TMS320C6x Peripheral Support Library Programmer's Reference

Literature Number: SPRU273A
April 1998

Printed on Recycled Paper

IMPORTANT NOTICE

Texas Instruments (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 the specifications applicable at the time of sale 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.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

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.

Preface

Read This First

About This Manual

This manual describes the TMS320C6x peripheral support library of functions and macros. This library consists of low-level macros and functions that initialize and control the on-chip peripherals of the TMS320C6x digital signal processor (DSP). This document serves as a reference for the C programmer in creating code for the TMS320C6x.

For a summary of updates in this book, see Appendix C, *Summary of Updates in this Document*.

How to Use This Manual

The information in this document describes the contents of the TMS320C6x peripheral support library in several different ways. Chapters 2 through 4 each contain descriptions of all of the macros and functions within the library, but the chapters organize the information in different ways:

- Chapter 2 provides a discussion of the purposes and actions of each header file and lists the macros and functions it contains. This chapter uses examples to show how these elements are used.
- Chapter 3 also organizes macros and functions by header file, but it simply lists the macros and functions, provides a brief description of each, and gives a page reference to Chapter 4 where more detailed information is available.
- Chapter 4 is an alphabetical reference of all macros and functions. It gives a syntax, description, and provides a code example showing how each is used.

Notational Conventions

This document uses the following conventions:

- Program listings, program examples, and interactive displays are shown in a `special typeface`.
- In syntax descriptions, the function or macro appears in a **bold typeface** and the parameters appear in plainface within parentheses. Portions of a syntax that are in **bold** should be entered as shown; portions of a syntax that are within parentheses describe the type of information that should be entered.
- Macro names are written in uppercase text; function names are written in lowercase.
- The TMS320C6x is also referred to as the ’C6x.

Related Documentation From Texas Instruments

The following books describe the TMS320C6x devices and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477–8924. When ordering, please identify the book by its title and literature number.

TMS320C62x/C67x Technical Brief (literature number SPRU197) gives an introduction to the ’C62x/C67x digital signal processors, development tools, and third-party support.

TMS320C62x/C67x CPU and Instruction Set Reference Guide (literature number SPRU189) describes the ’C62x/C67x CPU architecture, instruction set, pipeline, and interrupts for these digital signal processors.

TMS320C6201/C6701 Peripherals Reference Guide (literature number SPRU190) describes common peripherals available on the TMS320C6201/C6701 digital signal processors. This book includes information on the internal data and program memories, the external memory interface (EMIF), the host port, multichannel buffered serial ports, direct memory access (DMA), clocking and phase-locked loop (PLL), and the power-down modes.

TMS320C62x/C67x Programmer’s Guide (literature number SPRU198) describes ways to optimize C and assembly code for the TMS320C62x/C67x DSPs and includes application program examples.

TMS320C6201 Digital Signal Processor Data Sheet (literature number SPRS051) describes the features of the TMS320C6201 fixed-point DSP and provides pinouts, electrical specifications, and timings for the device.

TMS320C6x Assembly Language Tools User's Guide (literature number SPRU186) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the 'C6x generation of devices.

TMS320C6x Optimizing C Compiler User's Guide (literature number SPRU187) describes the 'C6x C compiler and the assembly optimizer. This C compiler accepts ANSI standard C source code and produces assembly language source code for the 'C6x generation of devices. The assembly optimizer helps you optimize your assembly code.

TMS320C6x C Source Debugger User's Guide (literature number SPRU188) tells you how to invoke the 'C6x simulator and emulator versions of the C source debugger interface. This book discusses various aspects of the debugger, including command entry, code execution, data management, breakpoints, profiling, and analysis.

TMS320C6201 Digital Signal Processor Data Sheet (literature number SPRS051) describes the features of the TMS320C6201 and provides pinouts, electrical specifications, and timings for the device.

TMS320C6x Evaluation Module Reference Guide (literature number SPRU269) provides instructions for installing and operating the 'C6x evaluation module. It also includes support software documentation, application programming interfaces, and technical reference material.

Trademarks

320 Hotline On-line is a trademark of Texas Instruments Incorporated.

If You Need Assistance . . .

❑ World-Wide Web Sites

TI Online	http://www.ti.com
Semiconductor Product Information Center (PIC)	http://www.ti.com/sc/docs/pic/home.htm
DSP Solutions	http://www.ti.com/dsps
320 Hotline On-line™	http://www.ti.com/sc/docs/dsps/support.htm

❑ North America, South America, Central America

Product Information Center (PIC)	(972) 644-5580		
TI Literature Response Center U.S.A.	(800) 477-8924		
Software Registration/Upgrades	(214) 638-0333	Fax: (214) 638-7742	
U.S.A. Factory Repair/Hardware Upgrades	(281) 274-2285		
U.S. Technical Training Organization	(972) 644-5580		
DSP Hotline	(281) 274-2320	Fax: (281) 274-2324	Email: dsph@ti.com
DSP Modem BBS	(281) 274-2323		

DSP Internet BBS via anonymous ftp to ftp://ftp.ti.com/pub/tms320bbs

❑ Europe, Middle East, Africa

European Product Information Center (EPIC) Hotlines:

Multi-Language Support	+33 1 30 70 11 69	Fax: +33 1 30 70 10 32

Email: epic@ti.com

Deutsch	+49 8161 80 33 11 or +33 1 30 70 11 68
English	+33 1 30 70 11 65
Francais	+33 1 30 70 11 64
Italiano	+33 1 30 70 11 67

EPIC Modem BBS	+33 1 30 70 11 99	
European Factory Repair	+33 4 93 22 25 40	
Europe Customer Training Helpline		Fax: +49 81 61 80 40 10

❑ Asia-Pacific

Literature Response Center	+852 2 956 7288	Fax: +852 2 956 2200
Hong Kong DSP Hotline	+852 2 956 7268	Fax: +852 2 956 1002
Korea DSP Hotline	+82 2 551 2804	Fax: +82 2 551 2828
Korea DSP Modem BBS	+82 2 551 2914	
Singapore DSP Hotline		Fax: +65 390 7179
Taiwan DSP Hotline	+886 2 377 1450	Fax: +886 2 377 2718
Taiwan DSP Modem BBS	+886 2 376 2592	

Taiwan DSP Internet BBS via anonymous ftp to ftp://dsp.ee.tit.edu.tw/pub/TI/

❑ Japan

Product Information Center	+0120-81-0026 (in Japan)	Fax: +0120-81-0036 (in Japan)
	+03-3457-0972 or (INTL) 813-3457-0972	Fax: +03-3457-1259 or (INTL) 813-3457-1259
DSP Hotline	+03-3769-8735 or (INTL) 813-3769-8735	Fax: +03-3457-7071 or (INTL) 813-3457-7071
DSP BBS via Nifty-Serve	Type “Go TIASP”	

❑ Documentation

When making suggestions or reporting errors in documentation, please include the following information that is on the title page: the full title of the book, the publication date, and the literature number.

Mail: Texas Instruments Incorporated
Technical Documentation Services, MS 702
P.O. Box 1443
Houston, Texas 77251-1443

Email: dsph@ti.com

Note: When calling a Literature Response Center to order documentation, please specify the literature number of the book.

Contents

Tables

Chapter 1

Introduction

The TMS320C6x peripheral support library is a collection of macros and functions for programming the ’C6x digital signal processor (DSP) registers and peripherals using the C programming language. The library allows the user to control the following:

- **Internal peripherals.** These include the direct memory access (DMA) controller, multichannel buffered serial ports (McBSPs), host port interfaces (HPIs), external memory interface (EMIF) and runtime support timers.
- **Interrupt functionality**. This comes from the memory-mapped interrupt selector registers and the interrupt polarity register, as well as from memory-mapped registers in the control register file.
- **CPU operational modes.** These include big- and little-endian modes, cache control, circular addressing, and power-down modes. These modes are also controlled by registers in the register file.

