESTART

DMA_AUX_RESTART *Restarts the Specified DMA Auxiliary-Channel Function*

Syntax

```
#include <dma40.h>
void DMA_AUX_RESTART(int ch_no);
```

Parameters

ch_no — DMA channel number (0–5)

Defined in

dma40.h (as a macro)

Description

The *DMA_AUX_RESTART* macro restarts the specified DMA auxiliary-channel function (by setting the aux_start field of the control register to binary 11).

Example

Restart DMA auxiliary channel 5.

```
DMA_AUX_RESTART(5); /* restart DMA auxiliary channel 5   */
```

Related Macros

DMA_AUX_HALT, DMA_AUX_HALT_B, DMA_AUX_RESET

Syntax

#include <dma40.h>
void dma_cmplx(int ch_no, void *src, void *dest, size_t FFT_size
int priority);

Parameters

ch_no	—	DMA channel number (0–5)
*src	—	Data source pointer
*dest	—	Data destination pointer
FFT_size	—	The size of the FFT
priority	—	The priority scheme between CPU and DMA

Defined in

dma_fft.c in prts40.src

Description

The *dma_cmplx* function transfers an array of complex-numbered data (real/image pairs in contiguous memory locations) with bit-reversed addressing from *src to *dest via a specified DMA channel. After the DMA transfer is completed, the DMA interrupt flag is set. The DMA interrupt can be served if the GIE bit, IIE register, DMA interrupt vector, and DMA interrupt service are set properly. You can configure the CPU/DMA priority scheme as follows:

If priority equals
0 — CPU has higher priority,
1 — CPU/DMA have rotated priority scheme,
3 — DMA has higher priority.

This function is useful for complex FFT data input/output. The destination address needs to be on a specific boundary for bit-reversed addressing.

The DMA channel function is overridden if the DMA is busy.

Example

The function below sets up the dma_complx function for 1024-point complex-data bit-reversed transfer from src to dest using DMA channel 2 with CPU/DMA rotated priority scheme. Note that the destination pointer is updated with bit-reversed order. Therefore, the base address of destination should be aligned on a specific boundary. Refer to the *TMS320C4x User's Guide* for details on base-address requirements of bit-reversed addressed buffers.

```
while (chk_dma (2));
dma_cmplx(2, src, dest, 1024, 1);
```

Related Functions

chk_dma

Syntax

```
#include <dma40.h>
void dma_extrig(int ex_int, int ch_no, DMA_REG *register);
```

Parameters

ex_int	—	External interrupt signals (IIOF0–3)
ch_no	—	DMA channel number (0–5)
*register	—	DMA register structure pointer

Defined in

dma_trig.c in prts40.src

Description

This *dma_extrig* function sets up a DMA data transfer with a specified channel to be triggered by a specified external-interrupt signal. The structure DMA_REG is defined in the header file. It contains the DMA register values for DMA transfer function setup. The set_dma_auto function also sets up the DMA-register structure pointer.

The DMA channel function is overridden if the DMA is busy.

Example

Start DMA #3 to wait for IIOF1 interrupt signal.

```
DMA_REG   tab;
set_dma_auto(&tab, DMA_CTRL, src, idxs, size, dest, idxd, 0);

while(chk_dma(3));      /* Check if the DMA # 3 is busy      */

dma_extrig(1, 3, &tab); /* Start DMA #3 to wait for
                            IIOF1 interrupt signal            */
```

Related Functions

chk_dma, set_dma_auto

Syntax

```
#include <dma40.h>
void dma_go(int ch_no, DMA_REG *register);
```

Parameters

ch_no	—	DMA channel number (0–5)
*register	—	DMA register structure pointer

Defined in

dma_go.c in prts40.src

Description

The *dma_go* function starts a specified DMA channel to perform a specified DMA transfer function. The structure DMA_REG is defined in the header file. It contains the DMA register values for the DMA transfer function setup. The set_dma_auto function sets up the DMA-register structure pointer.

The DMA channel function is overridden if the DMA is busy.

Example

See the chk_dma function example.

Related Functions

chk_dma, set_dma_auto

Syntax	#include <dma40.h>
	void DMA_HALT(int ch_no);

Parameters	ch_no — DMA channel number (0–5)
Defined in	dma40.h (as a macro)
Description	The *DMA_HALT* macro halts the specified DMA unified/primary channel function at the first available read or write boundary (by setting the start field of the control register to binary 01).
Example	Halt DMA primary channel 3 with 01 in the start field of the control register.

```
DMA_HALT(3);  /* halt DMA primary channel 3 with 01
                 in start field of control register          */
```

Related Macros DMA_HALT_B, DMA_RESET, DMA_RESTART

Syntax

#include <dma40.h>
void DMA_HALT_B(int ch_no);

Parameters

ch_no — DMA channel number (0–5)

Defined in

dma40.h (as a macro)

Description

The *DMA_HALT_B* macro halts the specified DMA unified/primary-channel function at a read/write boundary (by setting the start field of the control register to binary 10).

Example

Halt DMA primary channel 1 with 10 in the start field of the control register.

```
DMA_HALT_B(1);    /* halt DMA primary channel 1 with 10 (base 2)
                     in start field of control register   */
```

Related Macros

DMA_HALT, DMA_RESET, DMA_RESTART

Syntax

```
#include <dma40.h>
void dma_int_move(int ex_int, int ch_no, void *src,
void *dest, size_t length);
```

Parameters

ex_int	—	External interrupt signals (IIOF0–3)
ch_no	—	DMA channel number (0–5)
*src	—	Data source pointer
*dest	—	Data destination pointer
length	—	Number of data to be transferred

Defined in

dmaintmv.c in prts40.src

Description

The *dma_int_move* function sets up a DMA data transfer from *src to *dest with a specified DMA channel to be triggered by a specified external-interrupt signal. After the DMA transfer is completed, the DMA interrupt flag is set. The DMA interrupt can be served if the GIE bit, IIE register, DMA interrupt vector, and DMA interrupt service are set properly.

The DMA channel function is overridden if the DMA is busy.

Example

Set up the 64-point DMA channel 1 data transfer from source to destination to be triggered by external interrupt signal IIOF2.

```
set_iiof(2,1);                      /* configure IIOF2 as edge */
                                    /* trigger interrupt pin   */
dma_int_move(2, 1, src, dest, 64); /* setup DMA 1 to wait for
                                       IIOF2 signal            */
```

Related Functions

set_iiof

Syntax

```
#include <dma40.h>
void dma_move(int ch_no, void *src, void *dest, size_t length);
```

Parameters

ch_no	—	DMA channel number
*src	—	Data source pointer
*dest	—	Data destination pointer
length	—	Number of data to be transferred

Defined in

dma_move.c in prts40.src

Description

The *dma_move* function transfers a block of data array with specified size and length from *src to *dest by a specified DMA channel. After the DMA transfer is completed, the DMA interrupt flag is set. The DMA interrupt can be served if the GIE bit, IIE register, DMA interrupt vector, and DMA interrupt service are set properly.

The DMA channel function is overridden if the DMA is busy.

Example

See chk_dma_flag function example.

Related Functions

chk_dma

Syntax

```
#include <dma40.h>
void dma_prigo(int ch_no, DMA_PRI_REG *register);
```

Parameters

ch_no	—	DMA channel number
*register	—	DMA primary channel register structure pointer

Defined in

dma_pri.c in prts40.src

Description

The *dma_prigo* function starts a DMA split-mode primary-channel data transfer with a specified DMA channel. The structure DMA_PRI_REG is defined in the header file. It contains the DMA primary-channel register values for the DMA primary-channel transfer function setup. The set_pri_auto function can be used to set up the DMA primary-channel register structure pointer.

The DMA channel function is overridden if the DMAs (either primary or aux channel) are busy.

Example

See chk_pri_dma function example.

Related Functions

chk_pri_dma, set_pri_auto

Syntax

#include <dma40.h>
void DMA_RESET(int ch_no);

Parameters

ch_no — DMA channel number (0–5)

Defined in

dma40.h (as a macro)

Description

The *DMA_RESET* macro resets the specified DMA unified/primary-channel function (by setting the start field of control register to 00_2).

Example

Reset DMA unified/primary channel 5.

```
DMA_RESET(5); /* reset DMA unified/primary channel 5  */
```

Related Macros

DMA_HALT, DMA_HALT_B, DMA_RESTART

DMA_RESTART *Restarts the Specified DMA Unified/Primary Channel*

Syntax

```
#include <dma40.h>
void DMA_RESTART(int ch_no);
```

Parameters

ch_no — DMA channel number

Defined in

dma40.h (as a macro)

Description

The *DMA_RESTART* macro restarts the specified DMA unified/primary-channel function (by setting the start field of the control register to 11_2).

Example

Restart DMA unified/primary channel 4.

```
DMA_RESTART(4);  /* restart DMA unified/primary channel 4    */
```

Related Macros

DMA_HALT, DMA_HALT_B, DMA_RESET

Syntax

```
#include <dma40.h>
void dma_sync_set(int ch_no, int bit_value, int r_w);
```

Parameters

ch_no	—	DMA channel number
bit_value	—	DMA synchronization control bit value for DIE register
r_w	—	Read/write synchronization

Defined in

die_set.asm in prts40.src

Description

The *dma_sync_set* function sets up a DIE register value for a specified DMA channel. The bit_value will be loaded into the specified DMA channel (ch_no) read/write-synchronization field (r_w).

If r_w equals

0	—	Read synchronization
1	—	Write synchronization

Example

Set up the DIE register for DMA channel 2 source or read synchronization with the IIOF0 interrupt signal.

```
dma_sync_set(2, 2, 0);
```

Related Functions

dma_extrig, dma_int_move

Syntax

#include <timer40.h>
float elapse(void);

Parameters

None

Defined in

elapse.c in prts40.src or timer40.h (if INLINE option is used)

Description

The *elapse* function returns the approximate elapsed time, in seconds, since the last time_go function call.

Note:

The speed of the processor is target-specific. The default value of CLOCK_PER_SEC is set to 25000000.0 by prts40.src, corresponding to a 50-MHz-input-clock 'C4x. For a different processor speed, you must initialize in your program the global variable CLOCK_PER_SEC (defined in timer40.c) to the desired value. CLOCK_PER_SEC must be set to half the number of input system clocks per second. For example, for a 40-MHz input clock 'C4x:

If X2/XCLKIN = 40 MHz, then H1 = X2/XCLKIN ÷ 2 = 20 MHz
so CLOCK_PER_SEC = H1 = 20000000.0

The maximum time that the elapse() function can handle is about 2^{64} (1.85 * 10^19/CLOCK_PER_SEC) seconds.

Example

Set up the high-level-timer function to do the benchmark a code segment for a system with a 40-MHz input clock speed.

```
#include  <timer40.h>
extern    float  CLOCK_PER_SEC=20000000.0;
float     time1, time2, time3;

time_run();         /* start timer 0 for benchmark          */
   :   :      :
  *** program code # 1 ***
   :   :      :
time1 = elapse();   /* take the first benchmark             */
   :   :      :
  *** program code # 2 ***
   :   :      :
time2 = elapse();   /* take the second benchmark            */
   :   :      :
  *** program code # 3 ***
   :   :      :
time3 = time_end(); /* take third benchmark and stop timer  */
```

Related Functions

time_end, time_run

Syntax

```
#include <intpt40.h>
void GET_DIE(void);
```

Parameters

None

Defined in

intpt40.h (as a macro)

Description

The *GET_DIE* macro loads the value of the DIE register into R0. It is used in the die_value() function.

Example

Use of the GET_DIE macro in the die_value function in val_die.c

```
#include <intpt40.h>

int die_value()
   {
      GET_DIE();
   }
```

Related Functions

die_value

GET_IIE *Loads IIE Register Value to R0 Register*

Syntax

#include <intpt40.h>
void GET_IIE(void);

Parameters

None

Defined in

intpt40.h (as a macro)

Description

The *GET_IIE* macro loads the value of the IIE register into R0. It is used in the iie_value() function.

Example

Use of the GET_IIE macro in the iie_value function in val_iie.c

```
#include <intpt40.h>

int iie_value()
    {
        GET_IIE();
    }
```

Related Functions

iie_value

Syntax

#include <intpt40.h>
void GET_IIF(void);

Parameters

None

Defined in

intpt40.h (as a macro)

Description

The *GET_IIF* macro loads the value of the IIF register into R0. It is used in the iif_value() function.

Example

Use of the GET_IIF macro in the iif_value function in val_iif.c

```
#include <intpt40.h>

int iif_value()
   {
      GET_IIF();
   }
```

Related Functions

iif_value

GET_IVTP *Loads IVTP Register Value to R0 Register*

Syntax

#include <intpt40.h>
void GET_IVTP(void);

Parameters

None

Defined in

intpt40.h (as a macro)

Description

The *GET_IVTP* macro loads the value of the IVTP register into R0. It is used in the ivtp_value() function.

