IMPORTANT NOTICE

Texas Instruments Incorporated (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Please be aware that TI products are not intended for use in life-support appliances, devices, or systems. Use of TI product in such applications requires the written approval of the appropriate TI officer. Certain applications using semiconductor devices may involve potential risks of personal injury, property damage, or loss of life. In order to minimize these risks, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards. Inclusion of TI products in such applications is understood to be fully at the risk of the customer using TI devices or systems.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.
Read This First

About This Manual

This book is a user’s guide for the TMS320C4x Parallel Runtime Support Library. The library provides a standard method for programming the TMS320C4x 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 TMS320C40 include six direct memory access (DMA) channels, six 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 TMS320C40 can maintain the throughput as well as the numerical execution required for today’s parallel systems.

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 4: Functions Reference. Presents an alphabetical reference of functions with examples.

Appendix A: Listing of Parallel Runtime Support Library Header Files. Lists examples of header files.
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 publications in the following reference list contain useful information regarding functions, operations, and applications of digital signal processing—in particular, image processing. These books also provide other references to many useful technical papers.


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 TMS320C40 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 six DMA channels and six communication ports. Software and hardware applications are included.
**If You Need Assistance. . .**

<table>
<thead>
<tr>
<th>If you want to . . .</th>
<th>Do this. . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request more information about Texas Instruments Digital Signal Processing (DSP) products</td>
<td>Call the CRCT†: (800) 336–5236</td>
</tr>
<tr>
<td></td>
<td>Or write to:</td>
</tr>
<tr>
<td></td>
<td>Texas Instruments Incorporated</td>
</tr>
<tr>
<td></td>
<td>Market Communications Manager, MS 736</td>
</tr>
<tr>
<td></td>
<td>P.O. Box 1443</td>
</tr>
<tr>
<td></td>
<td>Houston, Texas 77251–1443</td>
</tr>
<tr>
<td>Order Texas Instruments documentation</td>
<td>Call the CRCT†: (800) 336–5236</td>
</tr>
<tr>
<td>Ask questions about product operation or report suspected problems</td>
<td>Call the DSP hotline: (713) 274–2320</td>
</tr>
<tr>
<td></td>
<td>or FAX: (713) 274-2324</td>
</tr>
<tr>
<td></td>
<td>or EM ail address: <a href="mailto:438-9750@mcimail.com">438-9750@mcimail.com</a></td>
</tr>
<tr>
<td></td>
<td>or call the DSP Bulletin Board Service: (713) 274-2323</td>
</tr>
<tr>
<td>Report mistakes in this document or any other TI documentation</td>
<td>Fill out and return the reader response card at the end of this book, or send your comments to:</td>
</tr>
<tr>
<td></td>
<td>Texas Instruments Incorporated</td>
</tr>
<tr>
<td></td>
<td>Technical Publications Manager, MS 702</td>
</tr>
<tr>
<td></td>
<td>P.O. Box 1443</td>
</tr>
<tr>
<td></td>
<td>Houston, Texas 77251–1443</td>
</tr>
</tbody>
</table>

† Texas Instruments Customer Response Center
Contents

1 Library Files .......................................................... 1-1

2 Header Files .......................................................... 2-1
   2.1 How Header Files Work ........................................... 2-2
   2.2 Communication Port Functions (compt40.h) .................. 2-3
   2.3 DMA Functions (dma40.h) ....................................... 2-5
   2.4 Interrupt Functions (intpt40.h) ............................... 2-7
   2.5 Multiprocessor Functions (mulpro40.h) ....................... 2-8
   2.6 Timer Functions (timer40.h) .................................. 2-9

3 Summary of Parallel Runtime Support Functions and Macros ........... 3-1
   3.1 Communication Port Functions and Macros ................... 3-2
   3.2 DMA Functions ................................................. 3-4
   3.3 DMA Macros .................................................... 3-5
   3.4 Interrupt Functions ........................................... 3-6
   3.5 Interrupt Macros .............................................. 3-7
   3.6 Multiprocessor Functions and Macros ......................... 3-8
   3.7 Timer Functions and Macros .................................. 3-9

4 Functions Reference ................................................ 4-1

A Listing of Parallel Runtime Support Library Header Files ............ A-1
   A.1 compt40.h ..................................................... A-2
   A.2 dma40.h ....................................................... A-7
   A.3 intpt40.h ..................................................... A-12
   A.4 mulpro40.h ................................................... A-15
   A.5 timer40.h ..................................................... A-17
## Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3–1</td>
<td>Communication Port Functions and Macros</td>
<td>3-2</td>
</tr>
<tr>
<td>3–2</td>
<td>DMA Functions</td>
<td>3-4</td>
</tr>
<tr>
<td>3–3</td>
<td>DMA Macros</td>
<td>3-5</td>
</tr>
<tr>
<td>3–4</td>
<td>Interrupt Functions</td>
<td>3-6</td>
</tr>
<tr>
<td>3–5</td>
<td>Interrupt Macros</td>
<td>3-7</td>
</tr>
<tr>
<td>3–6</td>
<td>Multiprocessor Functions and Macros</td>
<td>3-8</td>
</tr>
<tr>
<td>3–7</td>
<td>Timer Functions and Macros</td>
<td>3-9</td>
</tr>
</tbody>
</table>
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. 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 maximum optimization. The --h compiler option will install 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
c30 -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 TMS320C4x C compiler, refer to the TMS320 Floating-Point DSP Optimizing C Compiler User’s Guide. For information about debugging C source code, refer to the TMS320C4x C Source Debugger User’s Guide.

You can inspect or modify library functions by using the archiver to extract the appropriate source file from the prts40.src file, as shown in the first example on page 1-1. For more information about the archiver, refer to the TMS320 Floating-Point DSP Assembly Language Tools User’s Guide.
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 assembler 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*. 
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:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>How Header Files Work</td>
<td>2-2</td>
</tr>
<tr>
<td>Communication Port Functions (compt40.h)</td>
<td>2-3</td>
</tr>
<tr>
<td>DMA Functions (dma40.h)</td>
<td>2-5</td>
</tr>
<tr>
<td>Interrupt Function (intpt40.h)</td>
<td>2-7</td>
</tr>
<tr>
<td>Multiprocessor Functions (mulpro40.h)</td>
<td>2-8</td>
</tr>
<tr>
<td>Timer Functions (timer40.h)</td>
<td>2-9</td>
</tr>
</tbody>
</table>
2.1 How Header Files Work

In order 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.

```c
#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)`:

```c
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 TMS320C40 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:

```c
COMPORT_REG *cp_ptr = CP_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:

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

Alternately, by integer assignment, this statement:

```c
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 completed information regarding the structure names.

Refer to Appendix A for completed 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.

- **Synchronized transfer functions**—The synchronous communication port transfer functions use the CPU to transfer data between memory and the six TMS320C40 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, CP_IN_ADDR, CP_OUT_ADDR. These macros are used to set up communication port channel pointers. The two data types are:

- **CP_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

- **CP_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`, and `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 programming language 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 TMS320C40 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 TMS320C40 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 fourteen macros are INT_ENABLE, INT_DISABLE, CACHE_ON, CACHE_OFF, CACHE_CLEAR, CACHE_FREEZE, CACHE_DEFROST, CPU_IDLE, GET_ST, GET_DIE, GET_IIE, GET_IIF, GET_IVTP, and GET_TVTP. The CPU_IDELE 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 declares two parallel runtime support 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 pre-defined-memory location, and
- The lock() and unlock() functions implement a shared-memory semaphore.

The mulpro40.h header also declares two functions, one macro, and two data types and defines the default processor identification number location at internal RAM block 1. You can use the #define ID_ADDR preprocessor directive to change the processor identification-number location. The macro MY_ID() is used to read the processor identification number. The lock() and unlock() assembly program functions implement the interlock instructions for accessing the shared-memory semaphore. The two data types are:

- 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 TMS320C40 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 timer functions are listed below.

- **Low-level timer functions**
  - time_start()
  - time_read()
  - time_stop()
  - count_down()
  - count_left()
  - time_delay()

- **High-level timer functions**
  - time_go()
  - time_run()
  - elapse()
  - time_end()
  - alarm()
  - time_left()
  - sleep()

- **General-purpose I/O functions**
  - out_timer()
  - in_timer()

The timer40.h header also declares several functions, one macro, and two data types and defines the default processor speed at 40 ns. You can use the \#define CLOCK_PER_SEC preprocessor directive to change the default processor speed, such as when running a TMS320C40 at 50 ns (see example for elapse() function). The macro TIMER_ADDR is used to set up the different timer pointers. The two data types are:

- **TIM_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

- **TIM_REG**, a structure data type that describes the timer registers.
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:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication Port Functions and Macros</td>
<td>3-2</td>
</tr>
<tr>
<td>DMA Functions</td>
<td>3-4</td>
</tr>
<tr>
<td>DMA Macros</td>
<td>3-5</td>
</tr>
<tr>
<td>Interrupt Functions</td>
<td>3-6</td>
</tr>
<tr>
<td>Interrupt Macros</td>
<td>3-7</td>
</tr>
<tr>
<td>Multiprocessor Functions and Macros</td>
<td>3-8</td>
</tr>
<tr>
<td>Timer Functions and Macros</td>
<td>3-9</td>
</tr>
</tbody>
</table>
3.1 Communication Port Functions and Macros