Topic **Page**

1.1 Source Files Included in Library

The ’C6x peripheral support library consists of several header, C, and assembly source files. These are supplied to the user in the source file devlib6x.src. The following header files included in devlib6x.src provide access to device library macro definitions and functions:

- regs.h
- mcbsp.h
- dma.h
- timer.h
- cache.h
- emif.h
- hpi.h
- intr.h

There are a few additional C and assembly source files that are used to build the devlib6x.lib library file, which is linked into user code. These files include:

- dma.c
- emif.c
- mcbsp.c
- timer.c
- intr_.asm

1.2 Building the TMS320C6x Peripheral Support Library

You must build the ‘C6x peripheral support (object) library before referencing it in the linker command line. The options selected during compile must match those that you used used in building the application code. For instance, if your code is built in big-endian mode with the large memory model, the entire peripheral support library must also be built with these options. The following example extracts source files from devlib6x.src, compiles with the big-endian (–me) and large-memory-model (–ml) options, and archives the resulting object files to produce the peripheral support library, devlib6x.lib:

```
ar6x –x devlib6x.src   ; extracts source files from archive
cl6x –me –ml *.c*.asm   ; compile c and asm source files
ar6x –r devlib6x.lib *.obj; create object library devlib6x.lib
```

You can use many other compiler options to compile the devlib6x.lib library. For more information about the ‘C6x C compiler, and library-build utility, see the *TMS320C6x Optimizing C Compiler User’s Guide*. For information about debugging C source code, see the *TMS320C6x C Source Debugger User’s Guide*.

Peripheral support library functions are handled in one of two ways, depending upon the state of the _INLINE preprocessor symbol when compiling user code. See the *TMS320C6x Optimizing C Compiler User's Guide* for more information on controlling this symbol. If the _INLINE symbol is defined, peripheral support library functions are included as expanded inline code taken from the corresponding header file. If the _INLINE symbol is not defined, the linker uses the peripheral support library code to resolve the external reference. In this case, function calls are generated.

During program linking, the devlib6x.lib object library must be specified as an input file to the linker so that references to the peripheral 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, see the *TMS320C6x Assembly Language Tools User's Guide*.

1.3 Using the TMS320C6x Peripheral Support Library

To use a peripheral support library function or macro, you must first use the #include preprocessor directive to include the header file that declares the function. For example, since the dma_reset() function is declared by the dma.h header file, you must include the dma.h header file as shown before you use the dma_reset( ) function.

```
#include <dma.h>
dma_reset();
```

Header files may be included in any order. However, they must be included before referring to any of the functions that they declare.

Header files also include macros that use #define to perform macro substitution to improve readability. The following example assigns *dma_ptr to point to the DMA channel #1 primary control register using the macro named DMA_PRIMARY_CTRL_ADDR( ):

```
unsigned init *dma_ptr = (unsigned int *)DMA_PRIMARY_CTRL_ADDR(1);
```

Chapter 2

Source Files Description

Source files are C files that declare a set of related functions and macros. To use the elements declared in a header file, each file must be declared in your program using the #include preprocessor directive. This chapter describes each header file, its actions, and the functions and macros within it.

Topic **Page**

2.1 Bit-Field Definitions

Each bit and bit field within memory-mapped peripheral registers on the ‘C6x has a corresponding name (macro define). These macros are defined according to the type of peripheral the header file controls. Each bit field greater than 1 also has an associated macro indicating its length. These macros are identical to the named macros, with the addition of an _SZ suffix. The following code that obtains the current value of the transmit data delay field within the receive control register of the multichannel buffered serial ports shows the macro define and the associated bit-field length macro:

```
#include <regs.h>
#include <mcbsp.h>
{
unsigned int txDataDelay;
unsigned int addr;
addr        = MCBSP_RCR_ADDR(0);
txDataDelay = GET_FIELD(addr,XDATDLY,XDATADLY_SZ)
}
```

Macro define tables that list the bit fields and their relative positions are provided in this chapter for each header file that uses memory-mapped peripheral registers. See the *TMS320C6201/C6701 Peripherals Reference Guide* for the associated control register diagrams.

2.2 Peripheral Support Library Source Files

The following sections describe each of the source files included in the ’C6x peripheral support library. These files are listed according to the peripheral device supported. Each section also lists the macros and functions contained in each file.

2.2.1 Device Register Support (regs.h)

The regs.h header file contains bit and bit-field manipulation macros and defines the non-memory-mapped control registers. The regs.h file is the lowest level file in the peripheral support library and is included by all of the other peripheral-specific header files. It provides two kinds of macros: those that manipulate bits within a register when given its memory-mapped address and those that manipulate bits within non-memory-mapped registers when given the register’s name.

Memory-mapped register bit-manipulation macros are used to control bits and bit fields within the specified register. These macros use four arguments: addr, val, bit, and length. The addr argument refers to the address of the register to control. The bit argument refers to the least significant bit (LSB) location of the field to be controlled. Bit numbers are zero relative, thus a peripheral register's bits are numbered 0 (LSB) through 31 (most significant bit–MSB). The val argument represents the value to write to the specified bit field. The length argument represents the length of the bit field in bits. Note that val is masked by the number of bits specified by length. You must ensure that the value specified can be represented within the number of bits specified by length. The memory-mapped register bit-manipulation macros are as follows:

- ASSIGN_BIT_VAL(addr,bit,val)
- GET_BIT(addr,bit)
- GET_FIELD(addr,bit,length)
- LOAD_FIELD(addr,val,bit,length)
- MASK_BIT(bit)
- MASK_FIELD(bit,length)
- REG_READ(addr)
- REG_WRITE(addr,val)
- RESET_BIT(addr,bit)
- RESET_FIELD(addr,bit,length)
- SET_BIT(addr,bit)

Non-memory-mapped register bit-manipulation macros are used to control bits and bit fields within the specified register when given the register's name. Non-memory-mapped registers are declared with the external cregister volatile keyword in regs.h. The 'C6x compiler extends the C language by adding the cregister keyword to allow high-level access to control registers. The following registers are valid when using this group of macros:

- Addressing mode register (AMR)
- Control status register (CSR)
- Interrupt clear register (ICR)
- Interrupt enable register (IER)
- Interrupt flag register (IFR)
- General-purpose input register (IN)
- Interrupt set register (ISR)
- General-purpose output register (OUT)

For example, the following macro could be used to globally enable interrupts by setting the GIE bit within the CSR:

```
SET_REG_BIT(CSR,GIE);
```

The list of memory-mapped register bit-manipulation macros is shown below:

- GET_REG(reg)
- GET_REG_BIT(reg,bit)
- GET_REG_FIELD(reg,bit,length)
- RESET_REG_BIT(reg,bit)
- SET_REG(reg,val)
- SET_REG_BIT(reg,bit)

Often, macros for other peripherals accomplish the same operations performed by the macros in regs.h. For instance, the intr.h header file defines the macro INTR_GLOBAL_ENABLE, which also sets the GIE bit in the CSR but requires no arguments. The regs.h file is the lowest-level include file and is used by the peripheral-specific include files. In fact, the macro INTR_GLOBAL_ENABLE is defined in intr.h as follows:

```
#define INTR_GLOBAL_ENABLE      SET_REG_BIT(CSR, GIE)
```

Although these two methods of setting the GIE bit are exactly the same, use of the higher-level macro INTR_GLOBAL_ENABLE improves code readability and demonstrates why you should use these higher-level macros when they are available.

Table 2–1 through Table 2–7 show the macro defines for the regs.h file, listed by the register to which they belong.