Example

Use of the GET_IVTP macro in the ivtp_value function in val_ivtp.c

```
#include <intpt40.h>

int ivtp_value()
    {
       GET_IVTP();
    }
```

Related Functions

ivtp_value

Syntax

#include <intpt40.h>
void GET_ST(void);

Parameters

None

Defined in

intpt40.h (as a macro)

Description

The *GET_ST* macro loads the value of the ST register into R0. It is used in the st_value() function.

Example

Use of the GET_ST macro in the st_value function in val_st.c

```
#include <intpt40.h>

int ivtp_value()
   {
      GET_ST();
   }
```

Related Functions

st_value

Syntax

#include <intpt40.h>
void GET_TVTP(void);

Parameters

None

Defined in

intpt40.h (as a macro)

Description

The *GET_TVTP* macro loads the value of the TVTP register into R0. It is used in the tvtp_value() function.

Example

Use of the GET_TVTP macro in the tvtp_value function in val_tvtp.c

```
#include <intpt40.h>

int tvtp_value()
   {
      GET_TVTP();
   }
```

Related Functions

tvtp_value

Syntax

```
#include <intpt40.h>
int iie_value(void);
```

Parameters

None

Defined in

val_iie.c in prts40.src

Description

The *iie_value* function returns the current data value of the 'C4x CPU register, IIE (Internal Interrupt Enable).

Example

Read the IIE register value from C program.

```
i = iie_value();     /* Reads the IIE register value   */
```

Related Functions/ Macros

GET_IIE, iif_value

Syntax

#include <intpt40.h>
int iif_value(void);

Parameters

None

Defined in

val_iif.c in prts40.src

Description

The *iif_value* function returns the current data value of the 'C4x CPU register, IIF (IIOF and Internal Interrupt Flag).

Example

Read the IIF register value from C program.

```
i = iif_value();     /* Reads the IIF register value   */
```

Related Functions/ Macros

GET_IIF, iie_value

Syntax
#include <intpt40.h>
int iiof_in(int pin_no);

Parameters
pin_no — IIOF pin number (0–3)

Defined in
iiof_in.c in prts40.src

Description
The *iiof_in* function sets the IIOF as a general-purpose input pin and reads the value of the IIOF pin. *pin_no* defines whether IIOF0, IIOF1, IIOF2, or IIOF3 is read.

Example
Read the status of the IIOF3 pin.

```
in = iiof_in(3);    /* read in the IIOF3 pin status  */
```

Related Functions
iiof_out

Syntax

```
#include <intpt40.h>
void iiof_out(int pin_no, int flag);
```

Parameters

pin_no	—	IIOF pin number (0–3)
flag	—	Output signal value of the IIOF pin

Defined in

iiof_out.c in prts40.src

Description

The *iiof_out* function sets the IIOF as a general-purpose output pin and outputs the value of *flag* to the IIOF pin. *pin_no* defines whether IIOF0, IIOF1, IIOF2, or IIOF3 is used.

Example

Set the IIOF2 pin high.

```
iiof_out(2, 1);     /* set the IIOF2 pin high  */
```

Related Functions

iiof_in

Syntax

#include <compt40.h>
size_t in_msg(int ch_no, void *message, int step);

Parameters

ch_no	—	Communication port channel number (source)
*message	—	Data array pointer (destination)
step	—	Data array pointer increment step size

Defined in

in_msg.c in prts40.src

Description

The *in_msg* function reads data from a specified communication port channel, ch_no, to a word array that is pointed to by *message. The pointer *message increment step size is defined in parameter *step*. The function returns the size of the array that is received.

Example

Read in the data from communication port number 4 and put the data into column 1 of a 5 × 5 matrix.

```
int mat[5][5], data_size;       /* declare the matrix         */

data_size = in_msg(4, mat, 5); /* read data from comm port 4
                                  to the 1st column of matrix*/
```

Related Functions

out_msg

Syntax

#include <compt40.h>
size_t in_msg16(int ch_no, void *halfword_array);

Parameters

ch_no	—	Communication port channel number (source)
*halfword_array	—	Halfword-wide array pointer (destination)

Defined in

in_msg16.c in prts40.src

Description

The *in_msg16* function reads data from a specified communication port channel and unpacks the data to a 16-bit data array. The function returns the size of the unpacked 16-bit data array that is received.

Example

Read in the data from communication port number 3 and unpack the data into 16-bit-wide data array dat16.

```
data_size = in_msg16(3, dat16);/* read data from comm port 3
                                  to 16-bit wide data array */
```

Related Functions

out_msg16, unpack_halfword

Syntax

#include <compt40.h>
size_t in_msg8(int ch_no, void *byte_array);

Parameters

ch_no	— Communication port channel number (source)
*byte_array	— Byte-wide array pointer (destination)

Defined in

in_msg8.c in prts40.src

Description

The *in_msg8* function reads data from a specified communication port channel and unpacks the data to a byte array. The function returns the size of the unpacked-byte array that is received.

Example

Read in the data from communication port number 2 and unpack the data into byte-wide data array dat8.

```
data_size = in_msg8(2, dat8); /* read data from comm port 2
                                 to byte wide data array     */
```

Related Functions

out_msg8, unpack_byte

Syntax

#include <timer40.h>
int in_timer(int t);

Parameters

t — Timer channel number (0,1)

Defined in

tim_in.c in prts40.src or in timer40.h (if INLINE option is used)

Description

The *in_timer* function reads the value of the TCLK pin and configures the timer as a general-purpose input pin. *t* defines whether TCLK0 or TCLK1 is read.

Example

Read the status of the TCLK1 pin.

```
in = in_timer(1);       /* read in the TCLK1 pin status   */
```

Related Functions

out_timer

Syntax

#include <compt40.h>
long in_word(int ch_no);

Parameters

ch_no — Communication port channel number (source)

Defined in

in_word.c in prts40.src or in compt40.h (if INLINE option is used)

Description

The *in_word* function reads a single word from a specified communication port channel.

Example

Read in one word from communication port number 1.

```
data = in_word(1);      /* read one word from comm port 1*/
```

Related Functions

out_word

Syntax

#include <intpt40.h>
void install_int_vector(void *isr, int N);

Parameters

*isr	—	Interrupt service routine address
N	—	The number of the interrupt vector location

Defined in

set_vect.asm in prts40.src

Description

The *install_int_vector* function sets up the interrupt vector (interrupt-service routine address) into the section where the IVTP register points to plus the displacement N. The old value in that location is saved in a corresponding global array, int_vect_buf[N].

Example

If the .vector uninitialized section is allocated at 0x2FFE00 but must be on a 512-word boundary, the example below sets the IVTP to point to 0x2FFE00 and put the c_int02 timer 0 interrupt-service-routine address in memory location 0x2FFE02. Therefore, when timer 0 interrupt occurs, the processor branches to the c_int02 interrupt-service routine if the GIE bit of the ST status register and the corresponding IIE bit are preset.

```
set_ivtp((void *)0x2ffe00);    /* set the IVTP = 0x2FFE00     */

install_int_vector((void *)c_int02, 2);
```

Related Functions

deinstall_int_vector, set_ivtp

Syntax

```
#include <intpt40.h>
void INT_DISABLE(void);
```

Parameters

None

Defined in

intpt40.h (as a macro)

Description

The *INT_DISABLE* macro resets bit 14 (GIE) of the 'C4x status register (ST) globally disabling 'C4x interrupts.

Example

Globally disable 'C4x interrupts.

```
INT_DISABLE();      /* Reset the GIE bit       */
```

Related Macros

INT_ENABLE

Syntax

```
#include <intpt40.h>
void INT_ENABLE(void);
```

Parameters

None

Defined in

intpt40.h (as a macro)

Description

The *INT_ENABLE* macro sets bit 14 (GIE) of the 'C4x status register (ST), globally enabling the 'C4x interrupts.

Example

Globally enable 'C4x interrupts.

```
INT_ENABLE();    /* Set the GIE bit      */
```

Related Macros

INT_DISABLE

Syntax

#include <intpt40.h>
int ivtp_value(void);

Parameters

None

Defined in

ivtp_value() in prts40.src

Description

The *ivtp_value* function returns the current data value of the 'C4x CPU register, IVTP (Interrupt Vector Table Pointer).

Example

Read the IVTP register from C program.

```
i = ivtp_value();       /* Reads the IVTP register value */
```

Related Functions/ Macros

GET_IVTP, tvtp_value

Syntax

```
#include <intpt40.h>
void load_die(unsigned die_data);
```

Parameters

die_data — the data to be loaded into the DIE register

Defined in

load_die.c in prts40.src

Description

The *load_die* function loads data die_data into the DIE (DMA Interrupt Enable) register.

Example

Load 0x10 into the DIE register.

```
load_die(0x10);      /* Load data, 10h, into DIE register     */
```

Related Functions

load_iie, load_iif

Syntax

```
#include <intpt40.h>
void load_iie(unsigned iie_value);
```

Parameters

iie_value — the data to be loaded into the IIE register

Defined in

load_iie.c in prts40.src

Description

The *load_iie* function loads data iie_value into the IIE (Internal Interrupt Enable) register.

Example

Load 0x800 into the IIE register.

```
load_iie(0x800);    /* Load data, 800h, into IIE register   */
```

Related Functions

load_die, load_iif, reset_iie, set_iie

Syntax

```
#include <intpt40.h>
void load_iif(unsigned iif_value);
```

Parameters

iif_value — the data to be loaded into the IIF register

Defined in

load_iif.c in prts40.src

Description

The *load_iif* function loads data iif_value into the IIF (IIOF and Internal Interrupt Flag) register.

Example

Load 0x8000000 into the IIF register.

```
load_iif(0x8000000); /* Load data, 8000000h, into IIF register */
```

Related Functions

load_die, load_iie, reset_iif_flag, set_iif_flag

Syntax

#include <mulpro40.h>
int lock(int *semaphore);

Parameters

*semaphore	—	semaphore flag pointer

Defined in

lock.asm in prts40.src

Description

The *lock* function implements the P(s) function of the shared-memory-interlock operation. It returns the value of the shared-memory semaphore, *semaphore, and sets the shared-memory semaphore to one.

Example

Use the lock function to implement P(S).

```
while  (!lock(&S)); /* to gain control of the semaphore S   */
                   /* Shared memory processing begin       */
       .
       .
       .
                   /* Shared memory processing end         */
unlock(&S));       /* to release control of the semaphore S */
```

Related Functions

unlock

Syntax

```
#include <mulpro40.h>
int MY_ID(void);
```

Parameters None

Defined in mulpro40.h (as a macro)

Description The *MY_ID* macro reads the processor-identification number from the specified memory that is defined by #define of ID_ADDR. The default location of ID_ADDR is 0x2FFF00.

Note:

If location 0x2FFF00 conflicts with the memory use of your program, you can use the #define ID_ADDR preprocessor directive to change the processor-identification number location. **The rts40.lib should be rebuilt with mk30 for this change to take effect.**

Example Read in the processor-identification number from the predefined-memory location.

```
#include  <mulpro40.h>

id_number = MY_ID();    /* read in the processor id #    */
```

Related Functions None

Syntax

#include <compt40.h>
void out_msg(int ch_no, void *message, size_t message_size,
int step);

Parameters

ch_no	—	Communication port channel number (destination)
*message	—	Data array pointer (source)
message_size	—	Number of data to be sent
step	—	Data array pointer increment step size

Defined in

out_msg.c in prts40.src

Description

The *out_msg* function sends a word array that is pointed to by *message to a specified communication port channel, ch_no. The array pointer *message increment step size is defined in parameter *step*.

Example

Read the data from column 2 of a 5 × 5 matrix and send the data out from communication port number 2.

```
int mat[5][5];                  /* declare the matrix        */

out_msg(2, &mat[0][1], 5, 5);   /* read data from 2nd column of
                                   the matrix to comm port 2 */
```

Related Functions

in_msg

Syntax

#include <compt40.h>
void out_msg16(int ch_no, void *halfword_array, size_t array_size);

Parameters

ch_no	—	Communication port channel number (destination)
*halfword_array	—	Halfword-wide array pointer (source)
array_size	—	Number of halfword wide data to be sent

Defined in

outmsg16.c in prts40.src or in compt40.h (if INLINE option is used)

Description

The *out_msg16* function packs a 16-bit-wide array, *halfword_array, to a 32-bit-wide array and sends it to a specified communication port channel (ch_no). First, the size of the packed-data array is sent to the communication port, and then the data is sent. The data is packed from LSBs. If there is an extra 16-bit space in the last word, it is padded with zeros.

Example

Pack the fifteen 16-bit-wide data from 16-bit-wide data array dat16 to eight 32-bit data and send packed data out by communication port number 3.

```
out_msg16(3, dat16, 15);  /* pack 16-bit wide data from dat16
                             and send it to comm port 3      */
```

Related Functions

in_msg16, pack_halfword

Syntax

#include <compt40.h>
void out_msg8(int ch_no, void *byte_array, size_t array_size);

Parameters

ch_no	—	Communication port channel number (destination)
*byte_array	—	Byte-wide array pointer (source)
array_size	—	Number of byte wide data to be sent

Defined in

out_msg8.c in prts40.src or in compt40.h (if INLINE option is used)

Description

The *out_msg8* function packs a byte array, *byte_array, to 32-bit-wide array and sends it to a specified communication port channel (ch_no). First, the size of the packed-data array is sent to the communication port, and then the data is sent. The first byte is packed from LSBs. If there is extra byte space in the last word, it is padded with zeros.