Table 3–1. Communication Port Functions and Macros

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPORT_REG *COMPORT_ADDR(int ch_no);</td>
<td>Sets up a structure pointer to communication port channel number register address.</td>
</tr>
<tr>
<td>long *CP_IN_ADDR(int ch_no);</td>
<td>Sets up a pointer to communication port channel number input register address.</td>
</tr>
<tr>
<td>long *CP_OUT_ADDR(int ch_no);</td>
<td>Sets up a pointer to communication port channel number output register address.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int cp_in_level(int ch_no);</td>
<td>Checks the communication port channel number input buffer level.</td>
</tr>
<tr>
<td>int cp_out_level(int ch_no);</td>
<td>Checks the communication port channel number output buffer level.</td>
</tr>
<tr>
<td>long in_word(int ch_no);</td>
<td>Reads one-word data from communication port channel number.</td>
</tr>
<tr>
<td>size_t in_msg(int ch_no, void *message, int step);</td>
<td>Reads data from communication port channel number to a word array that is pointed to by *message with the pointer step size (step).</td>
</tr>
<tr>
<td>size_t in_msg8(int ch_no, void *message);</td>
<td>Reads data from communication port channel number and unpacks it to byte-wide-data-array message.</td>
</tr>
<tr>
<td>size_t in_msg16(int ch_no, void *message);</td>
<td>Reads data from communication port channel number and unpacks it to 16-bit-wide-data-array message.</td>
</tr>
<tr>
<td>size_t unpack_byte(void *pack_msg, void *msg, size_t in_size);</td>
<td>Reads in_size 32-bit data from pack_msg and unpacks them to byte-wide data array message.</td>
</tr>
<tr>
<td>size_t unpack_halfword(void *pack_msg, void *msg, size_t in_size);</td>
<td>Reads in_size 32-bit data from pack_msg and unpacks them to 16-bit-wide data array message.</td>
</tr>
<tr>
<td>void cp_in_halt(int ch_no);</td>
<td>Halts the communication port channel number input port.</td>
</tr>
<tr>
<td>void cp_in_release(int ch_no);</td>
<td>Unhalts the communication port channel number input port.</td>
</tr>
<tr>
<td>void cp_out_halt(int ch_no);</td>
<td>Halts the communication port channel number output port.</td>
</tr>
<tr>
<td>void cp_out_release(int ch_no);</td>
<td>Unhalts the communication port channel number output port.</td>
</tr>
<tr>
<td>void out_msg(int ch_no, void *message, size_t message_size, int step);</td>
<td>Writes message_size words data from message with the pointer step size (step) to communication port channel number.</td>
</tr>
<tr>
<td>void out_msg8(int ch_no, void *message, size_t message_size);</td>
<td>Packs message_size bytes data to 32-bit data from message and writes them to communication port channel number.</td>
</tr>
</tbody>
</table>
### Table 3–1. Communication Port Functions and Macros (Concluded)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>void out_msg16(int ch_no, void *message, size_t message_size);</code></td>
<td>Packs <code>message_size</code> 16-bit data to 32-bit data from memory address <code>*message</code> and writes them to communication port channel number.</td>
</tr>
<tr>
<td><code>void out_word(int ch_no);</code></td>
<td>Writes one-word data to communication port channel number.</td>
</tr>
<tr>
<td><code>void pack_byte(void *message, void *pack_msg, size_t in_size);</code></td>
<td>Packs <code>in_size</code> bytes data to 32-bit data from memory address <code>*message</code> and writes them to memory location <code>*pack_msg</code>.</td>
</tr>
<tr>
<td><code>void pack_halfword(void *message, void *pack_msg, size_t in_size);</code></td>
<td>Packs <code>in_size</code> 16-bit data to 32-bit data from memory address <code>*message</code> and writes them to memory location <code>*pack_msg</code>.</td>
</tr>
<tr>
<td><code>void receive_msg(int ch_no, void *message, int step);</code></td>
<td>Asynchronously reads data from communication port channel number to message with a given pointer step size (step).</td>
</tr>
<tr>
<td><code>void send_msg(int ch_no, void *message, size_t message_size, int step);</code></td>
<td>Asynchronously writes <code>message_size</code> words data from memory address <code>*message</code> with a given pointer step size (step) to communication port channel number.</td>
</tr>
</tbody>
</table>
### 3.2 DMA Functions

**Table 3–2. DMA Functions**

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

Table 3–3. DMA Macros

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DMA_REG *DMA_ADDR(int ch_no);</td>
<td>Sets up a structure pointer to DMA channel number register address.</td>
</tr>
<tr>
<td>void DMA_AUX_HALT(int ch_no);</td>
<td>Halts DMA channel number auxiliary channel in read or write boundary.</td>
</tr>
<tr>
<td>void DMA_AUX_HALT_B(int ch_no);</td>
<td>Halts DMA channel number auxiliary channel in read/write boundary.</td>
</tr>
<tr>
<td>void DMA_AUX_RESET(int ch_no);</td>
<td>Resets DMA channel number auxiliary channel.</td>
</tr>
<tr>
<td>void DMA_AUX_RESTART(int ch_no);</td>
<td>Restarts DMA channel number auxiliary channel.</td>
</tr>
<tr>
<td>void DMA_HALT(int ch_no);</td>
<td>Halts DMA channel number primary channel in read or write boundary.</td>
</tr>
<tr>
<td>void DMA_HALT_B(int ch_no);</td>
<td>Halts DMA channel number primary channel in read/write boundary.</td>
</tr>
<tr>
<td>void DMA_RESET(int ch_no);</td>
<td>Resets DMA channel number primary channel.</td>
</tr>
<tr>
<td>void DMA_RESTART(int ch_no);</td>
<td>Restarts DMA channel number primary channel.</td>
</tr>
</tbody>
</table>
## 3.4 Interrupt Functions

### Table 3–4. Interrupt Functions

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

Table 3-5. Interrupt Macros

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void CACHE_CLEAR();</td>
<td>Clears the C40 on-chip cache.</td>
</tr>
<tr>
<td>void CACHE_DEFROST();</td>
<td>Takes the C40 out of the cache freeze mode.</td>
</tr>
<tr>
<td>void CACHE_FREEZE();</td>
<td>Freezes the C40 on-chip cache function.</td>
</tr>
<tr>
<td>void CACHE_OFF();</td>
<td>Turns off the C40 on-chip cache function.</td>
</tr>
<tr>
<td>void CACHE_ON();</td>
<td>Turns on the C40 on-chip cache function.</td>
</tr>
<tr>
<td>void CPU_IDLE();</td>
<td>Sets C40 CPU idle to wait for interrupt.</td>
</tr>
<tr>
<td>void GET_DIE();</td>
<td>Loads DIE register to R0. It is used in the die_value function.</td>
</tr>
<tr>
<td>void GET_IIE();</td>
<td>Loads IIE register to R0. It is used in the iie_value function.</td>
</tr>
<tr>
<td>void GET_IIF();</td>
<td>Loads IIF register to R0. It is used in the iif_value function.</td>
</tr>
<tr>
<td>void GET_IVTP();</td>
<td>Loads IVTP register to R0. It is used in the ivtp_value function.</td>
</tr>
<tr>
<td>void GET_ST();</td>
<td>Loads ST register to R0. It is used in the st_value function.</td>
</tr>
<tr>
<td>void GET_TVTP();</td>
<td>Loads TVTP register to R0. It is used in the tvtp_value function.</td>
</tr>
<tr>
<td>void INT_DISABLE();</td>
<td>Disables the C40 CPU interrupt function globally.</td>
</tr>
<tr>
<td>void INT_ENABLE();</td>
<td>Enables the C40 CPU interrupt function globally.</td>
</tr>
</tbody>
</table>
### 3.6 Multiprocessor Functions and Macros

Table 3–6. Multiprocessor Functions and Macros

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int MY_ID();</td>
<td>Reads the processor-identification number.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>int lock(int *semaphore);</td>
<td>Returns the value of the <code>semaphore</code> and sets it to one.</td>
</tr>
<tr>
<td>void unlock(int *semaphore);</td>
<td>Sets the <code>semaphore</code> to zero.</td>
</tr>
</tbody>
</table>
### 3.7 Timer Functions and Macros