Table 2–1. AMR Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
A4_MODE	0	A4_MODE_SZ	2
A5_MODE	2	A5_MODE_SZ	2
A6_MODE	4	A6_MODE_SZ	2
A7_MODE	6	A7_MODE_SZ	2
B4_MODE	8	B4_MODE_SZ	2
B5_MODE	10	B5_MODE_SZ	2
B6_MODE	12	B6_MODE_SZ	2
B7_MODE	14	B7_MODE_SZ	2
BK0	16	BK0_SZ	5
BK1	21	BK1_SZ	5

Table 2–2. CSR Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
GIE	0	—	1
PGIE	1	—	1
DCC	2	DCC_SZ	3
PCC	5	PCC_SZ	3
EN	8	—	1
SAT	9	—	1
PWRD	10	PWRD_SZ	6
REVISION_ID	16	REVISION_ID_SZ	8
CPU_ID	24	CPU_ID_SZ	8

Table 2–3. IFR Bits and Bit-Relative Positions

Bit Field	Relative Position
NMIF	1
IF4	4
IF5	5
IF6	6
IF7	7
IF8	8
IF9	9
IF10	10
IF11	11
IF12	12
IF13	13
IF14	14
IF15	15

Table 2–4. ISR Bits and Bit-Relative Positions

Bit Field	Relative Position
IS4	4
IS5	5
IS6	6
IS7	7
IS8	8
IS9	9
IS10	10
IS11	11
IS12	12
IS13	13
IS14	14
IS15	15

Table 2–5. ICR Bits and Bit-Relative Positions

Bit Field	Relative Position
IC4	4
IC5	5
IC6	6
IC7	7
IC8	8
IC9	9
IC10	10
IC11	11
IC12	12
IC13	13
IC14	14
IC15	15

Table 2–6. IER Bits and Bit-Relative Positions

Bit Field	Relative Position
NMIE	1
IE4	4
IE5	5
IE6	6
IE7	7
IE8	8
IE9	9
IE10	10
IE11	11
IE12	12
IE13	13
IE14	14
IE15	15

Table 2–7. ISTP Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
ISTB	10	ISTB_SZ	22
HPEINT	5	HPEINT_SZ	5

The following examples show the use of macros, functions, and defines in the device register support header file:

Example 1: The following code reads the revision ID field of the CSR to determine the revision ID of the ’C6x on which the code is currently running:

```
unsigned int revision_id;
revision_id=GET_REG_FIELD(CSR,REVISION_ID,REVISION_ID_SZ);
```

Example 2: The following macro generates the INT4 interrupt and sets the corresponding bit in the ISR:

```
SET_REG_BIT(ISR,IS4);
```

Example 3: The following code reads the current value of the timer 0 counter register:

```
unsigned int count_val;
count_val = REG_READ(TIMER_COUNTER_ADDR(0));
```

Note that the macro function TIMER_COUNTER_ADDR(chan), which returns the address of the indicated timer counter register, is supplied in the timer.h file. This file must be included for this code to work.

2.2.2 Cache Support (cache.h)

The cache.h file provides macro functions for controlling the mode of the internal program memory of the ‘C6x. These macros manipulate the program cache control (PCC) field of the CPU CSR. There are no arguments to any of these macro functions. The following is a list of the macros supplied in cache.h:

- CACHE_BYPASS( )
- CACHE_DISABLE( )
- CACHE_ENABLE( )
- CACHE_FLUSH( )
- CACHE_FREEZE( )
- IDLE( )

The following examples show the use of macros, functions, and defines in the cache support header file:

Example 1: The following call enables the internal program memory as program cache:

```
CACHE_ENABLE( );
```

Example 2: The following call returns the program memory area to mapped mode:

```
CACHE_DISABLE( );
```

2.2.3 Direct Memory Access Support (dma.h, dma.c)

The dma.h and dma.c files provide macros and functions that control the operation of the ‘C6x DMA controller. Functions are provided in dma.c (as well as their corresponding inline functions in dma.h) to initialize and reset all channels of the DMA controller. Control macros are provided in dma.h that start operation in normal and autoinitialization modes, pause, and stop the indicated DMA channel. These functions and macros are listed below:

The reset and initialization functions are as follows:

- dma_global_init(gcr,gcra,gcrb,gndxa,gndxb,gaddra,gaddrb,gaddrc, gaddrd)
- dma_init(channel,pri_ctrl,sec_ctrl,src_addr,dst_addr,trans_ctr)
- dma_reset( )

Operation mode macros are as follows:

- DMA_AUTO_START(chan)
- DMA_PAUSE(chan)
- DMA_RSYNC_CLR(chan)
- DMA_RSYNC_SET(chan)
- DMA_START(chan)
- DMA_STOP(chan)
- DMA_WSYNC_CLR(chan)
- DMA_WSYNC_SET(chan)

The dma.h file also provides a number of macros that may be used to obtain the memory-mapped address of a DMA register, based upon a given channel number. These macros are:

- DMA_DEST_ADDR_ADDR(chan)
- DMA_PRIMARY_CTRL_ADDR(chan)
- DMA_SECONDARY_CTRL_ADDR(chan)
- DMA_SRC_ADDR_ADDR(chan)
- DMA_XFER_COUNTER_ADDR(chan)

Table 2–8 shows the DMA register definition table.

Table 2–8. DMA Register Definition Table

Register Mnemonic	Register Address Mnemonic
DMA0_PRIMARY_CTRL	DMA0_PRIMARY_CTRL_ADDR
DMA0_SECONDARY_CTRL	DMA0_SECONDARY_CTRL_ADDR
DMA0_SRC_ADDR	DMA0_SRC_ADDR_ADDR
DMA0_DEST_ADDR	DMA0_DEST_ADDR_ADDR
DMA0_XFER_COUNTER	DMA0_XFER_COUNTER_ADDR
DMA1_PRIMARY_CTRL	DMA1_PRIMARY_CTRL_ADDR
DMA1_SECONDARY_CTRL	DMA1_SECONDARY_CTRL_ADDR
DMA1_SRC_ADDR	DMA1_SRC_ADDR_ADDR
DMA1_DEST_ADDR	DMA1_DEST_ADDR_ADDR
DMA1_XFER_COUNTER	DMA1_XFER_COUNTER_ADDR
DMA2_PRIMARY_CTRL	DMA2_PRIMARY_CTRL_ADDR
DMA2_SECONDARY_CTRL	DMA2_SECONDARY_CTRL_ADDR
DMA2_SRC_ADDR	DMA2_SRC_ADDR_ADDR
DMA2_DEST_ADDR	DMA2_DEST_ADDR_ADDR
DMA2_XEFR_COUNTER	DMA2_XFER_COUNTER_ADDR
DMA3_PRIMARY_CTRL	DMA3_PRIMARY_CTRL_ADDR
DMA3_SECONDARY_CTRL	DMA3_SECONDARY_CTRL_ADDR
DMA3_SRC_ADDR	DMA3_SRC_ADDR_ADDR
DMA3_DEST_ADDR	DMA3_DEST_ADDR_ADDR
DMA3_XFER_COUNTER	DMA3_XFER_COUNTER_ADDR
DMA_GCR_A	DMA_GCR_A
DMA_GCR_B	DMA_GCR_B_ADDR
DMA_GNDX_A	DMA_GNDX_A_ADDR
DMA_GNDX_B	DMA_GNDX_B_ADDR
DMA_GADDR_A	DMA_GADDR_A_ADDR
DMA_GADDR_B	DMA_GADDR_B_ADDR
DMA_GADDR_C	DMA_GADDR_C_ADDR
DMA_GADDR_D	DMA_GADDR_D_ADDR
DMA_GCTRL	DMA_GCTRL_ADDR

The dma.h file provides macro defines indicating the bit and bit-relative positions for the DMA registers. These are used as arguments to the dma.h and

regs.h macros. These are listed in Table 2–9 through Table 2–15, according to the DMA register to which they belong.