Example

Pack thirteen 8-bit (byte-wide) data packets from data array dat8 into four 32-bit data packets and send them out by communication port 4.

```
out_msg8(4, dat8, 13);   /* pack byte wide data from dat8
                             and send out by comm port 4       */
```

Related Functions

in_msg8, pack_byte

Syntax

#include <timer40.h>
void out_timer(int t, int flag);

Parameters

t	—	Timer channel number (0,1)
flag	—	Output signal value of the TCLK pin

Defined in

out_timer() in prts40.src

Description

The *out_timer* function outputs the value of *flag* to the TCLK pin when the timer is configured as a general-purpose output pin. *t* defines whether timer 0 or timer 1 is used.

Example

Set the TCLK0 pin high.

```
out_timer(0, 1);    /* set the TCLK0 pin high     */
```

Related Functions

in_timer

Syntax

#include <compt40.h>
void out_word(long word_value, int ch_no);

Parameters

word_value	— Output word data (source)
ch_no	— Communication port channel number (destination)

Defined in

out_word.c in prts40.src or in compt40.h (if INLINE option is used)

Description

The *out_word* function sends a word, word_value, to a specified communication port channel.

Example

Write a one-word *value* to communication port 0.

```
out_word(value, 0);          /* write value to comm port 0 */
```

Related Functions

in_word

Syntax	#include <compt40.h> void pack_byte(void *message, void *pack_msg, size_t msg_size)
Parameters	*message — Input byte-wide data array pointer (source) *pack_msg — Output data FIFO pointer (destination) message_size — Number of byte-wide data to be sent
Defined in	pack8.asm in prts40.src
Description	The *pack_byte* function packs the byte-wide data and sends it to the full-word data FIFO (or communication port), *pack_msg. First, the size of the packed-data array is sent to the communication port, and then the data is sent. The data is packed from LSBs. If the input data is not exactly the size of a full-word data, zeros append the last word's MSBs. This function is designed mainly for the out_msg8 function. The *pack_byte* function can be modified for data packing easily.
Example	Refer to the source code of the out_msg8 function in the prts40.src file.
Related Functions	unpack_byte, out_msg8

Syntax

```
#include <compt40.h>
size_t pack_halfword(void *message,void *pack_msg,size_t msg_size)
```

Parameters

*message	—	Input 16-bit wide data array pointer (source)
*pack_msg	—	Output data FIFO pointer (destination)
message_size	—	Number of 16-bit wide data to be sent

Defined in

pack16.asm in prts40.src

Description

The *pack_halfword* function packs the 16-bit-wide data and sends it to the full-word data FIFO (or communication port), *pack_msg. First, the size of the packed-data array is sent to the communication port, and then the data is sent. The data is packed from LSBs. If the input data is not exactly the size of full-word data, zeros append the last word's MSBs. This function is mainly designed for the out_msg16 function. The *pack_halfword* function can be modified for data packing easily.

Example

Refer to the source code of the out_msg16 function in the prts40.src file.

Related Functions

out_msg16, unpack_halfword

Syntax

#include <compt40.h>
void receive_msg(int ch_no, void *message, int step);

Parameters

ch_no	—	Communication port channel number (source)
*message	—	Data array pointer (destination)
step	—	Data array pointer increment step size

Defined in

rec_msg.c in prts40.src

Description

The *receive_msg* function sets up a DMA to read data from a specified communication port channel, ch_no, to a word array that is pointed to by *message. The pointer *message increment step size is defined in the parameter *step*. It checks whether the DMA channel is busy before setting the DMA function. This function uses DMA autoinitialization and communication port input-ready synchronization to perform the data transfer. It is asynchronous to CPU operation after the setup. In other words, the CPU can be used in parallel with the data transfer. Shifting priority between CPU and DMA has been used.

Example

Read in the data from communication port number 3 and put the data into column 4 of a 5 × 5 matrix. The data transfer is asynchronous to the CPU operation.

```
int mat[5][5];                   /* declare the matrix          */
receive_msg(3,&mat[0][3],5);     /* read data from comm port 3
    :    :    :                     to the column 4 of matrix   */
   continue CPU operation
     :    :    :
while(chk_dma(3));               /* Check if the data received */
sum = mat[0][3] + mat[1][3];
     :    :    :
```

Related Functions

send_msg

Syntax

#include <intpt40.h>
void reset_iie(int bit_no);

Parameters

bit_no — IIE register bit number

Defined in

rst_iie.asm in prts40.src

Description

The *reset_iie* function resets a specified bit number of the IIE register.

Example

Disable the ICFULL4 interrupt.

```
reset_iie(ICFULL4); /* Disable ICFULL4 interrupt     */
```

Related Functions

load_iie, set_iie

Syntax	#include <intpt40.h>
	void reset_iif_flag(int bit_no);

Parameters	bit_no — IIF register bit number
Defined in	rst_iif.asm in prts40.src
Description	The *reset_iif_flag* function resets a specified bit number of the IIF register.
Example	Clear the DMA4 flag in IIF. `reset_iif_flag(DMA4_FLAG); /* Clear DMA4 flag in IIF  */`
Related Functions	load_iif, set_iif_flag

Syntax
#include <intpt40.h>
void reset_ivtp(void);

Parameters
None

Defined in
rst_ivtp.asm in prts40.src

Description
The *reset_ivtp* function is a counterpart of the set_ivtp function. It restores the data from the global variable ivtp_buf to the IVTP register.

Example
See deinstall_int_vect function example.

Related Functions
deinstall_int_vector, set_ivtp

Syntax

#include <intpt40.h>
void reset_tvtp(void);

Parameters

None

Defined in

rst_tvtp.asm in prts40.src

Description

The *reset_tvtp* function is a counterpart of the set_tvtp function. It restores the data from the global variable tvtp_buf to the TVTP register.

Example

Restore the TVTP register value.

```
reset_tvtp();    /* Restore TVTP from tvtp_buf     */
```

Related Functions

set_tvtp

Syntax

#include <compt40.h>
void send_msg(int ch_no, void *message, size_t message_size,
int step);

Parameters

ch_no	—	Communication port channel number (destination)
*message	—	Data array pointer (source)
message_size	—	Number of data to be sent
step	—	Data array pointer increment step size

Defined in

send_msg.c in prts40.src

Description

The *send_msg* function sets up a DMA to send a word array that is pointed to by *message to a specified communication port channel. The pointer *message increment step size is defined in the parameter *step*. This function uses DMA autoinitialization and communication port output-ready synchronization to perform the data transfer. Shifting priority between CPU and DMA has been used. It is asynchronous to CPU operation after the setup. In other words, the CPU can be used in parallel with the data transfer. However, the output data should not be modified by the CPU before the data is sent out; if it is, the wrong data could be sent.

Note:

The *send_msg* checks whether the DMA channel is busy before setting the DMA function.

Example

Read the data from column 3 of a 5 × 5 matrix and send the data out from communication port number 0. The data transfer is asynchronous to the CPU operation.

```
int mat[5][5];                    /* declare the matrix         */

send_msg(0, &mat[0][2], 5, 5); /* read data from 3rd column of
    :   :   :                        the matrix to comm port 0 */
                                     continue CPU operation
    :   :   :
while(chk_dma(0));                /* Check if the data sent     */

mat[0][2] = datin[0];
mat[1][2] = datin[1];
    :   :
```

Related Functions

receive_msg

Syntax	#include <dma40.h> void set_aux_auto(void *tab_addr, long ctrl, void *dest, int dest_idx, size_t length, void *next_tab)

Parameters

*tab_addr	—	Auxiliary-channel autoinitialization-table pointer
ctrl	—	DMA function control word
*dest	—	Data destination address for DMA destination register
dest_idx	—	Destination pointer step size for DMA destination index register
length	—	Data transfer length for auxiliary-counter register
*next_tab	—	Next auxiliary-channel autoinitialization table address for auxiliary link pointer register

Defined in dma_seta.c in prts40.src or in dma40.h (if INLINE option is used)

Description The *set_aux_auto* function sets up an autoinitialization table for DMA split-mode auxiliary-channel autoinitialization.

Example See the chk_aux_dma function example.

Related Functions chk_aux_dma, dma_auxgo

Syntax

```
#include <dma40.h>
void set_dma_auto(void *tab_addr, long ctrl, void *src,
int src_idx, size_t length, void *dest,
int dest_idx, void *next_tab)
```

Parameters

*tab_addr	—	Autoinitialization-table pointer
ctrl	—	DMA function control word
*src	—	Data source address for DMA source register
src_idx	—	Source pointer step size for DMA source index register
length	—	Data transfer length
*dest	—	Data destination address for DMA destination register
dest_idx	—	Destination pointer step size for DMA destination index register
*next_tab	—	Next autoinitialization table address for link pointer register

Defined in

dma_setv.c in prts40.src

Description

The *set_dma_auto* function sets up an autoinitialization table for DMA-unified-mode autoinitialization. If tab_addr points to the DMA channel control register, it can be used for setting the unified mode DMA register's structure.

Example

See the chk_dma function example.

Related Functions

chk_dma, dma_auto_go

Syntax

#include <intpt40.h>
void set_iie(int bit_no);

Parameters

bit_no — IIE register bit number

Defined in

set_iie.asm in prts40.src

Description

The *set_iie* function sets a specified bit number of the IIE register.

Example

Enable the ICRDY2 interrupt.

```
set_iie(ICRDY2);    /* Enable ICRDY2 interrupt    */
```

Related Functions

load_iie, reset_iie

Syntax

#include <intpt40.h>
void set_iif_flag(int bit_no);

Parameters

bit_no — IIF register bit number

Defined in

set_iif.asm in prts40.src

Description

The *set_iif* function sets a specified bit number of the IIF register.

Example

Set the DMA3 interrupt flag.

```
set_iif_flag(DMA3_FLAG);   /* Set DMA3 interrupt flag    */
```

Related Functions

load_iif, reset_iif_flag

Syntax

#include <intpt40.h>
void set_iiof(int pin_no, int iiof_value);

Parameters

pin_no	—	IIOF pin number
iiof_value	—	The data to be loaded into the iiof field

Defined in

set_iiof.asm in prts40.src

Description

The *set_iiof* function loads the data iiof_value into the specified IIOF field in the IIF register. *pin_no* defines whether IIOF0, IIOF1, IIOF2, or IIOF3 is used.

Example

Set the IIOF3 pin as a level-trigger interrupt pin and enable it.

```
set_iiof(3, 0xB);    /* Set the IIOF3 pin as level trigger
                        interrupt pin and enable it        */
```

Related Functions

dma_int_move

Syntax

#include <timer40.h>
void set_ivtp(void *isr);

Parameters

*isr — Interrupt vector table address

Defined in

set_ivtp.asm in prts40.src

Description

The *set_ivtp* function sets up the interrupt-vector-table pointer (IVTP) register to the address specified by the *isr parameter. If the *isr value is DEFAULT (=–1), the IVTP pointer is set to the .vector section. You can relocate the .vector section in the linker command file. In any case, the original IVTP value is preserved in the ivtp_buf global variable defined in the int_ivtp.c. The reset_ivtp function restores the ivtp value back.

Note that IVTP must be on the boundary of 512 words.

Example

Set the interrupt vector table pointer equal to 0x2FFE00:

```
set_ivtp ((void *) 0x2FFE00);
```

Related Functions

install_int_vector, reset_ivtp

Syntax

```
#include <dma40.h>
void set_pri_auto(void *tab_addr, long ctrl, void *src,
int src_idx, size_t length, void *next_tab)
```

Parameters

*tab_addr	—	Primary channel autoinitialization table pointer
ctrl	—	DMA function control word
*src	—	Data source address for DMA source register
src_idx	—	Source pointer step size for DMA source index register
length	—	Data transfer length
*next_tab	—	Next primary channel autoinitialization table address for link pointer register

Defined in

dma_setp.c in prts40.src or in dma40.h (if INLINE option is used)

Description

The *set_pri_auto* function sets up an autoinitialization table for DMA-split-mode primary-channel autoinitialization.

Example

See the chk_pri_dma function example.

Related Functions

chk_pri_dma, dma_prigo

Syntax

#include <intpt40.h>
void set_tvtp(void *isr);

Parameters

*isr — Trap vector table address

Defined in

set_tvtp.asm in prts40.src

Description

The *set_tvtp* function sets the TVTP to point to the address *isr and saves the old TVTP value to the global variable tvtp_buf (defined in int_tvtp.c). The reset_tvtp function restores the tvtp value back.

Note that TVTP must be on the boundary of 512 words.

Example

Set the trap vector table pointer equal to 0x2FFA00.

```
set_tvtp((void *)0x2ffa00);   /* set the TVTP = 0x2FFA00 */
```

Related Functions

reset_tvtp

Syntax

#include <timer40.h>
void sleep(float x);

Parameters x — CPU delay time in second

Defined in sleep.c in prts40.src

Description

The *sleep* function delays the CPU operation approximately x seconds (include the time of the calling sequence).