**Table 3–7. Timer Functions and Macros**

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMER_REG *TIMER_ADDR(int ch_no);</td>
<td>Sets up a structure pointer to timer channel number register address.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>alarm</td>
<td>4-4</td>
</tr>
<tr>
<td>c_int45</td>
<td>4-5</td>
</tr>
<tr>
<td>CACHE_CLEAR</td>
<td>4-6</td>
</tr>
<tr>
<td>CACHE_DEFROST</td>
<td>4-7</td>
</tr>
<tr>
<td>CACHE_FREEZE</td>
<td>4-8</td>
</tr>
<tr>
<td>CACHE_OFF</td>
<td>4-9</td>
</tr>
<tr>
<td>CACHE_ON</td>
<td>4-10</td>
</tr>
<tr>
<td>chk_aux_dma</td>
<td>4-11</td>
</tr>
<tr>
<td>chk_dma</td>
<td>4-12</td>
</tr>
<tr>
<td>chk_dma_flag</td>
<td>4-13</td>
</tr>
<tr>
<td>chk_iie</td>
<td>4-14</td>
</tr>
<tr>
<td>chk_iif_flag</td>
<td>4-15</td>
</tr>
<tr>
<td>chk_pri_dma</td>
<td>4-16</td>
</tr>
<tr>
<td>COMPORT_ADDR</td>
<td>4-17</td>
</tr>
<tr>
<td>count_down</td>
<td>4-18</td>
</tr>
<tr>
<td>count_left</td>
<td>4-19</td>
</tr>
<tr>
<td>CP_IN_ADDR</td>
<td>4-20</td>
</tr>
<tr>
<td>cp_in_halt</td>
<td>4-21</td>
</tr>
<tr>
<td>cp_in_level</td>
<td>4-22</td>
</tr>
<tr>
<td>cp_in_release</td>
<td>4-23</td>
</tr>
<tr>
<td>CP_OUT_ADDR</td>
<td>4-24</td>
</tr>
<tr>
<td>cp_out_halt</td>
<td>4-25</td>
</tr>
<tr>
<td>cp_out_level</td>
<td>4-26</td>
</tr>
<tr>
<td>cp_out_release</td>
<td>4-27</td>
</tr>
<tr>
<td>CPU_IDLE</td>
<td>4-28</td>
</tr>
<tr>
<td>deinstall_int_vector</td>
<td>4-29</td>
</tr>
<tr>
<td>die_value</td>
<td>4-30</td>
</tr>
<tr>
<td>DMA_ADDR</td>
<td>4-31</td>
</tr>
<tr>
<td>dma_auto_go</td>
<td>4-32</td>
</tr>
<tr>
<td>dma_auxgo</td>
<td>4-33</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------</td>
</tr>
<tr>
<td>DMA_AUX_HALT</td>
<td>4-34</td>
</tr>
<tr>
<td>DMA_AUX_HALT_B</td>
<td>4-35</td>
</tr>
<tr>
<td>DMA_AUX_RESET</td>
<td>4-36</td>
</tr>
<tr>
<td>DMA_AUX_RESTART</td>
<td>4-37</td>
</tr>
<tr>
<td>dma_cmpix</td>
<td>4-38</td>
</tr>
<tr>
<td>dma_extrig</td>
<td>4-39</td>
</tr>
<tr>
<td>dma_go</td>
<td>4-40</td>
</tr>
<tr>
<td>DMA_HALT</td>
<td>4-41</td>
</tr>
<tr>
<td>DMA_HALT_B</td>
<td>4-42</td>
</tr>
<tr>
<td>dma_int_move</td>
<td>4-43</td>
</tr>
<tr>
<td>dma_move</td>
<td>4-44</td>
</tr>
<tr>
<td>dma_prigo</td>
<td>4-45</td>
</tr>
<tr>
<td>DMA_RESET</td>
<td>4-46</td>
</tr>
<tr>
<td>DMA_RESTART</td>
<td>4-47</td>
</tr>
<tr>
<td>dma_sync_set</td>
<td>4-48</td>
</tr>
<tr>
<td>elapse</td>
<td>4-49</td>
</tr>
<tr>
<td>GET_DIE</td>
<td>4-50</td>
</tr>
<tr>
<td>GET_IIE</td>
<td>4-51</td>
</tr>
<tr>
<td>GET_IIF</td>
<td>4-52</td>
</tr>
<tr>
<td>GET_IVTP</td>
<td>4-53</td>
</tr>
<tr>
<td>GET_ST</td>
<td>4-54</td>
</tr>
<tr>
<td>GET_TVTP</td>
<td>4-55</td>
</tr>
<tr>
<td>iie_value</td>
<td>4-56</td>
</tr>
<tr>
<td>iif_value</td>
<td>4-57</td>
</tr>
<tr>
<td>iiof_in</td>
<td>4-58</td>
</tr>
<tr>
<td>iiof_out</td>
<td>4-59</td>
</tr>
<tr>
<td>in_msg</td>
<td>4-60</td>
</tr>
<tr>
<td>in_msg16</td>
<td>4-61</td>
</tr>
<tr>
<td>in_msg8</td>
<td>4-62</td>
</tr>
<tr>
<td>in_timer</td>
<td>4-63</td>
</tr>
<tr>
<td>in_word</td>
<td>4-64</td>
</tr>
<tr>
<td>install_int_vector</td>
<td>4-65</td>
</tr>
<tr>
<td>INT_DISABLE</td>
<td>4-66</td>
</tr>
<tr>
<td>INT_ENABLE</td>
<td>4-67</td>
</tr>
<tr>
<td>ivtp_value</td>
<td>4-68</td>
</tr>
<tr>
<td>load_die</td>
<td>4-69</td>
</tr>
<tr>
<td>load_iie</td>
<td>4-70</td>
</tr>
<tr>
<td>load_iif</td>
<td>4-71</td>
</tr>
<tr>
<td>lock</td>
<td>4-72</td>
</tr>
<tr>
<td>MY_ID</td>
<td>4-73</td>
</tr>
<tr>
<td>out_msg</td>
<td>4-74</td>
</tr>
<tr>
<td>out_msg16</td>
<td>4-75</td>
</tr>
<tr>
<td>out_msg8</td>
<td>4-76</td>
</tr>
<tr>
<td>out_timer</td>
<td>4-77</td>
</tr>
<tr>
<td>out_word</td>
<td>4-78</td>
</tr>
<tr>
<td>Function</td>
<td>Page</td>
</tr>
<tr>
<td>------------------</td>
<td>------</td>
</tr>
<tr>
<td>pack_byte</td>
<td>4-79</td>
</tr>
<tr>
<td>pack_halfword</td>
<td>4-80</td>
</tr>
<tr>
<td>receive_msg</td>
<td>4-81</td>
</tr>
<tr>
<td>reset_iie</td>
<td>4-82</td>
</tr>
<tr>
<td>reset_iif_flag</td>
<td>4-83</td>
</tr>
<tr>
<td>reset_ivtp</td>
<td>4-84</td>
</tr>
<tr>
<td>reset_tvtp</td>
<td>4-85</td>
</tr>
<tr>
<td>send_msg</td>
<td>4-86</td>
</tr>
<tr>
<td>set_aux_auto</td>
<td>4-87</td>
</tr>
<tr>
<td>set_dma_auto</td>
<td>4-88</td>
</tr>
<tr>
<td>set_dma_flag</td>
<td>4-89</td>
</tr>
<tr>
<td>set_ie</td>
<td>4-90</td>
</tr>
<tr>
<td>set_iif_flag</td>
<td>4-91</td>
</tr>
<tr>
<td>set_iiof</td>
<td>4-92</td>
</tr>
<tr>
<td>set_ivtp</td>
<td>4-93</td>
</tr>
<tr>
<td>set_pri_auto</td>
<td>4-94</td>
</tr>
<tr>
<td>set_tvtp</td>
<td>4-95</td>
</tr>
<tr>
<td>sleep</td>
<td>4-96</td>
</tr>
<tr>
<td>st_value</td>
<td>4-97</td>
</tr>
<tr>
<td>time_delay</td>
<td>4-98</td>
</tr>
<tr>
<td>time_end</td>
<td>4-99</td>
</tr>
<tr>
<td>time_go</td>
<td>4-100</td>
</tr>
<tr>
<td>time_left</td>
<td>4-101</td>
</tr>
<tr>
<td>time_read</td>
<td>4-102</td>
</tr>
<tr>
<td>time_run</td>
<td>4-103</td>
</tr>
<tr>
<td>time_start</td>
<td>4-104</td>
</tr>
<tr>
<td>time_stop</td>
<td>4-105</td>
</tr>
<tr>
<td>TIMER_ADDR</td>
<td>4-106</td>
</tr>
<tr>
<td>tvtp_value</td>
<td>4-107</td>
</tr>
<tr>
<td>unlock</td>
<td>4-108</td>
</tr>
<tr>
<td>unpack_byte</td>
<td>4-109</td>
</tr>
<tr>
<td>unpack_halfword</td>
<td>4-110</td>
</tr>
<tr>
<td>wakeup</td>
<td>4-111</td>
</tr>
</tbody>
</table>
## alarm

**Starts timer 0 With Period Register to ~x Seconds**

### Syntax

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

### Parameters

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

### Defined in

alarm() in prts40.src

### 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 will be executed if the GIE bit, IIE register, and interrupt vector are all configured properly.

Because the speed of the processor is target-system specific, you must use #define to define the speed of the processor, CLOCK_PER_SEC, in seconds by setting the CLOCK_PER_SEC macro to 1/2 the number of input system clocks per second.

### Example

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

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

### Parameters
None

### Defined in
c_int45() 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
CACHE_CLEAR  Clears Cache Memory

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

Parameters
None

Defined in
intpt40.h (as a macro)

Description
CACHE_CLEAR sets bit 12 of the 'C40 status register, ST, clearing the 'C40 on-chip cache.

Example
Clear the 'C40 on-chip cache.