Table 2–9. DMA Primary Control Register Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
START	0	START_SZ	2
STATUS	2	STATUS_SZ	2
SRC_DIR	4	SRC_DIR_SZ	2
DST_DIR	6	DST_DIR_SZ	2
ESIZE	8	ESIZE_SZ	2
SPLIT	10	SPLIT_SZ	2
CNT_RELOAD	12	—	1
INDEX	13	—	1
RSYNC	14	RSYNC_SZ	5
WSYNC	19	WSYNC_SZ	5
PRI	24	—	1
TCINT	25	—	1
FS	26	—	1
EMOD	27	—	1
SRC_RELOAD	28	SRC_RELOAD_SZ	2
DST_RELOAD	30	DST_RELOAD_SZ	2

Table 2–10. DMA Secondary Control Register Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
SX_COND	0	—	1
SX_IE	1	—	1
FRAME_COND	2	—	1
FRAME_IE	3	—	1
LAST_COND	4	—	1
LAST_IE	5	—	1
BLOCK_COND	6	—	1
BLOCK_IE	7	—	1
RDROP_COND	8	—	1
RDOPR_IE	9	—	1
WDROP_COND	10	—	1
WDROP_IE	11	—	1
RSYNC_STAT	12	—	1
RSYNC_CLR	13	—	1
WSYNC_STAT	14	—	1
WSYNC_CLR	15	—	1
DMAC_EN	16	DMAC_EN_SZ	3

Table 2–11. DMA Channel Transfer Counter Register Bits and Bit-Relative Position

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
ELEMENT_COUNT	0	ELEMENT_COUNT_SZ	16
FRAME_COUNT	16	FRAME_COUNT_SZ	16

Table 2–12. DMA Global Count Reload Register Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
ELEMENT_COUNT_ RELOAD	0	ELEMENT_COUNT_ RELOAD_SZ	16
FRAME_COUNT_ RELOAD	16	FRAME_COUNT_ RELOAD_SZ	16

Table 2–13. DMA Global Index Register Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
ELEMENT_INDEX	0	ELEMENT_INDEX_SZ	16
FRAME_INDEX	16	FRAME_INDEX_SZ	16

Table 2–14. DMA Global Address Register Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
SPLIT_ADDRESS	3	SPLIT_ADDRESS_SZ	29

Table 2–15. DMA Auxiliary Control Register Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
CH_PRI	0	CH_PRI_SZ	4
AUXPRI	4		

The dma.h file also provides a set of macro defines that provide the bit-field *values* for the bit-fields defined in Table 2–9 through Table 2–15. These defines are shown in Table 2–16.

Table 2–16. DMA Channel Primary Control Register Bits and Possible Values

(a) START field

Mnemonic	Possible Value
DMA_STOP_VAL	0
DMA_PAUSE_VAL	2
DMA_START_VAL	1
DMA_AUTO_START_VAL	3

(b) Source/destination address modification after element transfers (SRC_DIR, DST_DIR)

Mnemonic	Possible Value
DMA_ADDR_NO_MOD	0
DMA_ADDR_DEC	2
DMA_ADDR_INC	1
DMA_ADDR_INDX	3

(c) Read and write DMA synchronization event numbers (RSYNC, WSYNC)

Mnemonic	Possible Value
SEN_NONE	0
SEN_TINT0	1
SEN_TINT1	2
SEN_SD_INT	3
SEN_EXT_INT4	4
SEN_EXT_INT5	5
SEN_EXT_INT6	6
SEN_EXT_INT7	7
SEN_DMA_INT0	8
SEN_DMA_INT1	9
SEN_DMA_INT2	10
SEN_DMA_INT3	11
SEN_XEVT0	12
SEN_REVT0	13
SEN_XEVT1	14
SEN_REVT1	15
SEN_DSPINT	16

Table 2–16. DMA Channel Primary Control Register Bits and Possible Values (Continued)

(d) Element size defines (ESIZE)

Mnemonic	Possible Value
DMA_ESIZE32	0
DMA_ESIZE16	1
DMA_ESIZE8	2

(e) Priority field defines (PRI)

Mnemonic	Possible Value
DMA_CPU_PRI	0
DMA_DMA_PRI	1

(f) Split mode defines (SPLIT)

Mnemonic	Possible Value
DMA_SPLIT_DIS	0
DMA_SPLIT_GARA	1
DMA_SPLIT_GARB	2
DMA_SPLIT_GARC	3

(g) DMA channel transfer counter reload for autoinitialization and multiframe transfers (CNT_RELOAD)

Mnemonic	Possible Value
DMA_CNT_RELOADA	0
DMA_CNT_RELOADB	1

(h) DMA global data register to use as a programmable index (INDEX)

Mnemonic	Possible Value
DMA_INDX2	0
DMA_INDX3	1

Table 2–16. DMA Channel Primary Control Register Bits and Possible Values (Continued)

(i) Emulation mode (EMOD)

Mnemonic	Possible Value
DMA_NO_EM_HALT	0
DMA_EM_HALT	1

(j) DMA channel source/destination address reload for autoinitialization (SRC_RELOAD, DST_RELOAD)

Mnemonic	Possible Value
DMA_RELOAD_NONE	0
DMA_RELOAD_GARB	1
DMA_RELOAD_GARC	2
DMA_RELOAD_GARD	3

(k) DMA channel EN pin control (DMAC_EN)

Mnemonic	Possible Value
DMAC_LO	0
DMAC_HI	1
DMAC_RSYNC_STAT	2
DMAC_WSYNC_STAT	3
DMAC_FRAME_COND	4
DMAC_BLOCK_COND	5

The following example shows the use of macros, functions, and defines in the direct memory access support header file:

Example: The following code was taken from the implementation of a ’C6x multichannel buffered serial port (McBSP) driver that uses the ’C6x peripheral control library. This example sets up the indicated DMA channel for a block transfer to the McBSP data transmit register (DXR) from an initialized memory buffer. An integer pointer to this buffer, p_buffer, is assumed to have been passed into this function as an argument. The buffer size is indicated by num_words.

```
unsigned int dma_pri_ctrl= 0;
unsigned int dma_sec_ctrl= 0;
unsigned int dma_src_addr= 0;
unsigned int dma_dst_addr= 0;
unsigned int dma_tcnt    = 0;

unsigned int num_frames;

/* configure dma primary control register */
LOAD_FIELD(&dma_pri_ctrl,DMA_ADDR_INC,SRC_DIR,SRC_DIR_SZ);
LOAD_FIELD(&dma_pri_ctrl,SEN_XEVT0,WSYNC,WSYNC_SZ);
SET_BIT(&dma_pri_ctrl,TCINT);

/* configure dma secondary control register */
SET_BIT(&dma_sec_ctrl,BLOCK_IE);

/* configure transfer counter */
num_frames= 1;

LOAD_FIELD(&dma_tcnt,num_frames,FRAME_COUNT,FRAME_COUNT_SZ);
LOAD_FIELD(&dma_tcnt,num_words,ELEMENT_COUNT,ELEMENT_COUNT_SZ);

/* configure source address (supplied by caller)   */
dma_src_addr = (unsigned int)(p_buffer);

/* configure destination address */
dma_dst_addr = MCBSP_DXR_ADDR(dev->port);

/* Write to DMA channel 0 configuration registers */
dma_init( DMA_CH0,

      dma_pri_ctrl,
      dma_sec_ctrl,
      dma_src_addr,
      dma_dst_addr,
      dma_tcnt);
```

After the configuration is complete, you may start the indicated DMA channel with the following:

```
DMA_START(DMA_CH0);
```

2.2.4 External Memory Interface Support (emif.h, emif.c)

The EMIF module of the device library provides a function and macros to control the external memory interface on the ‘C6x.