Note:

The speed of the processor is target-specific. The default value of CLOCK_PER_SEC is set to 25000000.0 by prts40.src, corresponding to a 50-MHz-input-clock 'C4x. For a different processor speed, you must initialize in your program the global variable CLOCK_PER_SEC (defined in timer40.c) to the desired value. CLOCK_PER_SEC must be set to half the number of input system clocks per second. For example, for a 40-MHz input clock 'C4x:

If X2/XCLKIN = 40 MHz, then H1 = X2/XCLKIN ÷ 2 = 20 MHz
so CLOCK_PER_SEC = H1 = 20000000.0

Example See the alarm function example.

Related Functions install_int_vector, time_delay

Syntax

```
#include <intpt40.h>
int st_value(void);
```

Parameters

None

Defined in

val_st.c in prts40.src

Description

The *st_value* function returns the current data value of the 'C4x CPU register ST (Status).

Example

Read the ST register value from C program.

```
i = st_value();      /* Reads the ST register value   */
```

Related Functions

GET_ST

time_delay *Delays the CPU Operation x Cycles*

Syntax

```
#include <timer40.h>
void time_delay(unsigned long x);
```

Parameters

x — Number of cycles to be delayed

Defined in

t_delay.c in prts40.src

Description

The *time_delay* function delays the CPU operation by x cycles. The delay time includes the overhead time for the calling sequence.

Example

See the count_down function example.

Related Functions

sleep

Syntax

```
#include <timer40.h>
float time_end(void);
```

Parameters

None

Defined in

tim_stop.c in prts40.src

Description

The *time_end* function stops the timer 0 function and returns the time left in the timer counter in seconds. It has a similar function, as does the elapse function. However, the function also stops the timer.

Note:

The speed of the processor is target-specific. The default value of CLOCK_PER_SEC is set to 25000000.0 by prts40.src, corresponding to a 50-MHz-input-clock 'C4x. For a different processor speed, you must initialize in your program the global variable CLOCK_PER_SEC (defined in timer40.c) to the desired value. CLOCK_PER_SEC must be set to half the number of input system clocks per second. For example, for a 40-MHz input clock 'C4x:

If X2/XCLKIN = 40 MHz, then H1 = X2/XCLKIN ÷ 2 = 20 MHz
so CLOCK_PER_SEC = H1 = 20000000.0

Example

See the elapse function example.

Related Functions

time_run, elapse

Syntax

#include <dma40.h>
void time_go(int ch_no, TIMER_REG *register);

Parameters

ch_no	—	Timer channel number (0,1)
*register	—	Timer register structure pointer

Defined in

time_go.c in prts40.src or in timer40.h (if INLINE option is used)

Description

This *time_go* function starts a specified timer channel to perform a specified timer function. The structure TIMER_REG is defined in the header file. It contains the timer-register values for timer function setup. The timer-channel function will be overriden if the timer is used.

Example

Set up the timer 1 to generate a pulse every 4 cycles.

```
TIMER_REG  tim_ptr;
tim_ptr->period   = 3;      /* Set timer period          */
tim_ptr->counter  = 0;      /* Set timer counter         */
tim_ptr->gcontrol = 0x2c1;  /* Set timer control         */
time_go(1, tim_ptr);        /* Start timer 1 function    */
```

Related Functions

dma_go

Syntax

```
#include <timer40.h>
float time_left(void);
```

Parameters

None

Defined in

tim_left.c in prts40.src or in time40.h (if INLINE option is used)

Description

The *time_left* function returns the approximate remaining time of the countdown timer in seconds. It uses 'C4x timer 0.

Note:

The speed of the processor is target-specific. The default value of CLOCK_PER_SEC is set to 25000000.0 by prts40.src, corresponding to a 50-MHz-input-clock 'C4x. For a different processor speed, you must initialize in your program the global variable CLOCK_PER_SEC (defined in timer40.c) to the desired value. CLOCK_PER_SEC must be set to half the number of input system clocks per second. For example, for a 40-MHz input clock 'C4x:

If X2/XCLKIN = 40 MHz, then H1 = X2/XCLKIN ÷ 2 = 20 MHz
so CLOCK_PER_SEC = H1 = 20000000.0

Example

See the alarm function example.

Related Functions

count_left

Syntax #include <timer40.h>
int time_read(int t);

Parameters t — Timer channel number (0,1)

Defined in t_read.c in prts40.src or in timer40.h (if INLINE option is used)

Description The *time_read* function reads the value of the timer-counter register without changing the status of the timer. *t* defines which timer is used. This function can be used as a *stopwatch* for program-code benchmarking.

Example Set up the low-level-timer function to benchmark code with timer 1.

```
#include   <timer40.h>
int cycles1, cycles2, cycles3;

time_start(1);          /* start timer 1 for benchmark    */
        : :       :
*** program code # 1 ***
        : :       :
cycles1 = time_read(1); /* take the first benchmark       */
        : :       :
*** program code # 2 ***
        : :       :
cycles2 = time_read(1); /* take the second benchmark      */
        : :       :
*** program code # 3 ***
        : :       :
cycles3 = time_stop(1); /* take the third benchmark and
                           stop the timer 1               */
```

Related Functions elapse, time_start, time_stop

Syntax

```
#include <timer40.h>
void time_run(void);
```

Parameters

None

Defined in

time_run.c in prts40.src

Description

The *time_run* function starts timer 0 with the period register equal to 0xFFFF FFFF and the counter register equal to 0. It also sets up the interrupt service routine c_int45() to increment the global-memory location time_count. Therefore, the maximum time measured is increased to 2^{64} cycles. For program-code benchmarking, call this function at the beginning of the program and use it with the elapse() function. You must specify in the linker command file that the .vector section be aligned on a 512-word boundary. Refer to the *TMS320 Floating-Point DSP Assembly Language Tools User's Guide* for information regarding linker command files.

Example

See the elapse function example.

Related Functions

c_int45(), elapse, time_end, time_start

Syntax

```
#include <timer40.h>
void time_start(int t);
```

Parameters

t — Timer channel number (0,1)

Defined in

t_start.c in prts40.src or in time40.h (if INLINE option is used)

Description

The *time_start* function starts the timer with the period register set to 0xFFFFFFFF (maximum value) and the counter register set to 0. *t* defines which timer is used. This function can be used with the time_read function for program-code benchmarking. It should be called in the beginning of the program for benchmarking.

Example

See the time_read function example.

Related Functions

time_read, time_run, time_stop

Syntax

#include <timer40.h>
int time_stop(int t);

Parameters

t — Timer channel number (0,1)

Defined in

t_stop.c in prts40.src or in timer40.h (if INLINE option is used)

Description

The *time_stop* function stops a timer and returns the timer-counter value. *t* defines whether timer 0 or timer 1 is stopped and read. time_stop performs the same function as time_read; however, time_stop also stops the timer function.

Example

See the time_read function example.

Related Functions

time_end, time_read, time_start

Syntax
#include <timer40.h>
TIMER_REG *TIMER_ADDR(int ch_no);

Parameters
ch_no — Timer channel number

Defined in
timer40.h (as a macro)

Description
The *TIMER_ADDR* macro sets up the timer-register memory location.

Example
Set up the tim_ptr pointer to point to timer 1.

```
TIMER_REG *tim_ptr = TIMER_ADDR(1);
```

Related Macros
DMA_ADDR, TIMER_ADDR

Syntax

```
#include <intpt40.h>
int tvtp_value(void);
```

Parameters

None

Defined in

val_tvtp.c in prts40.src

Description

The *tvtp_value* function returns the current data value of the 'C4x CPU register TVTP (Trap Vector Table Pointer).

Example

Read the TVTP register value from C program.

```
i = tvtp_value();       /* Reads the TVTP register value */
```

Related Functions/ Macros

GET_TVTP, ivtp_value

Syntax

```
#include <mulpro40.h>
void unlock(int *semaphore);
```

Parameters

*semaphore — semaphore flag pointer

Defined in

unlock.asm in prts40.src

Description

The *unlock* function implements the V(s) function of the shared-memory-interlock operation. It sets shared-memory semaphore, *semaphore, to zero.

Example

See the lock function example.

Related Functions

lock

Syntax

```
#include <compt40.h>
size_t unpack_byte(void *pack_msg, void *msg, size_t msg_size)
```

Parameters

*pack_msg	—	Input data FIFO pointer (source)
*msg	—	Output byte data array pointer (destination)
msg_size	—	Number of byte-wide data to be sent

Defined in

unpack8.asm in prts40.src

Description

The *unpack_byte* function unpacks data from FIFO (or communication port), *pack_msg, to the byte-wide-data array. The argument msg_size provides the input-data-array length, and the function returns the output-unpacked-data-array size. This function is designed mainly for the in_msg8 function. The function can be modified easily for data unpacking.

Example

Refer to the source code of the in_msg8 function in the prts40.src file.

Related Functions

in_msg8, pack_byte

Syntax

```
#include <compt40.h>
size_t unpack_halfword(void *pack_msg, void *msg, size_t msg_size)
```

Parameters

*pack_msg	—	Input data FIFO pointer (source)
*msg	—	Output 16-bit data-array pointer (destination)
msg_size	—	Number of 16-bit-wide data to be sent

Defined in

unpack16.asm in prts40.src

Description

The *unpack_halfword* function unpacks data from FIFO (or communication port) *pack_msg to the halfword-wide-data array. The argument msg_size provides the input-data-array length, and the function returns the output-unpacked-data-array size. This function is designed mainly for the in_msg16 function and can be modified easily for data unpacking.

Example

Refer to the source code of the in_msg16 function in the prts40.src file.

Related Functions

in_msg32, pack_halfword

Syntax

```
#include <timer40.h>
void wakeup(void);
```

Parameters None

Defined in wakeup.c in prts40.src

Description The *wakeup* timer 1 interrupt-service routine wakes up the CPU after the sleep or time_delay functions. After the interrupt has occurred, the routine disables the timer 1 interrupt in the IIE register.

Example See the sleep or time_delay function source codes for examples.

Related Functions install_int_vector, set_ivtp, sleep

Appendix A

Header Files Listing

This appendix lists the Parallel Runtime Support Library header files:

Topic **Page**

A.1 compt40.h

```
/*****************************************************************************/
/* compt40.h ver 4.6 (Refer to the prts40.src file for current version.)     */
/* Copyright (c) 1994 Texas Instruments Incorporated                         */
/*****************************************************************************/
#ifndef    _COMPORT
#define    _COMPORT

#ifndef    _SIZE_T
#define    _SIZE_T
typedef    unsigned        size_t;
#endif

#ifndef    gcontrol
#define    gcontrol        _gctrl._intval
#endif
#ifndef    gcontrol_bit
#define    gcontrol_bit    _gctrl._bitval
#endif

#if        _INLINE
#define    __INLINE        static inline
#else
#define    __INLINE
#endif

#define    RECEIVE_LENGTH  0x0C00049
#define    RECEIVE_DATA    0x0C40045
#define    SEND_LENGTH     0x0C00089
#define    SEND_DATA       0x0C40085
#define    COMPORT_BASE    0x0100040
#define    CP_IN_BASE      0x0100041
#define    CP_OUT_BASE     0x0100042

/*****************************************************************************/
/* MACRO DEFINITIONS FOR COMMUNICATION PORT BASE ADDRESS                     */
/*****************************************************************************/
#define    COMPORT_ADDR(A)       ((COMPORT_REG *)(COMPORT_BASE + (A << 4)))
#define    COMPORT_IN_ADDR(B)    ((long *)(CP_IN_BASE   + (B << 4)))
#define    COMPORT_OUT_ADDR(C)   ((long *)(CP_OUT_BASE  + (C << 4)))

/*****************************************************************************/
/* UNION AND STRUCTURE DEFINITION FOR COMM PORT GLOBAL CONTROL REGISTER      */
/*****************************************************************************/
typedef    union {
           struct {
           unsigned int r_01        :2;    /* Reserved bits 0 & 1            */
           unsigned int port_dir    :1;    /* Comm port direction bit        */
           unsigned int ich         :1;    /* Input fifo halt                */
           unsigned int och         :1;    /* Output fifo halt               */
           unsigned int out_level   :4;    /* Output fifo level              */
           unsigned int in_level    :4;    /* Input fifo level               */
           unsigned int r_rest      :19;   /* Reserved bits                  */
           } _bitval;                      /* Comm port ctrl bits field      */
           unsigned long _intval;          /* Comm port control word         */
         } COMPORT_CONTROL;
```

```
/*********************************************************************************/
/* STRUCTURE DEFINITION COMMUNICATION PORT CONTROL REGISTERS                     */
/*********************************************************************************/
typedef     struct {
            COMPORT_CONTROL   _gctrl;          /* Comm port control reg           */
            unsigned int      in_port;         /* Comm port input register        */
            unsigned int      out_port;        /* Comm port output register       */
            unsigned int      reserved1[13];   /* Unused reserved mem. map        */
       }    COMPORT_REG;

/*********************************************************************************/
/* GLOBAL MEMORY DEFINITION                                                      */
/*********************************************************************************/
extern  size_t msg_size[];                    /* Global memory for message size  */

/*********************************************************************************/
/* FUNCTION DEFINITIONS                                                          */
/*********************************************************************************/
void            pack_byte(void *, void *, size_t);
size_t          unpack_byte(void *, void *, size_t);
void            pack_halfword(void *, void *, size_t);
size_t          unpack_halfword(void *, void *, size_t);

__INLINE long   in_word(int ch_no);
size_t          in_msg(int ch_no, void *message, int step);
size_t          in_msg8(int ch_no, void *message);
size_t          in_msg16(int ch_no, void *message);
void            receive_msg(int ch_no, void *message, int step);

__INLINE void   out_word(long word_value, int ch_no);
void            out_msg(int ch_no, void *message, size_t message_size, int step);
__INLINE void   out_msg8(int ch_no, void *message, size_t message_size);
__INLINE void   out_msg16(int ch_no, void *message, size_t message_size);
void            send_msg(int ch_no, void *message, size_t message_size, int step);

__INLINE int    cp_in_level(int ch_no);
__INLINE void   cp_in_halt(int ch_no);
__INLINE void   cp_in_release(int ch_no);
__INLINE int    cp_out_level(int ch_no);
__INLINE void   cp_out_halt(int ch_no);
__INLINE void   cp_out_release(int ch_no);

#if _INLINE
/*********************************************************************************/
/*  in_word()                                                                    */
/*********************************************************************************/
static inline long in_word(int ch_no)
{
          /*---------------------------------------------------------------*/
          /* SET UP COMM PORT CHANNEL MEMORY POINTER AND SEND OUT DATA     */
          /*---------------------------------------------------------------*/
          COMPORT_REG *cp_ptr = COMPORT_ADDR(ch_no);       /* COMM PORT POINTER */
          return (cp_ptr->in_port);
}
```

```
/****************************************************************************/
/*  out_word()                                                              */
/****************************************************************************/
static inline void out_word(long word_value, int ch_no)
{
          /*----------------------------------------------------------------*/
          /* SET UP COMM PORT CHANNEL MEMORY POINTER AND SEND OUT DATA      */
          /*----------------------------------------------------------------*/
          COMPORT_REG *cp_ptr = COMPORT_ADDR(ch_no);      /* COMM PORT POINTER*/
          cp_ptr->out_port = word_value;
}

/****************************************************************************/
/*  out_msg8()                                                              */
/****************************************************************************/
static inline void out_msg8(int ch_no, void *message, size_t message_size)
{
          /*----------------------------------------------------------------*/
          /* SET UP COMM PORT CHANNEL MEMORY POINTER                        */
          /*----------------------------------------------------------------*/
          COMPORT_REG *cp_ptr = COMPORT_ADDR(ch_no);      /* COMM PORT POINTER*/

          /*----------------------------------------------------------------*/
          /* SEND OUT THE LENGTH AND THE MESSAGE DATA                       */
          /*----------------------------------------------------------------*/
          pack_byte(message, &cp_ptr->out_port, message_size);
}

/****************************************************************************/
/*  out_msg16()                                                             */
/****************************************************************************/
static inline void out_msg16(int ch_no, void *message, size_t message_size)
{
          /*----------------------------------------------------------------*/
          /* SET UP COMM PORT CHANNEL MEMORY POINTER                        */
          /*----------------------------------------------------------------*/
          COMPORT_REG *cp_ptr = COMPORT_ADDR(ch_no);     /* COMM PORT POINTER */

          /*----------------------------------------------------------------*/
          /* PACKED THE DATA AND SEND OUT THE LENGTH AND THE MESSAGE DATA   */
          /*----------------------------------------------------------------*/
          pack_halfword(message, &cp_ptr->out_port, message_size);
}

/****************************************************************************/
/*  cp_in_level()                                                           */
/****************************************************************************/
static inline int cp_in_level(int ch_no)
{
          int     level;

          /*----------------------------------------------------------------*/
          /* SET UP COMM PORT POINTER AND RETURN THE INPUT LEVEL            */
          /*----------------------------------------------------------------*/
          COMPORT_REG *cp_ptr = COMPORT_ADDR(ch_no);      /* COMM PORT POINTER*/
          level  = cp_ptr->gcontrol_bit.in_level;
          return (level == 15) ? 8 : level;
}
```

```
/***************************************************************************/
/*  cp_in_halt ()                                                          */
/*********************************************************************    */
static inline void cp_in_halt(int ch_no)
{
            /*-----------------------------------------------------------------*/
            /* SET UP COMM PORT POINTER AND HALT THE INPUT FIFO                */
            /*-----------------------------------------------------------------*/
            COMPORT_REG *cp_ptr      = COMPORT_ADDR(ch_no);   /* COMM PORT POINTER*/
            cp_ptr->gcontrol_bit.ich = 1;
}

/*********************************************************************    */
/*  cp_in_release()                                                        */
/*********************************************************************    */
static inline void cp_in_release(int ch_no)
{
            /*-----------------------------------------------------------------*/
            /* SET UP COMM PORT POINTER AND UNHALT THE INPUT FIFO              */
            /*-----------------------------------------------------------------*/
            COMPORT_REG *cp_ptr = COMPORT_ADDR(ch_no);  /* COMM PORT POINTER     */
            cp_ptr->gcontrol_bit.ich = 0;
}

/***************************************************************************/
/*  cp_out_level()                                                         */
/***************************************************************************/
static inline int cp_out_level(int ch_no)
{
            int     level;

            /*-----------------------------------------------------------------*/
            /* SET UP COMM PORT POINTER AND RETURN THE OUTPUT LEVEL            */
            /*-----------------------------------------------------------------*/
            COMPORT_REG *cp_ptr = COMPORT_ADDR(ch_no);  /* COMM PORT POINTER     */
            level  = cp_ptr->gcontrol_bit.out_level;
            return (level == 15) ? 8 : level;
}

/***************************************************************************/
/*  cp_out_halt()                                                          */
/***************************************************************************/
static inline void cp_out_halt(int ch_no)
{
            /*-----------------------------------------------------------------*/
            /* SET UP COMM PORT POINTER AND HALT THE OUTPUT FIFO               */
            /*-----------------------------------------------------------------*/
            COMPORT_REG *cp_ptr = COMPORT_ADDR(ch_no);      /* COMM PORT POINTER */
            cp_ptr->gcontrol_bit.och = 1;
}
```

```
/****************************************************************************/
/*  cp_out_release()                                                        */
/****************************************************************************/
static inline void cp_out_release(int ch_no)
{
          /*---------------------------------------------------------------*/
          /* SET UP COMM PORT POINTER AND UNHALT THE OUTPUT FIFO           */
          /*---------------------------------------------------------------*/
          COMPORT_REG *cp_ptr = COMPORT_ADDR(ch_no);     /* COMM PORT POINTER */
          cp_ptr->gcontrol_bit.och = 0;
}

#endif    /*  _INLINE  */

#undef    __INLINE

#endif    /*  compt40.h  */
```