CACHE_CLEAR();       /* Clear the Cache */

Related Macros
CASH_DEFROST, CACHE_FREEZE, CACHE_OFF, CACHE_ON
Disables the Cache Freeze Mode  \textbf{CACHE\\_DEFROST}

\begin{itemize}
  \item Syntax
    \begin{verbatim}
    #include <intpt40.h>
    void CACHE\\_DEFROST(void);
    \end{verbatim}
  \item Parameters
    None
  \item Defined in
    intpt40.h (as a macro)
  \item Description
    \texttt{CACHE\\_DEFROST()} resets bit 10 of the 'C40 status register, ST, unfreezing the 'C40 on-chip cache.
  \item Example
    Take the 'C40 on-chip cache out of freeze mode.
    \begin{verbatim}
    CACHE\\_DEFROST();  /* Defrost the Cache */
    \end{verbatim}
  \item Related Macros
    CACHE\\_CLEAR, CACHE\\_FREEZE, CACHE\\_OFF, CACHE\\_ON
\end{itemize}
CACHE_FREEZE  *Freezes the Cache*

**Syntax**

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

**Parameters**

None

**Defined in**

`intpt40.h` (as a macro)

**Description**

`CACHE_FREEZE` sets bit 10 of the 'C40 status register, ST, freezing the 'C40 on-chip cache.

**Example**

Freeze the 'C40 on-chip cache function.

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

**Related Macros**

`CACHE_CLEAR, CACHE_DEFROST, CACHE_OFF, CACHE_ON`
Disables the Cache  CACHE_OFF

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

Parameters
None

Defined in
intpt40.h (as a macro)

Description
CACHE_OFF resets bit 11 of the ’C40 status register, ST, disabling the ’C40 on-chip cache.

Example
Turn off the ’C40 on-chip cache.
CACHE_OFF();     /* Turns off the Cache */

Related Macros
CACHE_CLEAR, CACHE_DEFROST, CACHE_FREEZE, CACHE_ON
CACHE_ON  Enables the Cache

Syntax

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

Parameters

None

Defined in

intpt40.h (as a macro)

Description

`CACHE_ON` sets bit 11 of the 'C40 status register, ST, enabling the 'C40 on-chip cache.

Example

Turn on the 'C40 on-chip cache.

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

Related Macros

`CACHE_CLEAR, CACHE_DEFROST, CACHE_FREEZE, CACHE_OFF`
Checks if DMA Auxiliary Channel Is in Use  \textbf{chk\_aux\_dma}

\begin{itemize}
\item Syntax
\begin{verbatim}
#include <dma40.h>
int chk_aux_dma(int ch_no);
\end{verbatim}
\item Parameters
\begin{itemize}
\item ch\_no — DMA channel number (0–5)
\end{itemize}
\item Defined in
chk\_aux\_dma() in prts40.src
\item Description
The \textit{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.
\item Example
Check if the auxiliary channel of DMA #3 is busy.
\begin{verbatim}
DMA_AUX_REG *tab, *nextab;
set_aux_auto(tab, DMA_CTRL, dest, dest_indx, size, nextab);

while(chk_aux_dma(3)); /* Check if auxiliary channel of
the DMA # 3 is busy */
dma_auxgo(3, tab); /* Start DMA #3 auxiliary transfer */
\end{verbatim}
\item Related Functions
chk\_dma, chk\_pri\_dma, dma\_auxgo, set\_aux\_auto
\end{itemize}
chk dma  Checks if a DMA Channel Is in Use

Syntax

```
#include <dma40.h>

int chk_dma(int ch_no);
```

Parameters

- `ch_no` — DMA channel number (0–5)

Defined in

chk_dma() in prts40.src

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.

```
DMA_REG *tab, *nextab;

set_dma_auto(tab, DMA_CTRL, src, idxs, size, dest, idxd, nextab);

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**

```c
#include <dma40.h>
int chk_dma_flag(int ch_no);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch_no</td>
<td>DMA channel number (0–5)</td>
</tr>
</tbody>
</table>

**Defined in**

chk_dma_flag() in prts40.src

**Description**

The `chk_dma_flag` function checks whether a specified DMA channel flag in the IIF register has been set. If the return value equals 1, the DMA flag is set. If the return value equals 0, the DMA flag is not set.

**Example**

Check if a specified DMA channel flag in IIF register is set.

```c
dma_move(1, src, dest, size); /* Start DMA #1 transfer */
while(chk_dma_flag(1)==0);     /* Wait for the completion of DMA 1 */
```

**Related Functions**

set_dma_flag
chk_iie  Checks IIE Register Bit

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

Parameters
bit_no — IIE register bit number

Defined in
chk_iie() 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
Checks IIF Flag  \textbf{chk\_iif\_flag}

\begin{verbatim}
Syntax
#include <intpt40.h>
int chk_iif_flag(int bit_no);

Parameters
bit_no — IIF register bit number

Defined in
chk_iif_flag() in prts40.src

Description
The \textit{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.
\begin{verbatim}
if (chk_iif_flag(DMA0_FLAG))    /* Check if DMA0 flag bit, bit25, is set */
{
    //
    //
    //
}
\end{verbatim}

Related Functions
reset_iif_flag, set_iif_flag
\end{verbatim}
chk_pri_dma  Checks if a Specified DMA Primary Channel Is Being Used

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

Parameters
ch_no — DMA channel number (0–5)

Defined in
chk_pri_dma() in prts40.src

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.

DMA_PRI_REG *tab, *nextab;
  :
  :
  set_pri_auto(tab, DMA_CTRL, src, src_indx, size, nextab);
  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
Sets Up Communication Port Register’s Memory Location \textit{COMPORT_ADDR}

Syntax

\begin{verbatim}
#include <compt40.h>
COMPORT_REG *COMPORT_ADDR(int ch_no);
\end{verbatim}

Parameters

\begin{itemize}
  \item \texttt{ch\_no} — Communication port channel number (0–5)
\end{itemize}

Defined in

\texttt{compt40.h} (as a macro)

Description

\textit{COMPORT_ADDR} sets up the communication port register memory location.

Example

Set up a pointer to communication port channel 3.

\begin{verbatim}
COMPORT_REG *cp_ptr = COMPORT_ADDR(3);
\end{verbatim}

Related Macros

\texttt{DMA\_ADDR, TIMER\_ADDR}
**count_down**  
*Starts the Timer With Period Register Equal to x Cycles*

---

**Syntax**

```c
define <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**

`count_down()` in prts40.src

**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

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

**Related Functions**

alarm

---
`count_left` Returns the Remaining Cycle Time for the Timer

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

Parameters
- `t` — Timer channel number (0, 1)

Defined in
`count_left()` in prts40.src

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`
**CP_IN_ADDR**  *Sets Up Communication Port Input Register Memory Location*

**Syntax**

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

**Parameters**

- `ch_no` — Communication port channel number (0–5)

**Defined in**

compt40.h (as a macro)

**Description**

`CP_IN_ADDR` sets up a communication port-input pointer.

**Example**

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

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

**Related Macros**

`CP_OUT_ADDR`
Halts a Specified Communication Port Input Channel  

**cp_in_halt**

Syntax

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

Parameters

- `ch_no` — Communication port channel number (0–5)

Defined in

`cp_in_halt()` in prts40.src

Description

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

Example

Halt communication port 3 input channel.

```c
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 | `cp_in_level()` in prts40.src |
| 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 form comm port 2 */` |
| Related Functions | `cp_in_halt`, `cp_in_release`, `cp_out_level` |
Syntax

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

Parameters

- **ch_no** — Communication port channel number (0–5)

Defined in

- `cp_in_release()` in prts40.src

Description

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

Example

Start communication port 1 input channel.

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

Related Functions

- `cp_in_halt`, `cp_in_level`, `cp_out_release`
CP_OUT_ADDR  Sets Up Communication Port Output Register Memory Location

Syntax
#include <compt40.h>
long *CP_OUT_ADDR(int ch_no);

Parameters
ch_no — Communication port channel number (0–5)

Defined in
compt40.h (as a macro)

Description
The CP_OUT_ADDR 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
CP_IN_ADDR
### cp_out_halt

**Halts a Specified Communication Port Output Channel**

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

#### Parameters
- **ch_no** — Communication port channel number (0–5)

#### Defined in
- `cp_out_halt()` in prts40.src

#### Description
The `cp_out_halt` function halts a specified communication port output channel.

#### Example
Halt communication port 0 output channel.
```c
cp_in_halt(0); /* Halt comm port 0 output channel */
```

#### Related Functions
- `cp_in_level`, `cp_in_release`, `cp_out_halt`
**cp_out_level**  *Returns Output Buffer Level of a Specified Communication Port Channel*

**Syntax**

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

**Parameters**

- `ch_no` — Communication port channel number (0–5)

**Defined in**

`cp_out_level()` in `prts40.src`

**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.

```c
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**

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

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ch_no</td>
<td>Communication port channel number (0–5)</td>
</tr>
</tbody>
</table>

**Defined in**

`cp_out_release()` in `prts40.src`

**Description**

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

**Example**

Start communication port 0 output channel.

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

**Related Functions**

`cp_in_release, cp_out_halt, cp_out_level`
**CPU_IDLE**  
Puts CPU in Idle

**Syntax**

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

**Parameters**
None

**Defined in**
intpt40.h (as a macro)

**Description**
The `CPU_IDLE` function puts the 'C40 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.

```c
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
Restores Interrupt Vector  **deinstall_int_vector**

**Syntax**

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

**Parameters**

N — The number of the interrupt vector location

**Defined in**

dehinstall_int_vector() 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].

```c
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
### die_value  
*Reads DIE Register*

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

#### Parameters
None

#### Defined in
`die_value()` in `prts40.src`

#### Description
The `die_value` function returns the current data value of the 'C40 CPU register, DIE (DMA Interrupt Enable).

#### Example
The example below shows how to get the DIE register value in C program.
```c
i = die_value();    /* Reads the DIE register value */

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

#### Related Functions
`dma_int_move`
Sets Up DMA Registers Memory Location  DMA_ADDR

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 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
#dma_auto_go

**Starts a Specified DMA Channel Unified-Mode Autoinitialization**

## Syntax

```c
#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

dma_auto_go() in prts40.src

## 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 will be overridden if the DMA is busy.

## Example

Start DMA #1 autoinitialization.