The following function, emif_init, is an initialization routine that configures the entire EMIF based upon given values:

- emif_init(g_ctrl,ce0_ctrl,ce1_ctrl,ce2_ctrl,ce3_ctrl,sdram_ctrl, sdram_refresh)

The following macros provided in emif.h control the SDRAM refresh period, enable and disable SDRAM refresh, and initialize the SDRAM in each CE space:

- EMIF_GET_MAP_MODE( )
- SDRAM_INIT( )
- SDRAM_REFRESH_DISABLE( )
- SDRAM_REFRESH_ENABLE( )
- SDRAM_REFRESH_PERIOD(val)

Table 2–17 shows the EMIF register definition table.

Table 2–17. EMIF Register Definition Table

Register Mnemonic	Register Address Mnemonic
EMIF_GCTRL	EMIF_GCTRL_ADDR
EMIF_CE0_CTRL	EMIF_CE0_CTRL_ADDR
EMIF_CE1_CTRL	EMIF_CE1_CTRL_ADDR
EMIF_CE2_CTRL	EMIF_CE2_CTRL_ADDR
EMIF_CE3_CTRL	EMIF_CE3_CTRL_ADDR
EMIF_SDRAM_CTRL	EMIF_SDRAM_CTRL_ADDR
EMIF_SDRAM_REF	EMIF_SDRAM_REF_ADDR

Table 2–18 through Table 2–21 show the macro defines that name the bits and bit-relative positions for EMIF registers.

Table 2–18. EMIF Global Control Register Bits and Bit-Relative Positions

Bit Field	Relative Position
MAP	0
RBTR8	1
SSCRT	2
CLK2EN	3
CLK1EN	4
SSCEN	5
SDCEN	6
NOHOLD	7
HOLDA	8
HOLD	9
ARDY	10
CLK2INV	12
SDCINV	13

Table 2–19. EMIF CE0/1/2/3 Control Register Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
READ_HOLD	0	READ_HOLD_SZ	2
MTYPE	4	MTYPE_SZ	3
READ_STROBE	8	READ_STROBE_SZ	6
TA	14	TA_SZ	2
READ_SETUP	16	READ_SETUP_SZ	4
WRITE_HOLD	20	WRITE_HOLD_SZ	2
WRITE_STROBE	22	WRITE_STROBE_SZ	6
WRITE_SETUP	28	WRITE_SETUP_SZ	4

Table 2–20. EMIF SDRAM Control Register Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
TRC	12	TRC_SZ	4
TRP	16	TRP_SZ	4
TRCD	20	TRCD_SZ	4
INIT	24	—	1
RFEN	25	—	1
SDWID	26	—	1

Table 2–21. EMIF SDRAM Timing Register Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
PERIOD	0	PERIOD_SZ	12
COUNTER	12	COUNTER_SZ	12

Table 2–22 provides values for the MTYPE bit fields.

Table 2–22. EMIF CE Space Control Register Memory Type (MTYPE) Bit-Field Values

Mnemonic	Possible Value
MTYPE_8ROM	0
MTYPE_16ROM	1
MTYPE_32ASYNC	2
MTYPE_32SDRAM	3
MTYPE_32SBSRAM	4

The following examples show the use of macros, functions, and defines in the external memory interface support header file:

Example1: The following code was taken from the board support library for the ’C6x evaluation module (EVM) and demonstrates using the emif_init( ) function in the EMIF support files emif.c and emif.h. This code can be found on the CD accompanying the EVM. This example configures the EMIF with default values that operate the board at any selected clock rate.

```
/* RBTR8 preemption, SBSRAM at 1/2, clk1&2 disable, hold enabled, no clock inv */
#define DEFAULT_EMIF_GCTRL         0x00003060

/* CE0 space SBSRAM, all other field are dont cares                            */
#define DEFAULT_EMIF_CE0_CTRL      0x00000040

/* CE1 space async expansion and CODEC, holds setups and strobes maximum val   */
#define DEFAULT_EMIF_CE1_CTRL      0x30f30323

/* CE2,CE3 space SDRAM, all other fields are dont cares                        */
#define DEFAULT_EMIF_CE2_CTRL      0x00000030
#define DEFAULT_EMIF_CE3_CTRL      0x00000030

/* SDRAM, default TRC TRP TRCD, init SDRAM, refresh enable, 16 bit devices     */
#define DEFAULT_EMIF_SDRAM_CTRL    0x07229000   /*                             */

/* SDRAM default refresh period                                                */
#define DEFAULT_EMIF_SDRAM_REF     0x00000619

emif_init(DEFAULT_EVM_EMIF_GCTRL,
          DEFAULT_EVM_EMIF_CE0_CTRL,
          DEFAULT_EVM_EMIF_CE1_CTRL,
          DEFAULT_EVM_EMIF_CE2_CTRL,
          DEFAULT_EVM_EMIF_CE3_CTRL,
          DEFAULT_EVM_EMIF_SDRAM_CTRL,
          DEFAULT_EVM_EMIF_SDRAM_REF);
```

Example 2: Bit 0 of the EMIF global control register is the MAP bit that indicates the current map mode of the ’C6x. This may be determined by using the macro EMIF_GET_MAP_MODE as follows:

```
if (EMIF_GET_MAP_MODE( ))
  printf(“Map mode 1\n”);
else
  printf(“Map mode 0\n”);
```

2.2.5 Host Port Interface Support (hpi.h)

The hpi.h file provides macro support for the ‘C6x side of the host port interface. Macros are provided to generate an interrupt to the host, reset the interrupt flag generated by the host, and fetch the state of the host and DSP interrupts. These macros are:

- HPI_GET_DSPINT( )
- HPI_GET_HINT( )
- HPI_RESET_DSPINT( )
- HPI_SET_HINT( )

Table 2–23 shows defines for the HPI control (HPIC) register bits.

Table 2–23. HPIC Bits and Bit-Relative Positions

Bit Field	Relative Position
HWOB	0
DSPINT	1
HINT	2
HRDY	3
FETCH	4

The address of the HPIC register is also provided as a define in Table 2–24.

Table 2–24. HPIC Register Address

Register Mnemonic	Register Address Mnemonic
HPIC	HPIC_ADDR

The following examples show the use of macros, functions, and defines in the host port interface support header file:

Example 1: The HPI support macros set or retrieve bits within the HPIC register and require no arguments. To reset a DSP interrupt generated from the host, call:

```
HPI_RESET_DSPINT( );
```

Example 2: To generate a host interrupt, call:

```
HPI_SET_HINT( );
```

2.2.6 Interrupt Support (intr.h, intr.c, intr_.asm)

The interrupt module, which consists of the intr.h, intr.c and intr_.asm files, provides support for interrupts on the ‘C6x. The intr.h file contains defines for CPU interrupt numbers, interrupt selection numbers, and default interrupt selector values, as well as macro functions that manipulate interrupt-related bits within memory-mapped and non-memory-mapped registers.

The intr.c file provides subroutines that initialize interrupt processing, allow dynamic interrupt hooking, and control the interrupt selector registers. Interrupt processing is initialized by setting the interrupt service table pointer (ISTP) to the address of the IST. The contents of the IST is provided in intr_.asm and its location in memory is determined by the linker command file. Each interrupt service fetch packet (ISFP) contains code that looks up the address of its corresponding ISR from the isr_jump_table, which is defined in intr.c. If this location is “unhooked“ (indicated by a value of 0), no ISR is called and the ISFP simply returns from interrupt. If a non-zero value is found in the isr_jump_table, a branch to this location is executed. The intr_hook() routine is used to place the address of an indicated ISR in the isr_jump_table at the location corresponding to the indicated CPU interrupt number.

Interrupt processing functions include the following:

- ❏ intr_get_cpu_intr(int isn)
- ❏ intr_hook(void(*fp)(void),cpu_intr)
- ❏ intr_init(void)
- ❏ intr_isn(int cpu_intr)
- ❏ intr_map(int cpu_intr,int isn)

The value *fp is a function pointer to the user supplied ISR, cpu_intr refers to the CPU interrupt number, and isn refers to the interrupt selection number that specifies the interrupt source to map to a given CPU interrupt.