A.2 dma40.h

```
/*****************************************************************************/
/* dma40.h ver 4.6 (Refer to the prts40.src file for current version.)       */
/ *Copyright (c) 1994 Texas Instruments Incorporated                         */
/*****************************************************************************/
/* -6/20/94: changed DMA_MOVE_CONTROL, DMA_CMPLX_REAL, DMA_CMPLX_IMGN define    */
/*           statements from DMA low priority to DMA-CPU rotating priority.  */
/* -6/20/94: changed DMA_INT_TRIG define statement from DMA low priority to  */
/*           DMA high priority (to avoid mutual exclusion problems in dummy  */
/*           1-word read)                                                    */
/*****************************************************************************/
#ifndef    _DMA
#define    _DMA

#ifndef    _SIZE_T
#define    _SIZE_T
typedef    unsigned              size_t;
#endif

#ifndef    gcontrol
#define    gcontrol              _gctrl._intval
#endif
#ifndef    gcontrol_bit
#define    gcontrol_bit          _gctrl._bitval
#endif

#if        _INLINE
#define    __INLINE              static inline
#else
#define    __INLINE
#endif

#define    DMA_MOVE_CONTROL      0x000C40005
#define    DMA_CMPLX_REAL        0x000C02009
#define    DMA_CMPLX_IMGN        0x000C42005
#define    DMA_INT_TRIG          0x000C4004B
#define    DMA_CTRL_BASE         (long *)0x0001000A0
#define    DMA_TRIG_ADDR         (void *)0x02FF800
#define    DMA_STOP              0x0FF3FFFFF
#define    DMA_STOP01            0x000400000
#define    DMA_STOP10            0x000800000
#define    DMAUX_STOP            0x0FCFFFFFF
#define    DMAUX_STOP01          0x001000000
#define    DMAUX_STOP10          0x002000000
#define    DMA_GO11              0x000C00000
#define    DMAUX_GO11            0x003000000
```

```
/*********************************************************************************/
/* MACRO DEFINITIONS                                                              */
/*********************************************************************************/
#define    DMA_ADDR(A)          ((DMA_REG *)(DMA_CTRL_BASE + (A << 4)))
#define    DMA_RESET(B)         (*(DMA_CTRL_BASE + (B << 4)) &= DMA_STOP)
#define    DMA_HALT(C)          (*(DMA_CTRL_BASE+(C<<4))=(*(DMA_CTRL_BASE+(C<<4)) \
                                &DMA_STOP) | DMA_STOP01)
#define    DMA_HALT_B(D)        (*(DMA_CTRL_BASE+(D<<4))=(*(DMA_CTRL_BASE+(D<<4)) \
                                &DMA_STOP) | DMA_STOP10)
#define    DMA_AUX_RESET(E)     (*(DMA_CTRL_BASE + (E << 4)) &= DMAUX_STOP)
#define    DMA_AUX_HALT(F)      (*(DMA_CTRL_BASE+(F<<4))=(*(DMA_CTRL_BASE+(F<<4)) \
                                &DMAUX_STOP) | DMAUX_STOP01)
#define    DMA_AUX_HALT_B(G)    (*(DMA_CTRL_BASE+(G<<4))=(*(DMA_CTRL_BASE+(G<<4)) \
                                &DMAUX_STOP) | DMAUX_STOP10)
#define    DMA_RESTART(H)       (*(DMA_CTRL_BASE + (H << 4)) |= DMA_GO11)
#define    DMA_AUX_RESTART(I)   (*(DMA_CTRL_BASE + (I << 4)) |= DMAUX_GO11)

/*********************************************************************************/
/* UNION AND STRUCTURE DEFINITIONS FOR DMA GLOBAL CONTROL REGISTER              */
/*********************************************************************************/
typedef  union {
 struct {
   unsigned int dma_pri       :2;        /* DMA priority*/
   unsigned int transfer      :2;        /* Transfer mode                         */
   unsigned int aux_transfer  :2;        /* Auxiliary transfer mode               */
   unsigned int sync          :2;        /* Sync. mode                            */
   unsigned int auto_static   :1;        /* Autoinit static                       */
   unsigned int aux_autostat  :1;        /* Aux. autoinit static                  */
   unsigned int auto_sync     :1;        /* Autoinit Sync.                        */
   unsigned int aux_autosync  :1;        /* Aux. autoinit Sync.                   */
   unsigned int rd_bit_rev    :1;        /* Read bit reversed mode                */
   unsigned int wr_bit_rev    :1;        /* Write bit reversed mode               */
   unsigned int split         :1;        /* Split mode                            */
   unsigned int com_port      :3;        /* Communication port                    */
   unsigned int tcc           :1;        /* Transfer counter int.                 */
   unsigned int aux_tcc       :1;        /* Aux. transf count int.                */
   unsigned int tcc_flag      :1;        /* Tcc flag should be 0                  */
   unsigned int aux_tcc_flag  :1;        /* Aux. Tcc should be 0                  */
   unsigned int start         :2;        /* DMA start bits                        */
   unsigned int aux_start     :2;        /* DMA aux. start bits                   */
   unsigned int status        :2;        /* DMA status bits                       */
   unsigned int aux_status    :2;        /* DMA aux. status bits                  */
   unsigned int pri_scheme    :1;        /* Pri. scheme: on DMA0 only             */
   unsigned int r_31          :1;        /* Reserved bit 31                       */
 } _bitval;                              /* DMA control bit fields                */
   unsigned long _intval;                /* DMA control word                      */
 } DMA_CONTROL;
```

```
/*****************************************************************************/
/* STRUCTURE DEFINITION FOR DMA TRANSFER REGISTERS                           */
/*****************************************************************************/
typedef  struct {
         void          *src;          /* Source register                     */
         long          src_idx;       /* Source index register               */
         unsigned long count ;        /* Counter register                    */
         void          *dst;          /* Destination register                */
         long          dst_idx;       /* Destination index reg               */
       } DMA_REGSET;

/*****************************************************************************/
/* STRUCTURE DEFINITION FOR DMA REGISTERS */
/*****************************************************************************/
typedef  struct {
         DMA_CONTROL   _gctrl;        /* Global control register             */
         DMA_REGSET    dma_regs;      /* Src. & dest. regs set               */
         unsigned long *dma_link;     /* Link pointer register               */
         unsigned long dma_aux_count; /* Aux. counter register               */
         unsigned long *dma_aux_link; /* Aux. link pointer reg               */
         unsigned long unused[7];     /* Unused reserved mem. map            */
       } DMA_REG;

/*****************************************************************************/
/* STRUCTURE DEFINITION FOR SPLIT MODE DMA PRIMARY CHANNEL REGISTERS         */
/*****************************************************************************/
typedef  struct {
         DMA_CONTROL        _gctrl;        /* Global control register        */
         void               *dma_src;      /* Source register                */
         long               dma_src_idx;   /* Source index register          */
         unsigned   long    dma_count;     /* Counter register               */
         unsigned   long    *dma_link;     /* Link pointer register          */
       } DMA_PRI_REG;

/*****************************************************************************/
/* STRUCTURE DEFINITION FOR SPLIT MODE DMA AUXILIARY CHANNEL REGISTERS       */
*/****************************************************************************/
typedef  struct {
         DMA_CONTROL        _gctrl;        /* Global control register        */
         void               *dma_dst;      /* Destination register           */
         ong                dma_dst_idx;   /* Destination index reg          */
         unsigned     long  dma_aux_count; /* Aux. counter register          */
         unsigned     long  *dma_aux_link; /* Aux. link pointer reg          */
       } DMA_AUX_REG;
```

```
/*************************************************************************/
/* STRUCTURE DEFINITION FOR DMA AUTOINITIALIZATION TABLE                 */
/*************************************************************************/
typedef  struct {
       unsigned     long    ctrl1;     /* 1st global control reg         */
       void                 *src1;     /* 1st source register            */
       long                 src_idx1;  /* 1st source index register      */
       unsigned     long    count1;    /* 1st counter register           */
       void                 *dst1;     /* 1st destination register       */
       long                 dst_idx1;  /* 1st destination index reg      */
       unsigned     long    *link1;    /* 1st link pointer register      */
       unsigned     long    ctrl2;     /* 2nd global control reg         */
       DMA_REGSET           dma_regs;  /* Src. & dest. regs set          */
       unsigned     long    *link2;    /* 1st link pointer register      */
      } AUTOINIT;

/*************************************************************************/
/* GLOBAL MEMORY DEFINITION                                              */
/*************************************************************************/
extern   AUTOINIT     auto_tab[];          /* Memory for autoinit table     */

/*************************************************************************/
/* FUNCTION DEFINITIONS                                                  */
/*************************************************************************/
__INLINE int  chk_dma(int ch_no);
__INLINE int  chk_pri_dma(int ch_no);
__INLINE int  chk_aux_dma(int ch_no);

void         dma_move(int ch_no, void *src, void *dest, size_t length);
void         dma_cmplx(int ch_no, void *src, void  *dest,
                 size_t fft_size, int priority);
void         dma_int_move(int ex_int, int  ch_no, void *src,
                 void *dest, size_t length);

void             dma_go(int ch_no, DMA_REG *reg);
void             dma_prigo(int ch_no, DMA_PRI_REG *reg);
void             dma_auxgo(int ch_no, DMA_AUX_REG *reg);
void             dma_extrig(int ex_int, int ch_no, DMA_REG *reg);

void             set_dma_auto(void *tab_addr, long  ctrl, void *src, int src_idx,
                   size_t length, void *dest, int dest_idx, void *next_tab);
__INLINE void    set_pri_auto(void *tab_addr, long   ctrl,   void *src,
                   int   src_idx, size_t length, void *next_tab);
__INLINE void    set_aux_auto(void *tab_addr, long   ctrl,   void *dest,
                   int   dest_idx, size_t length, void *next_tab);

__INLINE void    dma_auto_go(int ch_no, long ctrl, void *link_tab);
```

```
#if _INLINE
/****************************************************************************/
/*  chk_dma()                                                               */
/****************************************************************************/
static inline int chk_dma(int ch_no)
{
     /*---------------------------------------------------------------------*/
     /* SEND UP DMA POINTER AND CHECK ON THE START FIELD                     */
     /*---------------------------------------------------------------------*/
     DMA_REG *dma_ptr = DMA_ADDR(ch_no);      /* DMA REGISTER POINTER        */
     return (dma_ptr->gcontrol_bit.start == 3 |
         dma_ptr->gcontrol_bit.aux_start == 3);
}

/****************************************************************************/
/*  chk_pri_dma()                                                           */
/****************************************************************************/
static inline int chk_pri_dma(int ch_no)
{
       /*-------------------------------------------------------------------*/
       /* SEND UP DMA POINTER AND CHECK ON THE START FIELD FIELD             */
       /*-------------------------------------------------------------------*/
       DMA_REG *dma_ptr = DMA_ADDR(ch_no);   /* DMA REGISTER POINTER        */
       return  ((dma_ptr->gcontrol & 0x0C00000) == 0x0C00000);
}

/****************************************************************************/
/*  chk_aux_dma()                                                           */
/****************************************************************************/
static inline int chk_aux_dma(int ch_no)
{
       /*-------------------------------------------------------------------*/
       /* SEND UP DMA POINTER AND CHECK ON THE START FIELD FIELD             */
       /*-------------------------------------------------------------------*/
       DMA_REG *dma_ptr = DMA_ADDR(ch_no);   /* DMA REGISTER POINTER        */
       return  ((dma_ptr->gcontrol & 0x03000000) == 0x03000000);
}

/****************************************************************************/
/*  dma_auto_go()                                                           */
/****************************************************************************/
static inline void dma_auto_go(int ch_no, long ctrl, void *link_tab)
{
       /*-------------------------------------------------------------------*/
       /* SETUP DMA CHANNEL REGISTER POINTER AND START DMA AUTOINIT          */
       /*-------------------------------------------------------------------*/
       DMA_REG *dma_ptr          = DMA_ADDR(ch_no);       /* DMA REGISTER POINTER*/
       dma_ptr->dma_regs.count   = 0;
       dma_ptr->dma_link         = link_tab;
       dma_ptr->gcontrol         = ctrl;
}
```

```
/*****************************************************************************/
/*  set_pri_auto()                                                           */
/*****************************************************************************/
static inline void set_pri_auto(void *tab_addr, long   ctrl,   void *src,
                                int   src_idx,     size_t length, void *next_tab)
{
      DMA_PRI_REG *table = tab_addr;         /* DMA AUTOINIT LINK TABLE          */

      /*---------------------------------------------------------------------*/
      /* SETUP DMA SPLIT MODE AUTOINIT TABLE FOR PRIMARY CHANNEL             */
      /*---------------------------------------------------------------------*/
      table->gcontrol    = ctrl;
      table->dma_src     = src;
      table->dma_src_idx = src_idx;
      table->dma_count   = length;
      table->dma_link    = next_tab;
}

/*****************************************************************************/
/*  set_aux_auto()                                                           */
/*****************************************************************************/
static inline void set_aux_auto(void *tab_addr, long   ctrl,   void *dest,
                                int   dest_idx, size_t length, void *next_tab)
{
      DMA_AUX_REG *table = tab_addr;         /* DMA AUTOINIT LINK TABLE          */

      /*---------------------------------------------------------------------*/
      /* SETUP DMA SPLIT MODE AUTOINIT TABLE FOR AUXILIARY CHANNEL           */
      /*---------------------------------------------------------------------*/
      table->gcontrol            = ctrl;
      table->dma_dst             = dest;
      table->dma_dst_idx         = dest_idx;
      table->dma_aux_count = length;
      table->dma_aux_link  = next_tab;
}

#endif   /*  _INLINE  */

#undef   __INLINE

#endif   /*  dma40.h    */
```

A.3 intpt40.h

```
/*****************************************************************************/
/* intpt40.h ver 4.6 (Refer to the prts40.src file for current version.)     */
/* Copyright (c) 1994 Texas Instruments Incorporated                          */
/*****************************************************************************/
#ifndef    _INTERPT
#define    _INTERPT

#if          _INLINE
#define    __INLINE        static inline
#else
#define    __INLINE
#endif

#ifndef    gcontrol
#define    gcontrol        _gctrl._intval
#endif
#ifndef    gcontrol_bit
#define    gcontrol_bit    _gctrl._bitval
#endif

#ifndef    DEFAULT
#define    DEFAULT         (void *)-1
#endif

#ifndef    lcontrol
#define    lcontrol        _lctrl._intval
#endif
#ifndef    lcontrol_bit
#define    lcontrol_bit    _lctrl._bitval
#endif
```

```
/***************************************************************************/
/* NUMBER DEFINITIONS FOR IIE SETUP FUNCTIONS                              */
/***************************************************************************/
#define    TIMER0         0
#define    ICFULL0        1
#define    ICRDY0         2
#define    OCRDY0         3
#define    OCEMPTY0       4
#define    ICFULL1        5
#define    ICRDY1         6
#define    OCRDY1         7
#define    OCEMPTY1       8
#define    ICFULL2        9
#define    ICRDY2         10
#define    OCRDY2         11
#define    OCEMPTY2       12
#define    ICFULL3        13
#define    ICRDY3         14
#define    OCRDY3         15
#define    OCEMPTY3       16
#define    ICFULL4        17
#define    ICRDY4         18
#define    OCRDY4         19
#define    OCEMPTY4       20
#define    ICFULL5        21
#define    ICRDY5         22
#define    OCRDY5         23
#define    OCEMPTY5       24
#define    DMA0           25
#define    DMA1           26
#define    DMA2           27
#define    DMA3           28
#define    DMA4           29
#define    DMA5           30
#define    TIMER1         31

/***************************************************************************/
/* NUMBER DEFINITIONS FOR IIF SETUP FUNCTIONS                              */
/***************************************************************************/
#define    IIOF0_INT      0
#define    IIOF0_FLAG     2
#define    IIOF1_INT      4
#define    IIOF1_FLAG     6
#define    IIOF2_INT      8
#define    IIOF2_FLAG     10
#define    IIOF3_INT      12
#define    IIOF3_FLAG     14
#define    TIMER0_FLAG    24
#define    DMA0_FLAG      25
#define    DMA1_FLAG      26
#define    DMA2_FLAG      27
#define    DMA3_FLAG      28
#define    DMA4_FLAG      29
#define    DMA5_FLAG      30
#define    TIMER1_FLAG    31
```

```
/*****************************************************************************/
/* MACRO DEFINITIONS                                                         */
/*****************************************************************************/
#define   INT_ENABLE()      asm("    OR     2000H,ST          ;Enable GIE")
#define   INT_DISABLE()     asm("    ANDN   2000H,ST          ;Disable GIE")
#define   CPU_IDLE()        asm("    IDLE                     ;Wait for int")
#define   CACHE_ON()        asm("    OR     0800H,ST          ;Cache on")
#define   CACHE_OFF()       asm("    ANDN   0800H,ST          ;Cache off")
#define   CACHE_FREEZE()    asm("    OR     0400H,ST          ;Cache freeze")
#define   CACHE_DEFROST()   asm("    ANDN   0400H,ST          ;Cache defrost")
#define   CACHE_CLEAR()     asm("    OR     1000H,ST          ;Cache clear")
#define   GET_ST()          asm("    LDI    ST,R0             ;Get ST value")
#define   GET_IIF()         asm("    LDI    IIF,R0            ;Get IIF value")
#define   GET_IIE()         asm("    LDI    IIE,R0            ;Get IIE value")
#define   GET_DIE()         asm("    LDI    DIE,R0            ;Get DIE value")
#define   GET_IVTP()        asm("    LDEP   IVTP,R0           ;Get IVTP value")
#define   GET_TVTP()        asm("    LDEP   TVTP,R0           ;Get TVTP value")
/*****************************************************************************/
/* GLOBAL MEMORY DEFINITIONS                                                 */
/*****************************************************************************/
extern unsigned long      int_vect_buf[];
extern unsigned long      ivtp_buf;
extern unsigned long      tvtp_buf;
extern unsigned long      _vector[];
/*****************************************************************************/
/* FUNCTION DEFINITIONS                                                      */
/*****************************************************************************/
int       chk_iif_flag(int flag_bit);
void      set_iif_flag(int flag_bit);
void      reset_iif_flag(int flag_bit);
void      load_iif(unsigned long iif_value);

int       chk_iie(int enable_bit);
void      set_iie(int enable_bit);
void      reset_iie(int enable_bit);
void      load_iie(unsigned long iie_value);

void      load_die(unsigned long die_data);
void      dma_sync_set(int ch_no, int bit_value, int r_w);
void      set_iiof(int ch_no, int iiof_value);
int       iiof_in(int ch_no);
void      iiof_out(int ch_no, int flag_bit);

void      install_int_vector(void *isr, int N);
void      deinstall_int_vector(int N);
void      set_ivtp(void *int_vect);
void      reset_ivtp();
void      set_tvtp(void *trap_vect);
void      reset_tvtp();

int       st_value();
int       iif_value();
int       iie_value();
int       die_value();
int       ivtp_value();
int       tvtp_value();

#endif    /*  intpt40.h  */
```