```c
DMA_REG *tab, *nextab;
set_dma_auto(tab, DMA_CTRL, src, idxs, size, dest, idxd, nextab);

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`
Starts DMA Split-Mode Auxiliary-Channel Data Transfer  dma_auxgo

Syntax

```c
#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_auxgo() 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 will be 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
DMA_AUX_HALT  Halts the Specified DMA Auxiliary-Channel Function

Syntax

```c
#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

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

**DMA_AUX_HALT_B**

| 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  | `DMA_AUX_HALT_B` 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` |
DMA_AUX_RESET  Resets the Specified DMA Auxiliary-Channel Function

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
Restarts the Specified DMA Auxiliary-Channel Function  DMA_AUX_RESTART

Syntax

```c
#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.

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

Related Macros

- `DMA_AUX_HALT`, `DMA_AUX_HALT_B`, `DMA_AUX_RESET`
**dma_cmplx**  *Transfers a Block of Complex-Number Data Array With Bit-Reversed Addressing*

**Syntax**

```c
#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_cmplx() 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.

**Example**

The function below sets up the `dma_cmplx` 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.

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

**Related Functions**

- `chk_dma`

4-38  Functions Reference
Sets Up a DMA Data Transfer to Be Triggered by an External Interrupt Signal  

dma_extrig

Syntax

```c
#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_extrig() 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 will be overridden if the DMA is busy.

Example

Start DMA #3 to wait for IIOF1 interrupt signal.

```c
DMA_REG *tab, *nextab;
set_dma_auto(tab, DMA_CTRL, src, idxs, size, dest, idxd, nextab);

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`
dma_go  Starts a Specified DMA Channel to Perform DMA Transfer

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() 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 will be overridden if the DMA is busy.

Example
See the chk_dma function example.

Related Functions
chk_dma, set_dma_auto
Halts the Specified DMA Unified/Primary Channel  DMA_HALTS

Syntax

```c
#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

`DMA_HALTS` 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.

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

Related Macros

DMA_HALTS_B, DMA_RESET, DMA_RESTART
DMA_HALT_B  *Halts the Specified DMA Unified/Primary Channel*

**Syntax**

```c
#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` 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.

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

**Related Macros**

- DMA_HALT
- DMA_RESET
- DMA_RESTART
Sets Up an External-Interrupt-Triggered DMA Data Transfer from *src to *dest  dma_int_move

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
dma_int_move() 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 will be 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
dma_move Transfers a Block of Data Array With DMA

Syntax

```c
#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() 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 will be overridden if the DMA is busy.

Example

See `chk_dma_flag` function example.

Related Functions

`chk_dma`
Starts a DMA Split-Mode Primary Channel Data Transfer  dma_prigo

Syntax

```c
#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_prigo() 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 will be 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
DMA_RESET  Resets the Specified DMA Unified/Primary-Channel Function

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 binary 00).

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
Restarts the Specified DMA Unified/Primary Channel  DMA_RESTART

Syntax

```c
#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 binary 11).

Example

Restart DMA unified/primary channel 4.

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

Related Macros

DMA_HALT, DMA_HALT_B, DMA_RESET
dma_sync_set  Sets Up a DIE Register Value for a DMA Channel

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
dma_sync_set() 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
Returns Approximate Elapsed Time  

Syntax

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

Parameters

None

Defined in

elapse() in prts40.src

Description

The `elapse` function returns the approximate elapsed time, in seconds, since
the last `time_go` function call. Because the speed of the processor is target-system specific, you must define the processor speed macro, `CLOCK_PER_SEC`.

The maximum time that the `elapse()` function can handle is about
$2^{64}$ ($1.85 \cdot 10^{19}/\text{CLOCK\_PER\_SEC}$) seconds.

Example

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

```c
#define CLOCK_PER_SEC 20000000.0
#include <timer40.h>
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 the third benchmark and
stop the timer */
```

Related Functions

`time_end`, `time_run`
<table>
<thead>
<tr>
<th><strong>GET_DIE</strong></th>
<th><strong>Loads DIE Register Value to R0 Register</strong></th>
</tr>
</thead>
</table>

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

**Parameters**
None

**Defined in**
`intpt40.h` (as a macro)

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

**Example**
See the `die_value()` source code in Appendix A for an example.

**Related Functions**
die_value
**Syntax**

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

**Parameters**

None

**Defined in**

intpt40.h (as a macro)

**Description**

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

**Example**

See the `iie_value()` source code in Appendix A for an example.

**Related Functions**

`iie_value`
### GET_IIF  Loads IIF Register Value to R0 Register

| Syntax | `#include <intpt40.h>
void GET_IIF(void);` |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameters</td>
<td>None</td>
</tr>
<tr>
<td>Defined in</td>
<td><code>intpt40.h</code> (as a macro)</td>
</tr>
<tr>
<td>Description</td>
<td><code>GET_IIF</code> loads the value of the IIF register into R0. It is used in the <code>iif_value()</code> function.</td>
</tr>
<tr>
<td>Example</td>
<td>See the <code>iif_value()</code> source code in Appendix A for an example.</td>
</tr>
<tr>
<td>Related Functions</td>
<td><code>iif_value</code></td>
</tr>
</tbody>
</table>
**Loads IVTP Register Value to R0 Register**

**GET_IVTP**

**Syntax**

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

**Parameters**

None

**Defined in**

intpt40.h (as a macro)

**Description**

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

**Example**

See the `ivtp_value()` source code in Appendix A for an example.

**Related Functions**

`ivtp_value`
### GET_ST

**Loads ST Register Value to R0 Register**

<table>
<thead>
<tr>
<th>Syntax</th>
</tr>
</thead>
</table>
| #include <intpt40.h>  
  void GET_ST(void); |
| Parameters  | None |
| Defined in  | intpt40.h (as a macro) |
| Description  | GET_ST loads the value of the ST register into R0. It is used in the st_value() function. |
| Example  | See the st_value() source code in Appendix A for an example. |
| Related Functions  | st_value |
GET_TVTP

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

Parameters:  None

Defined in: intpt40.h (as a macro)

Description:  GET_TVTP loads the value of the TVTP register into R0. It is used in the tvtp_value() function.

Example:  See the tvtp_value() source code in Appendix A for an example.

Related Functions:  tvtp_value
iie_value  *Reads IIE Register Value*

**Syntax**

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

**Parameters**

None

**Defined in**

iie_value() in prts40.src

**Description**

The *iie_value* function returns the current data value of the 'C40 CPU register, IIE (Internal Interrupt Enable).

**Example**

Read the IIE register value from C program.

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

**Related Functions/Macros**

GET_IIE, iif_value
Reads IIF Register Value  \texttt{if\_value} \\

Syntax
\begin{verbatim}
#include <intpt40.h>
int iif_value(void);
\end{verbatim}

Parameters
None

Defined in
iif\_value() in prts40.src

Description
The \textit{iif\_value} function returns the current data value of the 'C40 CPU register, IIF (IIOF and Internal Interrupt Flag).

Example
Read the IIF register value from C program.
\begin{verbatim}
i = iif_value();    /* Reads the IIF register value */
\end{verbatim}

Related Functions/
Macros
GET\_IIF, iie\_value
### iiof_in

*Outputs IIOF as General-Purpose Output Pins*

#### Syntax

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

#### Parameters

- **pin_no** — IIOF pin number (0–3)

#### Defined in

iiof_in() 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
Inputs IIOF as General-Purpose Output Pins

**Syntax**

```c
#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()` 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.

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

**Related Functions**

- `iiof_in`
in_msg  Reads Data From a Communication Port Channel

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() 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 5x5 matrix.

```c
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

```c
#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()` 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`.

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

Related Functions

`out_msg16`, `unpack_halfword`
**in_msg8**  
*Reads Data From Communication Port Channel and Unpacks 8-Bit Data*

**Syntax**

```c
#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() 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`.

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

**Related Functions**

out_msg8, unpack_byte
Reads Value of TCLK Pin  \textbf{in\_timer}

Syntax

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

Parameters

- \( t \) — Timer channel number (0,1)

Defined in

`in\_timer()` in prts40.src

Description

The \textit{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.

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

Related Functions

`out\_timer`
in_word  Reads a Word From a Communication Port Channel

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

Parameters
ch_no — Communication port channel number (source)

Defined in
in_word() in prts40.src

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
Sets Up the Interrupt Vector  \texttt{install\_int\_vector}

\textbf{Syntax}

\begin{verbatim}
#include <intpt40.h>
void install_int_vector(void *isr, int N);
\end{verbatim}

\textbf{Parameters}

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

\textbf{Defined in}

install\_int\_vector() in prts40.ssrc

\textbf{Description}

The \texttt{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, \texttt{int\_vect\_buf[N]}.

\textbf{Example}

If the .vector uninitialized section is allocated at 0x2FFE00 but must be on a 512-word boundary, the example below will set 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 will branch to the c\_int02 interrupt-service routine if the GiZ bit of the ST status register and the corresponding IIE bit are preset.

\begin{verbatim}
set_ivtp((void *)0x2ffe00); /* set the IVTP = 0x2FFE00 */
install_int_vector((void *)c\_int02, 2);
\end{verbatim}

\textbf{Related Functions}

deinstall\_int\_vector, set\_ivtp
### INT_DISABLE  Disables Interrupt Globally

**Syntax**

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

**Parameters**

None

**Defined in**

intpt40.h (as a macro)

**Description**

*INT_DISABLE* resets bit 14 (GIE) of the `C40 status register (ST) globally disabling `C40 interrupts.

**Example**

Globally disable `C40 interrupts.

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

**Related Macros**

INT_ENABLE
**Syntax**

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

**Parameters**

None

**Defined in**

intpt40.h (as a macro)

**Description**

`INT_ENABLE` sets bit 14 (GIE) of the `C40 status register (ST)`, globally enabling the `C40` interrupts.

**Example**

Globally enable `C40` interrupts.

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

**Related Macros**

`INT_DISABLE`
ivtp_value  Reads IVTP Register Values

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 'C40 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
**.load_die**

Loads the DIE Register

---

**Syntax**

```c
#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() 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.