In the following list of interrupt processing macros, bit refers to the bit position on which to operate, val refers to the field value, and sel is 0 for the low interrupt selector register and non-zero for the high interrupt selector register.

- INTR_CHECK_FLAG(bit)
- INTR_CLR_FLAG(bit)
- INTR_DISABLE(bit)
- INTR_ENABLE(bit)
- INTR_EXT_POLARITY(bit,val)
- INTR_GET_ISN(intr,sel)
- INTR_GLOBAL_DISABLE
- INTR_GLOBAL_ENABLE
- INTR_MAP_RESET( )
- INTR_RETURN_ISN(intr,sel)
- INTR_SET_FLAG(bit)
- INTR_SET_MAP(intr,val,sel)

Table 2–25 shows the macro defines for the three interrupt selector registers in intr.h.

Table 2–25. Interrupt Select Register Addresses

Register Name	Address
INTR_MULTIPLEX_HIGH_ADDR	0x019c0000
INTR_MULTIPLEX_LOW_ADDR	0x019c0004
EXTERNAL_INTR_POL_ADDR	0x019c0008

Table 2–26 through Table 2–28 show the macro defines provided by intr.h, listed according to the register to which the defines belong.

Table 2–26. Interrupt Multiplexer Low Register Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
INTSEL4	0	INTSEL_SZ	4
INTSEL5	5	INTSEL_SZ	4
INTSEL6	10	INTSEL_SZ	4
INTSEL7	16	INTSEL_SZ	4
INTSEL8	21	INTSEL_SZ	4
INTSEL9	26	INTSEL_SZ	4

Table 2–27. Interrupt Multiplexer High Register Bits and Bit-Relative Positions

Bit Field	Relative Position
INTSEL10	0
INTSEL11	5
INTSEL12	10
INTSEL13	16
INTSEL14	21
INTSEL15	26

Table 2–28. External Interrupt Polarity Register Bits and Bit-Relative Positions

Bit Field	Relative Position
XIP4	0
XIP5	1
XIP6	2
XIP7	3

Table 2–29 and Table 2–30 show the macro defines provided by intr.h that name the CPU interrupt numbers and interrupt selection numbers. The interrupt selection values are used as the interrupt-selector-low and -high register values.

Table 2–29. CPU Interrupt Numbers

Mnemonic	Value
CPU_INT_RST	0
CPU_INT_NMI	1
CPU_INT_RSV1	2
CPU_INT_RSV2	3
CPU_INT4	4
CPU_INT5	5
CPU_INT6	6
CPU_INT7	7
CPU_INT8	8

Table 2–29. CPU Interrupt Numbers (Continued)

Mnemonic	Value
CPU_INT9	9
CPU_INT10	10
CPU_INT11	11
CPU_INT12	12
CPU_INT13	13
CPU_INT14	14
CPU_INT15	15

Table 2–30. Interrupt Selection Numbers

Mnemonic	Value
ISN_DSPINT	0
ISN_TINT0	1
ISN_TINT1	2
ISN_SD_INT	3
ISN_EXT_INT4	4
ISN_EXT_INT5	5
ISN_EXT_INT6	6
ISN_EXT_INT7	7
ISN_DMA_INT0	8
ISN_DMA_INT1	9
ISN_DMA_INT2	10
ISN_DMA_INT3	11
ISN_XINT0	12
ISN_RINT0	13
ISN_XINT1	14
ISN_RINT1	15

The following examples show how an interrupt service routine (ISR) can be hooked to a given interrupt.

Example 1: The interrupt source needs to be mapped to a CPU interrupt. This is accomplished by loading the interrupt selection number into the desired interrupt selection field in the interrupt multiplexer register (see the *TMS320C6201/6701 Peripherals Reference Guide* for more information). To map the DMA channel 0 interrupt source to CPU interrupt 8 (INT8), use the intr_map function as follows:

```
intr_map(CPU_INT8,ISN_DMA_INT0);
```

Example 2: Once the interrupt multiplexer register is configured, the ISR can be hooked to the CPU interrupt and enabled as follows:

```
interrupt void exampleISR(void)
{
  isrFlag= TRUE;
  return;
}
intr_hook( exampleISR, CPU_INT8 );
INTR_ENABLE( CPU_INT8 );
```

A DMA channel 0 interrupt event now causes the example ISR to be invoked.

2.2.7 Multichannel Buffered Serial Port Support (mcbsp.h, mcbsp.c)

The mcbsp.h and mcbsp.c files contain macros and one function that control the multichannel buffered serial port registers on the ‘C6x. Four groups of macros exist to control the operation of the indicated channel. The first group enables and disables functionality on the port. The second group is used to reset indicated portions of the McBSP. The third group is used during data transfer to start, stop, and receive status of the indicated port. The fourth group of macros returns the address of a particular McBSP register, based upon a given port number.

The McBSP enable and disable macros are as follows:

- MCBSP_ENABLE(port_no,type)
- MCBSP_FRAME_SYNC_ENABLE(port_no)
- MCBSP_IO_DISABLE(port_no)
- MCBSP_IO_ENABLE(port_no)
- MCBSP_LOOPBACK_DISABLE(port_no)
- MCBSP_LOOPBACK_ENABLE(port_no)
- MCBSP_SAMPLE_RATE_ENABLE(port_no)

McBSP reset and initialization macros and function are as follows:

- Macros:
  - MCBSP_FRAME_SYNC_RESET(port_no)
  - MCBSP_RX_RESET(port_no)
  - MCBSP_SAMPLE_RATE_RESET(port_no)
  - MCBSP_TX_RESET(port_no)
- Function:
  - mcbsp_init(port_no,spcr_ctrl,rcr_ctrl,xcr_ctrl,srgr_ctrl,mcr_ctrl, rcer_ctrl,xcer_ctrl,pcr_ctrl)

McBSP data transfer macros follow:

- MCBSP_ADDR(port_no)
- MCBSP_BYTES_PER_WORD(wdlen)
- MCBSP_READ(port_no)
- MCBSP_RRDY(port_no)
- MCBSP_WRITE(port_no)
- MCBSP_XRDY(port_no)

McBSP register address macros are as follows:

- MCBSP_DRR_ADDR(port_no)
- MCBSP_DXR_ADDR(port_no)
- MCBSP_MCR_ADDR(port_no)
- MCBSP_PCR_ADDR(port_no)
- MCBSP_RCER_ADDR(port_no)
- MCBSP_RCR_ADDR(port_no)
- MCBSP_SPCR_ADDR(port_no)
- MCBSP_SRGR_ADDR(port_no)
- MCBSP_XCER_ADDR(port_no)
- MCBSP_XCR_ADDR(port_no)

Table 2–31 shows the McBSP register definition table. McBSP register addresses may be obtained by using the register address macros shown in Table 3–7 on page 3-6.

Table 2–31. McBSP Register Definitions

Register Name
MCBSP0_DRR
MCBSP0_DXR
MCBSP0_SPCR
MCBSP0_RCR
MCBSP0_XCR
MCBSP0_SRGR
MCBSP0_MCR
MCBSP0_RCER
MCBSP0_XCER
MCBSP0_PCR
MCBSP1_DRR
MCBSP1_DXR
MCBSP1_SPCR
MCBSP1_RCR
MCBSP1_XCR
MCBSP1_SRGR
MCBSP1_MCR
MCBSP1_RCER
MCBSP1_XCER
MCBSP1_PCR

Table 2–32 through Table 2–38 show the McBSP macro defines that indicate the bits and bit-relative positions for the McBSP memory-mapped registers.