A.4 mulpro40.h

```
/****************************************************************************/
/* mulpro40.h ver 4.6 (Refer to the prts40.src file for current version.)   */
/* Copyright (c) 1994 Texas Instruments Incorporated                        */
/****************************************************************************/
#ifndef   _MULTIPRO
#define   _MULTIPRO

#if         _INLINE
#define   __INLINE        static inline
#else
#define   __INLINE
#endif

#ifndef   gcontrol
#define   gcontrol        _gctrl._intval
#endif
#ifndef   gcontrol_bit
#define   gcontrol_bit    _gctrl._bitval
#endif

#ifndef   lcontrol
#define   lcontrol        _lctrl._intval
#endif
#ifndef   lcontrol_bit
#define   lcontrol_bit    _lctrl._bitval
#endif

#ifndef   ID_ADDR
#define   ID_ADDR         (int *)0x02fff00
#endif
/****************************************************************************/
/* MACRO DEFINITIONS                                                        */
/****************************************************************************/
#define   MY_ID()         (*ID_ADDR)

/****************************************************************************/
/* UNION AND STRUCTURE DEFINITIONS FOR DMA GLOBAL CONTROL REGISTER          */
/****************************************************************************/
typedef  union {
          struct {
          unsigned int    ce_0            :1;  /* Control signal 0 enable   */
          unsigned int    ce_1            :1;  /* Control signal 0 enable   */
          unsigned int    de_0            :1;  /* Data bus enable           */
          unsigned int    ae_0            :1;  /* Address bus enable        */
          unsigned int    sww0            :2;  /* STRB0 S/W wait states     */
          unsigned int    sww1            :2;  /* STRB1 S/W wait states     */
          unsigned int    wtcnt0          :3;  /* STRB0 S/W wait state count */
          unsigned int    wtcnt1          :3;  /* STRB1 S/W wait state count */
          unsigned int    pagesize0       :5;  /* STRB0 page size control   */
          unsigned int    pagesize1       :5;  /* STRB1 page size control   */
          unsigned int    strb_active     :5;  /* STRB0,1 active range ctrl */
          unsigned int    strb_switch     :1;  /* STRB switch cycle control */
          unsigned int    r_rest          :2;  /* Reserved bits             */
        } _bitval;                             /* DMA control bit fields    */
          unsigned long   _intval;             /* DMA control word          */
        } BUS_CONTROL;
```

```
/*****************************************************************************/
/* STRUCTURE DEFINITION FOR BUS CONTROL REGISTERS                            */
/*****************************************************************************/
typedef   struct {
          BUS_CONTROL      _gctrl;            /* Global bus ctrl register        */
          unsigned long    reserved_1[3];     /* Unused reserved mem. map        */
          BUS_CONTROL      _lctrl;            /* Local bus ctrl register         */
          unsigned long    reserved_2[11];    /* Unused reserved mem. map        */
       }  BUS_CTRL_REG;

/*****************************************************************************/
/* FUNCTION DEFINITIONS                                                      */
/*****************************************************************************/
int       lock(int *semaphore);
void      unlock(int *semaphore);

#endif    /*  mulpro40.h  */
```

A.5 timer40.h

```
/****************************************************************************/
/*  timer40.h  ver 4.6 (Refer to the prts40.src file for current version.)     */
/*  Copyright (c) 1994 Texas Instruments Incorporated                           */
/****************************************************************************/
/*   -6/20/94: removed CLOCK_PER_SEC define statement and replaced it with a    */
/*    global variable in timer40.c                                              */
/****************************************************************************/
extern float CLOCK_PER_SEC;

#ifndef     _TIMER40
#define     _TIMER40

#if         _INLINE
#define     __INLINE            static inline
#else
#define     __INLINE
#endif

#ifndef     DEFAULT
#define     DEFAULT             (void *)-1
#endif

#define     TIM_START           0x02C1
#define     SLEEP_CALL_DELAY    65
#define     TIME_CALL_DELAY     62
#define     TIMER_BASE          ((TIMER_REG *)0x0100020)
#define     TIMER_SIZE          16
#define     TIMER_CTRL          (long *)0x0100020
#define     TIM_GO              0x0C0
#define     TIM_UNHALT          0x080
#define     TIM_HALT            0x0FF7F
#define     TIMER_CLOCK         (CLOCK_PER_SEC/2.0)

#ifndef     gcontrol
#define     gcontrol            _gctrl._intval
#endif
#ifndef     gcontrol_bit
#define     gcontrol_bit        _gctrl._bitval
#endif

extern      unsigned int        time_count;

/****************************************************************************/
/* MACRO DEFINITIONS                                                        */
/****************************************************************************/
#define     TIMER_ADDR(A)      ((TIMER_REG *)((char *)TIMER_BASE + A * TIMER_SIZE))
#define     TIMER_START(B)     (*(TIMER_CTRL + (B << 4)) |= TIM_GO)
#define     TIMER_HALT(C)      (*(TIMER_CTRL + (C << 4)) &= TIM_HALT)
#define     TIMER_RESTART(D)   (*(TIMER_CTRL + (D << 4)) |= TIM_UNHALT)
```

```
/******************************************************************************/
/* UNION AND STRUCTURE DEFINITION FOR TIMER GLOBAL CONTROL REGISTER          */
/******************************************************************************/
typedef   union {
          struct {
          unsigned int  func      :1;    /* Timer function control              */
          unsigned int  i_o       :1;    /* Input/output control                */
          unsigned int  datout    :1;    /* Data output bit                     */
          unsigned int  datin     :1;    /* Data input bit                      */
          unsigned int  r_45      :2;    /* Reserved bits 4 & 5                 */
          unsigned int  go        :1;    /* Timer GO bit                        */
          unsigned int  hld_      :1;    /* Timer hold                          */
          unsigned int  cp_       :1;    /* Clock/pulse mode                    */
          unsigned int  clksrc    :1;    /* Timer clock source                  */
          unsigned int  inv       :1;    /* Inverter control bit                */
          unsigned int  tstat     :1;    /* Status bit of the timer             */
          unsigned int  r_rest    :20;   /* Reserved bits                       */
        } _bitval;                       /* Timer control bit fields            */
          unsigned long _intval;         /* Timer control word                  */
        } TIMER_CONTROL;

/******************************************************************************/
/* STRUCTURE DEFINITION FOR TIMER CONTROL REGISTERS                           */
/******************************************************************************/
typedef   struct {
          TIMER_CONTROL _gctrl;          /* Timer control register              */
          unsigned long reserved1[3];    /* Unused reserved mem. map            */
          unsigned long counter;         /* Timer counter register              */
          unsigned long reserved2[3];    /* Unused reserved mem. map            */
          unsigned long period;          /* Timer period register               */
          unsigned long reserved3[3];    /* Unused reserved mem. map            */
        } TIMER_REG;

/******************************************************************************/
/* FUNCTION DEFINITIONS                                                       */
/******************************************************************************/
__INLINE  void   time_start(int t);
__INLINE  int    time_read(int t);
__INLINE  int    time_stop(int t);
__INLINE  void   count_down(int t, unsigned long x);
__INLINE  int    count_left(int t);
void             time_delay(unsigned long x);

void             c_int45();
void             wakeup();
__INLINE  void   time_go(int ch_no, TIMER_REG *reg);

void             time_run();
__INLINE  float  elapse();
float            time_end();
__INLINE  void   alarm(float x);
__INLINE  float  time_left();
void             sleep(float x);

__INLINE  int    in_timer(int t);
__INLINE  void   out_timer(int t, int flag);
```

```
#if _INLINE
/*********************************************************************** */
/*  time_start()                                                         */
/*********************************************************************** */
static inline void time_start(int t)
{
        /*----------------------------------------------------------------*/
        /* SET UP TIMER REGISTER POINTER AND START THE TIMER FUNCTION      */
        /*----------------------------------------------------------------*/
        TIMER_REG  *tim_ptr    = TIMER_ADDR(t);     /* TIMER REGISTER POINTER*/
        tim_ptr->period        = -1;
        tim_ptr->gcontrol      = TIM_START;
}

/************************************************************************/
/*  time_read()                                                          */
/*********************************************************************** */
static inline int time_read(int t)
{
        /*----------------------------------------------------------------*/
        /* SET UP TIMER REGISTER POINTER AND RETURN THE COUNTER VALUE      */
        /*----------------------------------------------------------------*/
        TIMER_REG  *tim_ptr = TIMER_ADDR(t);        /* TIMER REGISTER POINTER*/
        return (tim_ptr->counter);
}

/*********************************************************************** */
/*  time_stop()                                                          */
/************************************************************************/
static inline int time_stop(int t)
{
        /*----------------------------------------------------------------*/
        /* SET UP TIMER REGISTER POINTER AND STOP THE TIMER FUNCTION       */
        /*----------------------------------------------------------------*/
        TIMER_REG  *tim_ptr = TIMER_ADDR(t);        /* TIMER REGISTER POINTER*/
        tim_ptr->gcontrol_bit.hld_ = 0;
        return(tim_ptr->counter);
}

/*********************************************************************** */
/*  count_down()                                                         */
/************************************************************************/
static inline void count_down(int t, unsigned long x)
{
        /*----------------------------------------------------------------*/
        /* SET UP TIMER REGISTER POINTER AND START THE TIMER FUNCTION      */
        /*----------------------------------------------------------------*/
        TIMER_REG  *tim_ptr    = TIMER_ADDR(t);     /* TIMER REGISTER POINTER*/
        tim_ptr->period        = x/2;
        tim_ptr->gcontrol      = TIM_START;
}
```

```
/****************************************************************************/
/*  count_left()                                                            */
/****************************************************************************/
static inline int count_left(int t)
{
         /*-----------------------------------------------------------------*/
         /* SET UP TIMER REGISTER POINTER AND RETURN (PERIOD - COUNTER)/2 VALUE*/
         /*-----------------------------------------------------------------*/
         TIMER_REG  *tim_ptr = TIMER_ADDR(t);          /* TIMER REGISTER POINTER*/
         return ((tim_ptr->period - tim_ptr->counter)*2);
}

/****************************************************************************/
/*  time_go()                                                               */
/****************************************************************************/
static inline void time_go(int ch_no, TIMER_REG *reg)
{
         TIMER_REG  *tim_ptr = TIMER_ADDR(ch_no);     /* TIMER REGISTER POINTER */
         /*-----------------------------------------------------------------*/
         /* SETUP DMA CHANNEL REGISTER POINTER AND START DMA FUNCTION        */
         /*-----------------------------------------------------------------*/
         tim_ptr->counter        = reg->counter;
         tim_ptr->period         = reg->period;
         tim_ptr->gcontrol       = reg->gcontrol;
}

/****************************************************************************/
/*  elapse()                                                                */
/****************************************************************************/
static inline float elapse()
{
         float                        x;
         /*-----------------------------------------------------------------*/
         /*   RETURN THE COUNTER VALUE IN SECOND                             */
         /*-----------------------------------------------------------------*/
         TIMER_REG  *tim_ptr = TIMER_BASE;          /* TIMER 0 REGISTER POINTER*/
         x  = (float)time_count*4294967296.0;
         x += (float)tim_ptr->counter;
         return (x/TIMER_CLOCK);
}

/****************************************************************************/
/*  alarm()                                                                 */
/****************************************************************************/
static inline void alarm(float x)
{
         TIMER_REG  *tim_ptr = TIMER_BASE;          /* TIMER 0 REGISTER POINTER*/

         /*-----------------------------------------------------------------*/
         /* START THE TIMER FUNCTION                                         */
         /*-----------------------------------------------------------------*/
         tim_ptr->period   = (unsigned int)(x * TIMER_CLOCK);
         tim_ptr->gcontrol = TIM_START;
}
```

```
/****************************************************************************/
/*  time_left()                                                             */
/****************************************************************************/
static inline float time_left()
{
          TIMER_REG  *tim_ptr = TIMER_BASE;          /* TIMER 0 REGISTER POINTER*/

          /*-----------------------------------------------------------------*/
          /*RETURN (PERIOD - COUNTER) VALUE IN SECOND                         */
          /*-----------------------------------------------------------------*/
          return ((float)(tim_ptr->period - tim_ptr->counter)/TIMER_CLOCK);
}

/****************************************************************************/
/*  in_timer()                                                              */
/****************************************************************************/
static inline int in_timer(int t)
{
          /*-----------------------------------------------------------------*/
          /* SET UP TIMER REGISTER POINTER AND RETURN THE INPUT DATA          */
          /*-----------------------------------------------------------------*/
          TIMER_REG  *tim_ptr = TIMER_ADDR(t);         /* TIMER REGISTER POINTER*/
          tim_ptr->gcontrol       = 0;
          return(tim_ptr->gcontrol_bit.datin);
}

/****************************************************************************/
/*  out_timer()                                                             */
/****************************************************************************/
static inline void out_timer(int t, int flag)
{
          /*-----------------------------------------------------------------*/
          /*   SET UP TIMER REGISTER POINTER AND SET THE OUTPUT DATA FLAG     */
          /*-----------------------------------------------------------------*/
          TIMER_REG  *tim_ptr     = TIMER_ADDR(t);      /* TIMER REGISTER POINTER*/
          tim_ptr->gcontrol       = (flag << 2) | 2;
}

#endif    /*  _INLINE  */

#undef    __INLINE

#endif    /*  timer40.h  */
```

Glossary

A

alignment: A process in which the linker places an output section at an address that falls on an *n*-bit boundary, where *n* is a power of 2. You can specify alignment with the SECTIONS linker directive.

allocation: A process in which the linker calculates the final memory addresses of output sections.

archive library: A collection of individual files that have been grouped into a single file.

ar30: See archiver.

archiver: A software program that allows you to collect several individual files into a single file called an archive library. The archiver also allows you to delete, extract, or replace members of the archive library, as well as add new members.

assembler: A software program that creates a machine-language program from a source file that contains assembly language instructions, directives, and macro directives. The assembler substitutes absolute operation codes for symbolic operation codes, and absolute or relocatable addresses for symbolic addresses.

assembly-time constant: A symbol that is assigned a constant value with the .set directive.

assignment statement: A statement that assigns a value to a variable.

autoinitialization (DMA): Method for automatic reloading a 'C4x DMA channel register file when the transfer counter goes to zero. It allows the 'C4x DMA to autoinitialize itself without CPU intervention.