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

**Related Functions**

load_iie, load_iif
load_iie  Loads the IIE Register

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() 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
#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() in prts40.src

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

Example
Load 0x800000000 into the IIF register.

load_iif(0x800000000); /* Load data, 80000000h, into IIF register */

Related Functions
load_die, load_iie, reset_iif_flag, set_iif_flag
lock  Implements the P(s) Function

Syntax

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

Parameters

*semaphore — semaphore flag pointer

Defined in

lock() 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
# Reads the Processor-ID Number  
## MY_ID

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

### Parameters
None

### Defined in
mulpro40.h (as a macro)

### Description
The `my_id` macro function reads the processor-identification number from the specified memory that is defined by `#define ID_ADDR`. The default location of `ID_ADDR` is `0x2FFF00`. You must define the proper `ID_ADDR` in your program.

### Example
Read in the processor-identification number from the predefined-memory location `0xC5`.

```c
#define ID_ADDR (int *)0xC5
#include <mulpro40.h>
id_number = MY_ID();  /* read in the processor id */
```

### Related Functions
None
**out_msg**  
_Sends a Word Array to a Communication Port Channel_

**Syntax**
```c
#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() 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 5x5 matrix and sends the data out from communication port number 2.

```c
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
Packs a 16-Bit-Wide Array and Sends It to Specified Channel  

**out_msg16**

### Syntax

```c
#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

`out_msg16()` in `prts40.src`

### 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 will be padded with zeros.

### Example

Pack the 15 16-bit-wide data from 16-bit-wide data array `dat16` to 8 32-bit data and send those packed data out by communication port number 3.

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

### Related Functions

- `in_msg16`
- `pack_halfword`
**out_msg8**  Packs a Byte Array and Sends It to a Specified Channel

**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()` in `prts40.src`

**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, the space will be padded with zeros.

**Example**
Pack the 13-byte-wide data from byte-wide-data array `dat8` to 4 32-bit data and sends those packed data out by communication port number 4.
```
out_msg8(4, dat8, 13); /* pack byte wide data from dat8 and send them to comm port 4 */
```

**Related Functions**
in_msg8, pack_byte
Syntax

```c
#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.

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

Related Functions

`in_timer`
**out_word**  Sends a Word to a Specified Channel

---

**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() in prts40.src

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

**Related Functions**
in_word
Packs the Byte-Wide Data and Sends It to 32-Bit FIFO  

Syntax

```c
#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

pack_byte() 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`
**pack_halfword**  
*Packs the 16-Bit-Wide Data and Sends It to 32-Bit FIFO*

**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**
`pack_halfword()` 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`
Sets Up a DMA to Read Data From a Specified Communication Port  receive_msg

Syntax

```c
#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

`receive_msg()` 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 5x5 matrix. The data transfer is asynchronous to the CPU operation.

```c
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]; /* add the data */
```

Related Functions

`send_msg`
reset_iie  

Disables the CPU Interrupt Individually

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

Parameters
bit_no — IIE register bit number

Defined in
reset_iie() 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**

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

**Parameters**

`bit_no` — IIF register bit number

**Defined in**

`reset_iif_flag()` 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.

```c
reset_iif_flag(DMA4_FLAG);     /* Clear DMA4 flag in IIF */
```

**Related Functions**

`load_iif`, `set_iif_flag`
reset_ivtp  Restores the IVTP Register Value

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

Parameters
None

Defined in
reset_ivtp() 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
Restores the TVTP Register Value  

**reset_tvtp**

| Syntax          | `#include <intpt40.h>`  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>void reset_tvtp(void);</code></td>
</tr>
<tr>
<td>Parameters</td>
<td>None</td>
</tr>
<tr>
<td>Defined in</td>
<td><code>reset_tvtp()</code> in prts40.src</td>
</tr>
<tr>
<td>Description</td>
<td>The <code>reset_tvtp</code> function is a counterpart of the <code>set_tvtp</code> function. It restores the data from the global variable <code>tvtp_buf</code> to the TVTP register.</td>
</tr>
<tr>
<td>Example</td>
<td>Restore the TVTP register value.</td>
</tr>
<tr>
<td></td>
<td><code>reset_tvtp();</code></td>
</tr>
<tr>
<td></td>
<td><code>/* Restore TVTP from tvtp_buf */</code></td>
</tr>
<tr>
<td>Related Functions</td>
<td><code>set_tvtp</code></td>
</tr>
</tbody>
</table>
send_msg  Sets Up a DMA to Send a Word Array to a Specified Communication Port

Syntax

```c
#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() 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 may 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 5x5 matrix and send the data out from communication port number 0. The data transfer is asynchronous to the CPU operation.

```c
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
Sets Up an Autoinitialization Table for DMA-Split-Mode Auxiliary Channel  **set_aux_auto**

| 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  
|                   | length — Data transfer length for auxiliary-counter register  
|                   | *next_tab — Next auxiliary-channel autoinitialization table address for auxiliary link pointer register  
| Defined in        | set_aux_auto() in prts40.src  
| Description       | The set_aux_auto function sets up an autoinitialization table for DMA split-mode auxiliary-channel autoinitialization. Sometimes, it can be used for setting the split-mode auxiliary-channel DMA register's structure.  
| Example           | See the chk_aux_dma function example.  
| Related Functions | chk_aux_dma, dma_auxgo  

4-87
set_dma_auto  Sets Up an Autoinitialization Table for DMA-Unified-Mode Autoinitialization

Syntax

```c
#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

`set_dma_auto()` in prts40.src

Description

The `set_dma_auto` function sets up an autoinitialization table for DMA-unified-mode autoinitialization. Sometimes, 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`
Sets a Specified DMA-Channel Flag

**set_dma_flag**

Syntax

```c
#include <dma40.h>
void set_dma_flag(int ch_no, int flag_value);
```

Parameters

- `ch_no` — DMA channel number
- `flag_value` — The bit value to be put into IIF register

Defined in

`set_dma_flag()` in prts40.src

Description

The `set_dma_flag` function sets a specified DMA-channel flag in the IIF register to `flag_value` (either 0 or 1).

Example

Set the DINT3 flag in the IIF register.

```c
set_dma_flag(3, 0); /* clear DINT3 flag in IIF register */
```

Related Functions

`chk_dma_flag`
**set_iie**  *Enables the CPU Interrupt Individually*

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

**Parameters**  
bit_no — IIE register bit number

**Defined in**  
set_iie() 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
Sets the Interrupt Flag   **set_iif_flag**

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

Parameters
- **bit_no** — IIF register bit number

Defined in
- set_iif_flag() in prts40.src

Description
The `set_iif_flag` 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
**set_iiof**  *Sets Up the IIOF Pin Functions*

**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()` 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`
## Sets Up the Interrupt Vector Table Pointer  

### set_ivtp

**Syntax**

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

**Parameters**

None

**Defined in**

set_ivtp() in prts40.src

**Description**

The `set_ivtp` function sets up the interrupt-vector-table pointer (IVTP) register to point to an uninitialized section named .vector. You can relocate this section in the linker command file.

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

**Example**

See the `install_int_vector` function example.

**Related Functions**

install_int_vector, reset_ivtp
set_pri_auto | Sets Up an Autoinitialization Table for DMA-Split-Mode Primary Channel Autoinitialization

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
set_pri_auto() in prts40.src

Description
The set_pri_auto function sets up an autoinitialization table for DMA-split-mode primary-channel autoinitialization. Sometimes, this function can be used for setting the split-mode primary-channel DMA-register’s structure.

Example
See the chk_pri_dma function example.

Related Functions
chk_pri_dma, dma_prigo
**Syntax**

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

**Parameters**

`*isr` — Trap vector table address

**Defined in**

`set_tvtp()` 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 memory location tvtp_buf.

---

**CAUTION**

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

---

**Example**

Set the trap vector table pointer equal to 0x2FFA00.

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

**Related Functions**

`reset_tvtp`
sleep  Delays CPU Operation

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

Parameters
x — CPU delay time in second

Defined in
sleep() in prts40.src

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

Because the speed of the processor is target-system specific, you must define the processor speed macro, CLOCK_PER_SEC.

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**

`st_value()` in prts40.src

**Description**

The `st_value` function returns the current data value of the 'C40 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**

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

**Parameters**

- `x` — Number of cycles to be delayed

**Defined in**

time_delay() in prts40.src

**Description**

The `time_delay` function delays the CPU operation by `x` cycles. The smallest value of `x` should be greater than 16 cycles. The cycle time includes the time for the calling sequence.

**Example**

See the count_down function example.

**Related Functions**

sleep
Stops the timer 0 Function and Returns the Elapsed Time  \textit{time\_end}

\begin{itemize}
\item Syntax
\begin{verbatim}
#include <timer40.h>
float time_end(void);
\end{verbatim}
\item Parameters
None
\item Defined in
\texttt{time\_end()} in prts40.src
\item Description
The \textit{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.
\begin{quote}
\textbf{CAUTION}
Because the speed of the processor is target-system specific, you must define the processor speed macro, \texttt{CLOCK\_PER\_SEC}.
\end{quote}
\item Example
See the elapse function example.
\item Related Functions
\texttt{time\_run, elapse}
\end{itemize}
**time_go**  
*Starts a Customized Timer Function*