Table 2–32. McBSP Control Register Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
RRST	0	—	1
RRDY	1	—	1
RFULL	2	—	1
RSYNC_ERR	3	—	1
RINTM	4	RINTM_SZ	2
CLKSTP	10	CLKSTP_SZ	3
RJUST	13	RJUST_SZ	2
DLB	15	—	1
XRST	16	—	1
XRDY	17	—	1
XEMPTY	18	—	1
XSYNC_ERR	19	—	1
XINTM	20	XINTM_SZ	2
GRST	22	—	1
FRST	23	—	1

Table 2–33. McBSP Pin Control Register Bits and Bit-Relative Positions

Bit Field	Relative Position
CLKRP	0
CLKXP	1
FSRP	2
FSXP	3
DR_STAT	4
DX_STAT	5
CLKS_STAT	6
CLKRM	8
CLKXM	9
FSRM	10
FSXM	11
RIOEN	12
XIOEN	13

Table 2–34. McBPS Receive and Transmit Register Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
RWDLEN1	5	RWDLEN1_SZ	3
RFRLEN1	8	RFRLEN1_SZ	7
RDATDLY	16	RDATDLY_SZ	2
RFIG	18	—	1
RCOMPAND	19	RCOMPAND_SZ	2
RWDLEN2	21	RWDLEN2_SZ	3
RFRLEN2	24	RFRLEN2_SZ	7
RPHASE	31	—	1
XWDLEN1	5	XWDLEN1_SZ	3
XFRLEN1	8	XFRLEN1_SZ	7
XDATDLY	16	XDATDLY_SZ	2
XFIG	18	—	1
XCOMPAND	19	XCOMPAND_SZ	2
XWDLEN2	21	XWDLEN2_SZ	3
XFRLEN2	24	XFRLEN2_SZ	7
XPHASE	31	—	1

Table 2–35. McBSP Sample Rate Generator Register Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
CLKGDV	0	CLKGDV_SZ	8
FWID	8	FWID_SZ	8
FPER	16	FPER_SZ	12
FSGM	28	—	1
CLKSM	29	—	1
CLKSP	30	—	1
GSYNC	31	—	1

Table 2–36. McBSP Multichannel Control Register Bits and Bit-Relative Positions

Bit Field	Relative Position	Bit Field Length Mnemonic	Bit Field Length
RMCM	0	—	1
RCBLK	2	RCBLK_SZ	3
RPABLK	5	RPABLK_SZ	2
RPBBLK	7	RPBBLK_SZ	2
XMCM	16	XMCM_SZ	2
XCBLK	18	XCBLK_SZ	3
XPABLK	21	XPABLK_SZ	2
XPBBLK	23	XPBBLK_SZ	2

Table 2–37. McBSP Receive Enable Register Bits and Bit-Relative Positions

Bit Field	Relative Position
RCEA0	0
RCEA1	1
RCEA2	2
RCEA3	3
RCEA4	4
RCEA5	5
RCEA6	6
RCEA7	7
RCEA8	8
RCEA9	9
RCEA10	10
RCEA11	11
RCEA12	12
RCEA13	13
RCEA14	14
RCEA15	15
RCEB0	16
RCEB1	17
RCEB2	18
RCEB3	19
RCEB4	20
RCEB5	21
RCEB6	22
RCEB7	23
RCEB8	24
RCEB9	25
RCEB10	26
RCEB11	27
RCEB12	28
RCEB13	29
RCEB14	30
RCEB15	31

Table 2–38. McBSP Transmit Enable Register Bits and Bit-Relative Positions

Bit Field	Relative Position
XCEA0	0
XCEA1	1
XCEA2	2
XCEA3	3
XCEA4	4
XCEA5	5
XCEA6	6
XCEA7	7
XCEA8	8
XCEA9	9
XCEA10	10
XCEA11	11
XCEA12	12
XCEA13	13
XCEA14	14
XCEA15	15
XCEB0	16
XCEB1	17
XCEB2	18
XCEB3	19
XCEB4	20
XCEB5	21
XCEB6	22
XCEB7	23
XCEB8	24
XCEB9	25
XCEB10	26
XCEB11	27
XCEB12	28
XCEB13	29
XCEB14	30
XCEB15	31

Table 2–39 through Table 2–41 shows the macro defines that indicate the bits and possible values for all McBSP specific registers.

Table 2–39. Serial Port Control Register (SPCR) Bits and Possible Values

(a) Transmit/receive mode (XINTM, RINTM)

Mnemonic	Possible Value
INTM_RDY	0x00
INTM_BLOCK	0x01
INTM_FRAME	0x02
INTM_SYNCERR	0x03

(b) Digital loopback mode (DLB)

Mnemonic	Possible Value
DLB_ENABLE	0x01
DLB_DISABLE	0x00

(c) Sign extension and justification (RJUST)

Mnemonic	Possible Value
RXJUST_RJZF	0x00
RXJUST_RJSE	0x01
RXJUST_LJZF	0x02

Table 2–40. Pin Control Register (PCR) Bits and Possible Values

(a) Clock polarity CLKRP bit

Mnemonic	Possible Value
CLKR_POL_RISING	0x01
CLKR_POL_FALLING	0x00

(b) Clock polarity CLKXP bit

Mnemonic	Possible Value
CLKX_POL_RISING	0x00
CLKX_POL_FALLING	0x01

(c) Transmit and receive frame sync polarity (FSXP, FSRP)

Mnemonic	Possible Value
FSYNC_POL_HIGH	0x00
FSYNC_POL_LOW	0x01

Table 2–40. Pin Control Register (PCR) Bits and Possible Values (Continued)

(d) Transmit and receive clock mode (CLKXM, CLKRM)

Mnemonic	Possible Value
CLK_MODE_EXT	0x00
CLK_MODE_INT	0x01

(e) Transmit and receive frame sync mode (FSXM, FSRM)

Mnemonic	Possible Value
FSYNC_MODE_EXT	0x00
FSYNC_MODE_INT	0x01

Table 2–41. Transmit Receive Control Register (XCR/RCR) Bits and Possible Values

(a) Transmit and receive phase mode (XPHASE, RPHASE)

Mnemonic	Possible Value
SINGLE_PHASE	0x00
DUAL_PHASE	0x01

(b) Transmit and receive frame length (XFRLEN1, XFRLEN2, RFRLEN1, RFRLEN2)

Mnemonic	Possible Value
MAX_FRAME_LENGTH	0x7F

(c) Transmit and receive word length (XWDLEN1, XWDLEN2, RWDLEN1, RWDLEN2)

Mnemonic	Possible Value
WORD_LENGTH_8	0x00
WORD_LENGTH_12	0x01
WORD_LENGTH_16	0x02
WORD_LENGTH_20	0x03
WORD_LENGTH_24	0x04
WORD_LENGTH_32	0x05
MAX_WORD_LENGTH	0x05

Table 2–41. Transmit Receive Control Register (XCR/RCR) Bits and Possible Values (Continued)

(d) Transmit and receive compand mode (XCOMPAND, RCOMPAND)

Mnemonic	Possible Value
NO_COMPAND_MSB_1ST	0x00
NO_COMPAND_LSB_1ST	0x01
COMPAND_ULAW	0x02
COMPAND_ALAW	0x03

(e) Transmit and receive frame sync ignore bit (XFIG, RFIG)

Mnemonic	Possible Value
FRAME_IGNORE	0x01
NO_FRAME_IGNORE	0x00

(f) Transmit and receive delay mode field (XDATDLY, RDATDLY)

Mnemonic	Possible Value
DATA_DELAY0	0x00
DATA_DELAY1	0x01
DATA_DELAY2	0x02

Table 2–42. Sample Rate Generator Register (SRGR) Bits and Possible Values

(a) SRGR clock rate divide (CLKGDV)

Mnemonic	Possible Value
MAX_SRG_CLK_DIV	0xFF

(b) SRGR frame width (FWID)