C

cache: Fast 'C4x on-chip memory that stores often-repeated sections of external-memory code, for faster program execution.

channel (DMA): Each of the six independent processing subunits of the 'C4x DMA coprocessor.

C compiler: A program that translates C source statements into TMS320 floating-point assembly language source statements.

command file: A file that contains linker options and names input files for the linker.

comment: A source statement (or portion of a source statement) that is used to document or improve readability of a source file. Comments are not compiled, assembled, or linked; they have no effect on the object file.

communication port: The 'C4x has six bidirectional, asynchronous communication ports for interprocessor communication at rates up to 20 MB/s.

common object file format (COFF): A binary object file format that promotes modular programming by supporting the concept of *sections*.

configured memory: Memory that the linker has specified for allocation.

constant: A numeric value that can be used as an operand.

cross-reference listing: An output file created by the assembler that lists the symbols that were defined, what line they were defined on, which lines referenced them, and their final values.

D

directive: Special-purpose commands that control the actions and functions of a software tool (as opposed to assembly language instructions, which control the actions of a device).

DMA: Direct Memory Access coprocessor. A self-programmable peripheral that performs data transfers to and from any location in the processor's memory map.

E

executable module: An object file that has been linked and can be executed in a TMS320 system.

expression: A constant, a symbol, or a series of constants and symbols separated by arithmetic operators.

external symbol: A kind of symbol that is either 1) defined in the current module and accessed in another, or 2) accessed in the current module but defined in another.

F

field: For the TMS320, a software-configurable data type whose length can be programmed to be any value in the range of 1-16 bits.

file header: A portion of a COFF object file that contains general information about the object file (such as the number of section headers, the type of system the object file can be downloaded to, the number of symbols in the symbol table, and the symbol table's starting address).

function pointer: In the C language, a variable that contains the address of the entry point to a function.

G

global: A kind of symbol that is either 1) defined in the current module and accessed in another, or 2) accessed in the current module but defined in another.

GIE bit: A global interrupt enable bit (bit 13 in the ST register). It enables CPU interrupts globally.

H

H1: 'C4x output clock. It corresponds to half X2/CLKIN input clock.

header file: A C file with an h extension that declares a set of related functions with the data types and the macros required to use them. To use functions in a header file, the file must be declared in your program using the #include preprocessor directive.

hole: An area between the input sections that compose an output section that contains no actual code or data.

I

IIE register: Internal interrupt enable register. This register enables/disables interrupts for the six communication ports, both timers and the six DMA coprocessor channels.

initialized section: A COFF section that contains executable code or initialized data. An initialized section can be built up with the .data, .text, or .sect directive.

input section: A section from an object file that will be linked into an executable module.

IVTP: 'C4x interrupt vector table pointer register. It points to the beginning of the interrupt vector table.

L

label: A symbol that begins in column 1 of a source statement and corresponds to the address of that statement.

linker: A software tool that combines object files to form an object module that can be allocated into TMS320 system memory and executed by the TMS320.

library build utility: A utility that runs the shell program cl30 on each source file in the archive to either compile or assemble it. It then collects all the object files into the output library.

M

mk30: See library build utility.

member: The elements or variables of a structure, union, archive, or enumeration.

macro: A user-defined routine that can be used as an instruction.

macro call: The process of invoking a macro.

macro definition: A block of source statements that define the name and the code that make up a macro.

macro expansion: The source statements that are substituted for the macro call and are subsequently assembled.

macro library: An archive library composed of macros. Each file in the library must contain one macro; its name must be the same as the macro name it defines, and it must have an extension of .asm.

memory map: A map of TMS320 target system memory space, which is partitioned into functional blocks.

O

object file: A file that has been assembled or linked and contains machine-language object code.

object library: An archive library made up of individual object files.

operand: The arguments, or parameters, of an assembly language instruction, assembler directive, or macro directive.

options: Command parameters that allow you to request additional or specific functions when you invoke a software tool.

output module: A linked, executable object file that can be downloaded and executed on a target system.

S

semaphore: Global variable used to guarantee mutual exclusion in critical sections of code when several processes access same shared data or common resource.

sign-extend: To fill the unused MSBs of a value with the value's sign bit.

source file: A file that contains C code or TMS320 assembly language code that will be compiled or assembled to form an object file.

static: A kind of variable whose scope is confined to a function or a program. The values of static variables are not discarded when the function or program is exited; their previous value is resumed when the function or program is re-entered.

ST register: 'C4x status register that contains global information about the CPU state.

structure: A collection of one or more variables grouped together under a single name.

T

TVTP: 'C4x trap vector table pointer register. It points to the trap vector table (TVT), which defines vectors for 512 traps.

U

union: A variable that may hold (at different times) objects of different types and sizes.

unsigned: A kind of value that is treated as a positive number, regardless of its actual sign.

W

well-defined expression: An expression that contains only symbols or assembly-time constants that have been defined before they appear in the expression.

word: A 32-bit addressable location in target memory.

X

X2/CLKIN: Crystal/oscillator input pin in the 'C4x.

Index

NOTES

NOTES

NOTES

TI Worldwide Sales and Representative Offices

AUSTRALIA / NEW ZEALAND: Texas Instruments Australia Ltd.: Sydney [61] 2-910-3100, Fax 2-805-1186; **Melbourne** 3-696-1211, Fax 3-696-4446.
BELGIUM: Texas Instruments Belgium S.A./N.V.: Brussels [32] (02) 242 75 80, Fax (02) 726 72 76.
BRAZIL: Texas Instrumentos Electronicos do Brasil Ltda.: Sao Paulo [55] 11-535-5133.
CANADA: Texas Instruments Canada Ltd.: Montreal (514) 335-8392; **Ottawa** (613) 726-3201; **Toronto** (416) 884-9181.
DENMARK: Texas Instruments A/S: Ballerup [45] (44) 68 74 00.
FINLAND: Texas Instruments/OY: Espoo [358] (0) 43 54 20 33, Fax (0) 46 73 23.
FRANCE: Texas Instruments France: Velizy-Villacoublay Cedex [33] (1) 30 70 10 01, Fax (1) 30 70 10 54.
GERMANY: Texas Instruments Deutschland GmbH.: Freising [49] (08161) 80-0, Fax (08161) 80 45 16; **Hannover** (0511) 90 49 60, Fax (0511) 64 90 331; **Ostfildern** (0711) 34 03 0, Fax (0711) 34 032 57.
HONG KONG: Texas Instruments Hong Kong Ltd.: Kowloon [852] 956-7288, Fax 956-2200.
HUNGARY: Texas Instruments Representation: Budapest [36] (1) 269 8310, Fax (1) 267 1357.
INDIA: Texas Instruments India Private Ltd.: Bangalore [91] 80 226-9007.
IRELAND: Texas Instruments Ireland Ltd.: Dublin [353] (01) 475 52 33, Fax (01) 478 14 63.
ITALY: Texas Instruments Italia S.p.A.: Agrate Brianza [39] (039) 68 42.1, Fax (039) 68 42.912; **Rome** (06) 657 26 51.
JAPAN: Texas Instruments Japan Ltd.: Tokyo [81] 03-769-8700, Fax 03-3457-6777; **Osaka** 06-204-1881, Fax 06-204-1895; **Nagoya** 052-583-8691, Fax 052-583-8696; **Ishikawa** 0762-23-5471, Fax 0762-23-1583; **Nagano** 0263-33-1060, Fax 0263-35-1025; **Kanagawa** 045-338-1220, Fax 045-338-1255; **Kyoto** 075-341-7713, Fax 075-341-7724; **Saitama** 0485-22-2440, Fax 0425-23-5787; **Oita** 0977-73-1557, Fax 0977-73-1583.
KOREA: Texas Instruments Korea Ltd.: Seoul [82] 2-551-2800, Fax 2-551-2828.
MALAYSIA: Texas Instruments Malaysia: Kuala Lumpur [60] 3-230-6001, Fax 3-230-6605.
MEXICO: Texas Instruments de Mexico S.A. de C.V.: Colina del Valle [52] 5-639-9740.
NORWAY: Texas Instruments Norge A/S: Oslo [47] (02) 264 75 70.
PEOPLE'S REPUBLIC OF CHINA: Texas Instruments China Inc.: Beijing [86] 1-500-2255, Ext. 3750, Fax 1-500-2705.
PHILIPPINES: Texas Instruments Asia Ltd.: Metro Manila [63] 2-817-6031, Fax 2-817-6096.
PORTUGAL: Texas Instruments Equipamento Electronico (Portugal) LDA.: Maia [351] (2) 948 10 03, Fax (2) 948 19 29.
SINGAPORE / INDONESIA / THAILAND: Texas Instruments Singapore (PTE) Ltd.: Singapore [65] 390-7100, Fax 390-7062.
SPAIN: Texas Instruments España S.A.: Madrid [34] (1) 372 80 51, Fax (1) 372 82 66; **Barcelona** (3) 31 791 80.
SWEDEN: Texas Instruments International Trade Corporation (Sverigefilialen): Kista [46] (08) 752 58 00, Fax (08) 751 97 15.
SWITZERLAND: Texas Instruments Switzerland AG: Dietikon [41] 886-2-3771450.
TAIWAN: Texas Instruments Taiwan Limited: Taipei [886] (2) 378-6800, Fax 2-377-2718.
UNITED KINGDOM: Texas Instruments Ltd.: Bedford [44] (0234) 270 111, Fax (0234) 223 459.

UNITED STATES: Texas Instruments Incorporated: ALABAMA: Huntsville (205) 430-0114; **ARIZONA: Phoenix** (602) 244-7800; **CALIFORNIA: Irvine** (714) 660-1200; **San Diego** (619) 278-9600; **San Jose** (408) 894-9000; **Woodland Hills** (818) 704-8100; **COLORADO: Aurora** (303) 368-8000; **CONNECTICUT: Wallingford** (203) 265-3807; **FLORIDA: Orlando** (407) 260-2116; **Fort Lauderdale** (305) 425-7820; **Tampa** (813) 882-0017; **GEORGIA: Atlanta** (404) 662-7967; **ILLINOIS: Arlington Heights** (708) 640-2925; **INDIANA: Indianapolis** (317) 573-6400; **KANSAS: Kansas City** (913) 451-4511; **MARYLAND: Columbia** (410) 312-7900; **MASSACHUSETTS: Boston** (617) 895-9100; **MICHIGAN: Detroit** (303) 553-1500; **MINNESOTA: Minneapolis** (612) 828-9300; **NEW JERSEY: Edison** (908) 906-0033; **NEW MEXICO: Albuquerque** (505) 345-2555; **NEW YORK: Poughkeepsie** (914) 897-2900; **Long Island** (516) 454-6601; **Rochester** (716) 385-6770; **NORTH CAROLINA: Charlotte** (704) 522-5487; **Raleigh** (919) 876-2725; **OHIO: Cleveland** (216) 765-7258; **Dayton** (513) 427-6200; **OREGON: Portland** (503) 643-6758; **PENNSYLVANIA: Philadelphia** (215) 825-9500; **PUERTO RICO: Hato Rey** (809) 753-8700; **TEXAS: Austin** (512) 250-6769; **Dallas** (214) 917-1264; **Houston** (713) 778-6592; **WISCONSIN: Milwaukee** (414) 798-1001.

North American Authorized Distributors

COMMERCIAL	**MILITARY**
Almac / Arrow	Alliance Electronics Inc
Anthem Electronics	Future Electronics (Canada)
Arrow / Schweber	Hamilton Hallmark
Future Electronics (Canada)	Zeus - An Arrow Company
Hamilton Hallmark	**CATALOG**
Marshall Industries	Allied Electronics
Wyle	Arrow Advantage
OBSOLETE PRODUCTS	Newark Electronics
Rochester Electronics 508/462-9332	

For Distributors outside North America, contact your local Sales Office.

A1194

TEXAS INSTRUMENTS

Texas Instruments

Printed in U.S.A., October 1994
2617613-9761 revision A

SPRU084A