```c
#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()` in prts40.src

**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 overridden if the timer is used.

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

```c
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
Returns Approximate Remaining Time  \texttt{time\_left}

\begin{tabular}{|l|}
\hline
\textbf{Syntax} & \#include <timer40.h> \\
& \texttt{float time\_left(\textit{void});} \\
\hline
\textbf{Parameters} & \texttt{None} \\
\hline
\textbf{Defined in} & \texttt{time\_left() in prts40.src} \\
\hline
\textbf{Description} & The \texttt{time\_left} function returns the approximate remaining time of the countdown timer in seconds. \\
\hline
\end{tabular}

\begin{center}
\textbf{CAUTION}
\end{center}

Because the speed of the processor is target-system specific, you must define the processor speed macro, \texttt{CLOCK\_PER\_SEC.}

\begin{tabular}{|l|}
\hline
\textbf{Example} & See the alarm function example. \\
\hline
\textbf{Related Functions} & \texttt{count\_left} \\
\hline
\end{tabular}
time_read  
Reads the Value of the Timer Counter Register

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

Parameters   
t — Timer channel number (0,1)

Defined in   
time_read() in prts40.src

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 */

Related Functions  
elapse, time_start, time_stop
Syntax

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

Parameters

None

Defined in
time_run() in prts40.src

Description

The `time_run` function starts timer 0 with the period register equal to 0xFFFFFFFF 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 `elapsed()` 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 `elapsed` function example.

Related Functions

c_int45(), elapsed, time_end, time_start
time_start  

Starts the Timer

Syntax

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

Parameters

t — Timer channel number (0,1)

Defined in

time_start() in prts40.src

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
Stops the Timer Function and Returns Timer-Counter Value

Syntax

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

Parameters

t — Timer channel number (0,1)

Defined in

time_stop() in prts40.src

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
The `TIMER_ADDR` sets up the timer registers memory location.

**Syntax**
```c
#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` sets up the timer-register memory location.

**Example**
Set up the `tim_ptr` pointer to point to timer 1.
```c
TIMER_REG *tim_ptr = TIMER_ADDR(1);
```

**Related Macros**
`DMA_ADDR`, `TIMER_ADDR`
**Syntax**

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

**Parameters**

None

**Defined in**

tvtp_value() in prts40.src

**Description**

The `tvtp_value` function returns the current data value of the ’C40 CPU register TVTP (Trap Vector Table Pointer).

**Example**

Read the TVTP register value from C program.

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

**Related Functions/Macros**

GET_TVTP, ivtp_value
**unlock**  
*Implements the V(s) Function*

---

**Syntax**

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

**Parameters**

- `*semaphore` — semaphore flag pointer

**Defined in**

unlock() 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
Unpacks 8-Bit Data From FIFO  

**unpack_byte**

**Syntax**
```c
#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**
unpack_byte() 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
unpack_halfword **Unpacks 16-Bit Data From FIFO**

```c
#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**
- `unpack_halfword()` 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() 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
This appendix lists the Parallel Runtime Support Library header files:

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>compt40.h</td>
<td>A-2</td>
</tr>
<tr>
<td>dma40.h</td>
<td>A-7</td>
</tr>
<tr>
<td>intpt40.h</td>
<td>A-12</td>
</tr>
<tr>
<td>mulpro40.h</td>
<td>A-15</td>
</tr>
<tr>
<td>timer40.h</td>
<td>A-17</td>
</tr>
</tbody>
</table>
A.1 compt40.h

#include <iostream>
#include <vector>

int main() {
    std::vector<int> v;
    return 0;
}

Listing of Parallel Runtime Support Library Header Files
/********************************************************************************/
/* 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);
}
Listing of Parallel Runtime Support Library Header Files
static inline void cp_in_halt(int ch_no)
{
    COMPORT_REG *cp_ptr = COMPORT_ADDR(ch_no); /* COMM PORT POINTER*/
    cp_ptr->gcontrol_bit.ich = 1;
}

static inline void cp_in_release(int ch_no)
{
    COMPORT_REG *cp_ptr = COMPORT_ADDR(ch_no); /* COMM PORT POINTER*/
    cp_ptr->gcontrol_bit.ich = 0;
}

static inline int cp_out_level(int ch_no)
{
    int level;
    COMPORT_REG *cp_ptr = COMPORT_ADDR(ch_no); /* COMM PORT POINTER*/
    level = cp_ptr->gcontrol_bit.out_level;
    return (level == 15) ? 8 : level;
}

static inline void cp_out_halt(int ch_no)
{
    COMPORT_REG *cp_ptr = COMPORT_ADDR(ch_no); /* COMM PORT POINTER*/
    cp_ptr->gcontrol_bit.och = 1;
}
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->control_bit.och = 0;
}
#endif /* _INLINE */
#undef __INLINE
#endif /* compt40.h */
A.2 dma40.h

#ifndef _DMA
#define _DMA
#endif
#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 0x000C40004
#define DMA_CMPLX_REAL 0x000C02008
#define DMA_CMPLX_IMGN 0x000C42004
#define DMA_INT_TRIG 0x000C40048
#define DMA_CTRL_BASE (long *)0x0001000A0
#define DMA_TRIG_ADDR (void *)0x02FF800
#define DMA_STOP 0xFFFF3FFFF
#define DMA_STOP01 0x000400000
#define DMA_STOP10 0x000800000
#define DMA_AUX_STOP 0x00FCFFFFF
#define DMA_AUX_STOP01 0x001000000
#define DMA_AUX_STOP10 0x002000000
#define DMA_GO11 0x000C00000
#define DMA_AUX_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_STOP01)
#define DMA_AUX_RESET(E) (* (DMA_CTRL_BASE + (E << 4))) & DMA_AUX_STOP)
#define DMA_AUX_HALT(F) (* (DMA_CTRL_BASE+(F<<4))) = (* (DMA_CTRL_BASE+(F<<4))) | DMA_AUX_STOP01)
#define DMA_AUX_HALT_B(G) (* (DMA_CTRL_BASE+(G<<4))) = (* (DMA_CTRL_BASE+(G<<4))) | DMA_AUX_STOP01)
#define DMA_RESTART(H) (* (DMA_CTRL_BASE + (H << 4))) = DMA_GO11)
#define DMA_AUX_RESTART(I) (* (DMA_CTRL_BASE + (I << 4))) = DMA_AUX_GO11)
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_auto :1; /* Aux. autoinit static */
        unsigned int auto_sync :1; /* Autoinit Sync. */
        unsigned int aux_auto_sync :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;

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;

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;
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;

typedef struct {
    DMA_CONTROL _gctrl;    /* Global control register */
    void *dma_dst;        /* Destination register */
    long 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;

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;

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

__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);
#define INLINE void set_pri_auto(void *tab_addr, long ctrl, void *src, int src_idx, size_t length, void *next_tab);
#define INLINE void set_aux_auto(void *tab_addr, long ctrl, void *dest, int dest_idx, size_t length, void *next_tab);
#define 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)
{
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)
{
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)
{
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)
{
    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 */
    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 */
    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 */
/* Copyright (c) 1992 Texas Instruments Incorporated */
*******************************************************************************
#ifndef _INTERPT
#define _INTERPT
#endif
#ifndef _INLINE
#define _INLINE static inline
#else
#define _INLINE
#endif
#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

Listing of Parallel Runtime Support Library Header Files

A-12
/* NUMBER DEFINITIONS FOR IIE SETUP FUNCTIONS */
#define TIMER0 0
#define ICFULL0 1
#define OCRDY0 2
#define OCRDY1 3
#define OCRDY2 4
#define IRCY0 5
#define OCRE0 6
#define OCRDY1 7
#define OCRE1 8
#define OCRDY2 9
#define OCRDY3 10
#define OCRDY4 11
#define OCRDY5 12
#define OCRDY6 13
#define OCRDY7 14
#define OCRDY8 15
#define OCRDY9 16
#define OCRDY10 17
#define OCRDY11 18
#define OCRDY12 19
#define OCRDY13 20
#define OCRDY14 21
#define OCRDY15 22
#define OCRDY16 23
#define OCRDY17 24
#define OCRDY18 25
#define DMA0 26
#define DMA1 27
#define DMA2 28
#define DMA3 29
#define DMA4 30
#define DMA5 31
#define TIMER1 32

/* NUMBER DEFINITIONS FOR IIF SETUP FUNCTIONS */
#define IIOF0_INT 0
#define IIOF0_FLAG 1
#define IIOF1_INT 2
#define IIOF1_FLAG 3
#define IIOF2_INT 4
#define IIOF2_FLAG 5
#define IIOF3_INT 6
#define IIOF3_FLAG 7
#define TIMER0_FLAG 8
#define DMA0_FLAG 9
#define DMA1_FLAG 10
#define DMA2_FLAG 11
#define DMA3_FLAG 12
#define DMA4_FLAG 13
#define DMA5_FLAG 14
#define TIMER1_FLAG 15
A-14  Listing of Parallel Runtime Support Library Header Files

intpt40.h

/*******************************************************************************
* 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 */
/* Copyright (c) 1992 Texas Instruments Incorporated */
.fragments
#ifndef _MULTIPRO
#define _MULTIPRO
#endif

#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

.fragments
/* MACRO DEFINITIONS */
.fragments
#define MY_ID() (*ID_ADDR)
.fragments
/* UNION AND STRUCTURE DEFINITIONS FOR DMA GLOBAL CONTROL REGISTER */
.fragments
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;
    unsigned long _intval; /* DMA control bit fields */
} BUS_CONTROL;
.fragments
/********************************************************************************/
/* 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 */
/* Copyright (c) 1992 Texas Instruments Inc. */

#ifndef _TIMER
#define _TIMER

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

#ifndef CLOCK_PER_SEC
#define CLOCK_PER_SEC 25000000.0
#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 *)__built__ + 0x0100020)
#define TIMER_SIZE 16
#define TIMER_CTRL (long *)__built__ + 0x0100020
#define TIM_GO 0x0C0
#define TIM_HALT 0x0FF7F
#define TIMER_CLOCK (CLOCK_PER_SEC/2.0)

extern unsigned int time_count;

/* MACRO DEFINITIONS */

#define TIMER_ADDR(A) ((TIMER_REG *)__built__ + A * TIMER_SIZE)
#define TIMER_START(B) ((__built__ += TIM_GO)
#define TIMER_HALT(C) ((__built__ &= TIM_HALT)
#define TIMER_RESTART(D) ((__built__ = TIM_UNHALT)
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;

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

/* time_go() */

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

/* elapse() */

static inline float elapse()
{
    float x;
    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); /* RETURN THE COUNTER VALUE IN SECOND */
}

/* alarm() */

static inline void alarm(float x)
{
    TIMER_REG  *tim_ptr = TIMER_BASE; /* TIMER 0 REGISTER POINTER*/
    tim_ptr->period = (unsigned int)(x * TIMER_CLOCK); /* START THE TIMER FUNCTION */
    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 */
### Index

#### A
- **alarm() function**, 4-4
- asynchronous transfer functions, 2-3

#### B
- bit-field structures, 2-2

#### C
- **c_int45() function**, 4-5
- CACHE_CLEAR macro, 4-6
- CACHE_DEFROST macro, 4-7
- CACHE_FREEZE macro, 4-8
- CACHE_OFF macro, 4-9
- CACHE_ON macro, 4-10
- chk_aux_dma() function, 4-11
- chk_dma() function, 4-12
- chk_dma_flag() function, 4-13
- chk_iie() function, 4-14
- chk_if_flag() function, 4-15
- chk_pri_dma() function, 4-16
- communication port
  - control functions, 2-3
  - functions, 2-1, 3-2, 3-3
  - macros, 3-7, 3-8
- communication port functions, compt40.h, 2-3, 2-4
- **COMPORT_ADDR macro**, 4-17
- compt40.h, 2-3
  - header files, A-2, A-6
  - control registers, 2-2
- count_down() function, 4-18
- count_left() function, 4-19
  - CP_IN_ADDR macro, 4-20
  - cp_in_halt() function, 4-21
  - cp_in_level() function, 4-22
  - cp_in_release() function, 4-23
  - **CP_OUT_ADDR macro**, 4-24
  - cp_out_halt() function, 4-25
  - cp_out_level() function, 4-26
  - cp_out_release() function, 4-27
  - CPU_IDLE macro, 4-28

#### D
- data structures, 2-2
- data types, 2-4, 2-8
  - structure, 2-8
  - union, 2-8
- deinstall_int_vector() function, 4-29
- die_value() function, 4-30

- **DMA functions**, 3-4
- **DMA macros**, 3-5
- DMA functions, 2-1
  - control, 2-6
  - dma40.h, 2-4, 2-6
  - high-level, 2-5
  - user-customizable, 2-5
- DMA_ADDR macro, 4-31
- dma_auto_go() function, 4-32
- DMA_AUX_HALT macro, 4-34
- DMA_AUX_HALT_B macro, 4-35
- DMA_AUX_RESET macro, 4-36
- DMA_AUX_RESTART macro, 4-37
- dma_auxgo() function, 4-33
- dma_cmplx() function, 4-38
- dma_extrig() function, 4-39
- dma_go() function, 4-40
Index

DMA_HALT macro, 4-41
DMA_HALT_B macro, 4-42
dma_int_move() function, 4-43
dma_move() function, 4-44
dma_prigo() function, 4-45
DMA_RESET macro, 4-46
DMA_RESTART macro, 4-47
dma_sync_set() function, 4-48
dma40.h 2-5
data types declared, 2-6
header files, A-7–A-11
macros declared, 2-6
documentation, iv

E
ellipse() function, 4-49

F
functions
  Parallel Runtime Support library, 2-1
  reference 4-1–4-3

G
GET_DIE macro, 4-50
GET_IIE macro, 4-51
GET_IIF macro, 4-52
GET_IVTP macro, 4-53
GET_ST macro, 4-54
GET_TVTP macro, 4-55

H
  –h compiler option, 1-1
  header files, 1-1  2-1  2-10
  how header files work, 2-2
  how to use this manual, vi

I
iiof_value() function, 4-56
iif_value() function, 4-57
iiof_in() function, 4-58

iiol_out() function, 4-59
in_msg() function, 4-60
in_msg16() function, 4-61
in_msg8() function, 4-62
in_timer() function, 4-63
in_word() function, 4-64
install_int_vector() function, 4-65
INT_DISABLE macro, 4-66
INT_ENABLE macro, 4-67
integer assignment, 2-2
interrupt
  functions, 3-6
  macros, 3-7
interrupt functions, 2-1
  CPU-register setup, 2-7
  CPU-register check-out, 2-7
  general-purpose, 2-7
  intpt40.h, 2-7
  vector setup, 2-7
intpt40.h
  functions, 2-7
  header files, A-12–A-14
ivtp_value() function, 4-68

L
large memory model (–mb), 1-1
library files, 1-1–1-2
literature, iv
load_die() function, 4-69
load_iie() function, 4-70
load_iif() function, 4-71
lock() function, 4-72

M
  –mb, model, 1-1
  –mr convention, 1-1
  macros, 2-3  2-4
    declared by intpt40.h header, 2-7
  model, large memory, 1-1
  mulpro40.h, header files, A-15–A-16
  mulpro40.h functions, 2-8
  multiprocessor
    functions, 3-8
    macros, 3-8

Index-2
Index

multiprocessor functions, 2-1
mulpro40.h, 2-8
MY_ID macro, 4-73

–o2 option, 1-1
out_msg() function, 4-74
out_msg16() function, 4-75
out_msg8() function, 4-76
out_timer() function, 4-77
out_word() function, 4-78

pack_byte() function, 4-79
pack_halfword() function, 4-80
parallel runtime support
functions, 3-1–3-10
macros, 3-1–3-10
summary of functions and macros, 3-1–3-10
Parallel Runtime Support Library, listing of header files, A-1–A-22
peripheral control registers, 2-2
PRTS40 library, 1-1
PRTS40 object library, 1-2
PRTS40.SRC file, 1-1

receive_msg() function, 4-81
references, iv
register-passing parameter convention (–mr), 1-1
registers, peripheral control, 2-2
related documentation, iv
reset_iie() function, 4-82
reset_iif_flag() function, 4-83
reset_ivtp() function, 4-84
reset_tvtp() function, 4-85

set_aux_auto() function, 4-87
set_dma_auto() function, 4-88
set_dma_flag() function, 4-89
set_iie() function, 4-90
set_iif_flag() function, 4-91
set_iiof() function, 4-92
set_ivtp() function, 4-93
set_pri_auto() function, 4-94
set_tvtp() function, 4-95
sleep() function, 4-96
source files, 1-1
st_value() function, 4-97
structure data type, 2-8
synchronous communication port transfer functions, 2-3

time_delay() function, 4-98
time_end() function, 4-99
time_go() function, 4-100
time_left() function, 4-101
time_read() function, 4-102
time_run() function, 4-103
time_start() function, 4-104
time_stop() function, 4-105
timer
functions, 3-1–3-10
macros, 3-1–3-10
timer functions, 2-1
general-purpose, 2-9
high-level, 2-9
low-level, 2-9
timer40.h, 2-9
TIMER_ADDR macro, 4-106	
timer40.h, header files, A-1–A-22
tvtp_value() function, 4-107

union data type, 2-8
unlock() function, 4-108
unpack_byte() function, 4-109
unpack_halfword() function, 4-110

wakeup() function, 4-111