Mnemonic	Possible Value
MAX_FRAME_WIDTH	0xFF

(c) SRGR frame period (FPER)

Mnemonic	Possible Value
MAX_FRAME_PERIOD	0x0FFF

Table 2–42. Sample Rate Generator Register (SRGR) Bits and Possible Values (Continued)

(d) SRGR frame sync mode (FSGM)

Mnemonic	Possible Value
FSX_DXR_TO_XSR	0x00
FSX_FSG	0x01

(e) SRGR clock polarity (CLKSP)

Mnemonic	Possible Value
CLKS_POL_FALLING	0x00
CLKS_POL_RISING	0x01

(f) SRGR clock sync bit (GSYNC)

Mnemonic	Possible Value
GSYNC_OFF	0x00
GSYNC_ON	0x01

These macros and defines may be used as the basis for a high-level McBSP driver routine. For more information, see the *TMS320C6x Evaluation Module Reference Guide.*

Example: The following example shows how to receive a buffer of data by polling the RRDY bit of the serial port control register. The code was taken from the McBSP driver supplied with the TMS320C6x EVM, and can be found on the CD that accompanies the board.

```
MCBSP_ENABLE(0,MCBSP_RX);

/* Enable sample rate generator internal frame sync (if needed)   */
if {frame_sync_enable)
{
 MCBSP_FRAME_SYNC_ENABLE(frame_sync_dev–>port);
}
/* Enter receive loop, polling RRDY for data             */
bytes_read = 0;

while (bytes_read < num_bytes)
{
 while (!(MCBSP_RRDY(dev–>port)))
 { }

 drr = MCBSP_READ(dev–>port);
 memcpy(p_buffer, (unsigned char *)&drr, bytes_per_word);
 p_buffer += bytes_per_word;
 bytes_read += bytes_per_word;
}
```

2.2.8 Timer Support (timer.h, timer.c)

The timer.h and timer.c files contain macros and a function that control the timer registers on the ‘C6x. They contains macros that start, stop, and resume timer operation. They also contains macros that control the TINP and TOUT as general-purpose I/O pins. See the *TMS320C6x Evaluation Module Reference Guide* for examples using these macros as a higher-level timer driver. These macros and functions are as follows:

- Macros:
  - TIMER_AVAILABLE(chan)
  - TIMER_CLK_EXTERNAL(chan)
  - TIMER_CLK_INTERNAL(chan)
  - TIMER_GET_COUNT(chan)
  - TIMER_GET_PERIOD(chan)
  - TIMER_GET_TSTAT(chan)
  - TIMER_INIT(chan,ctrl,per,cnt)
  - TIMER_MODE_SELECT(chan,mode)
  - TIMER_READ(chan)
  - TIMER_RESET(chan)
  - TIMER_RESUME(chan)
  - TIMER_SET_COUNT(chan,val)
  - TIMER_SET_PERIOD(chan,val)
  - TIMER_START(chan)
  - TIMER_STOP(chan)
  - TINP_GET(chan)
  - TOUT_ASSERT(chan)
  - TOUT_DISABLE(chan)
  - TOUT_ENABLE(chan)
  - TOUT_NEGATE(chan)
  - TOUT_VAL(chan,val)
- Function:
  - timer_delay(short num_timer_periods)

Like the other peripheral-specific header files, the timer module contains three additional macros that return the address of a particular timer register, based upon the channel number. These macros are:

- TIMER_COUNTER_ADDR(chan)
- TIMER_CTRL_ADDR(chan)
- TIMER_PERIOD_ADDR(chan)

Table 2–43 shows the macro defines for the timer file. Table 2–44 shows the mode values for the timer file. Table 2–45 shows the register definition table for timer.h.

Table 2–43. Timer Register Bits and Bit-Relative Positions

Bit Field	Relative Position
FUNC	0
INVOUT	1
DATOUT	2
DATIN	3
PWID	4
GO	6
HLD	7
C_P	8
CLKSRC	9
INVINP	10
TSTAT	11

Table 2–44. Timer Mode Values

Mode Mnemonic	Value
TIMER_PULSE_MODE	0
TIMER_CLOCK_MODE	1

Table 2–45. Timer Register Definition Table

Register Name	Register Address Mnemonic
TIMER0_CTRL	TIMER0_CTRL_ADDR
TIMER0_PERIOD	TIMER0_PERIOD_ADDR
TIMER0_COUNTER	TIMER0_COUNTER_ADDR
TIMER1_CTRL	TIMER1_CTRL_ADDR
TIMER1_PERIOD	TIMER1_PERIOD_ADDR
TIMER1_COUNTER	TIMER1_COUNTER_ADDR

Example: The following example illustrates the use of the timer support macros to implement a delay routine. It pauses the requested number of microseconds before returning to the caller. This routine is found in the board support library supplied with the TMS320C6x EVM.

```
int delay_usec(short numUsec)
{
  unsigned int period_reg;
  unsigned int ctrl_reg = 0;
  int             chan = 0;
  int             cpu_freqInMhz;

  if (!TIMER_AVAILABLE(chan))
  {
    chan++;
    if (!TIMER_AVAILABLE(chan))
      return(–1);
  }


  cpuFreqInMhz = cpu_freq( )  /* returns board CPU freq in Mhz    */

  if (cpu_freqInMhz == ERROR)
  {
    DEBUG(”ERROR reading CPU frequency\n\n”);
    return(ERROR);
  }

  period_reg = ((cpu_freqInMhz >> 3) * numUsec);
  ctrl_reg = MASK_BIG(C_P) | MASK_BIG(CLKSRC);

  TIMER_INIT(chan,ctrl_reg,period_reg,0);
  TIMER_START(chan);

  /* poll for high–low–high transition                          */
  while (!(TIMER_GET_TSTAT(chan))){NOPS;}       /* TSTAT = 1      */
  while   (TIMER_GET_TSTAT(chan)  {NOPS;}       /* TSTAT = 0      */
  while (!(TIMER_GET_TSTAT(chan))){NOPS;}       /* TSTAT = 1      */

  TIMER_STOP(chan);
  return(0);
}
```

Chapter 3

Macros and Functions Summary

This chapter consists of tables that list all macros and functions found in the ’C6x peripheral support library, listed by the source file in which each appears. The tables provide a concise description of each macro and function and give a page reference to the location in Chapter 4 where each is defined in more detail.

Table 3–1. Macros Defined in regs.h

(a) Memory-mapped register bit-manipulation macros

Function	Description	Page
ASSIGN_BIT_VAL(addr,bit,val)	Sets or clears bit in register at address based upon value	4-2
GET_BIT(addr,bit)	Returns value of bit in register at address	4-17
GET_FIELD(addr,bit,length)	Returns value of bits in register at address	4-18
LOAD_FIELD(addr,val,bit,length)	Assigns bits in register at address to value	4-31
MASK_BIT(bit)	Returns a bit mask for the specified field	4-31
MASK_FIELD(bit,length)	Returns field bit mask	4-32
REG_READ(addr)	Returns value in register at address	4-47
REG_WRITE(addr,val)	Sets register at address to value	4-47
RESET_BIT(addr,bit)	Clears bit in register at address	4-48
RESET_FIELD(addr,bit,length)	Resets/clears bits in register at address	4-48
SET_BIT(addr,bit)	Sets bit in register at address	4-51

(b) Non-memory-mapped register bit-manipulation macros

Function	Definition	Page
GET_REG(reg)	Returns value in register	4-18
GET_REG_BIT(reg,bit)	Returns value of bit in register	4-19
GET_REG_FIELD(reg,bit,length)	Returns value of bits in register	4-19
RESET_REG_BIT(reg,bit)	Resets or clear bit in register	4-49
SET_REG(reg,val)	Sets register to value	4-52
SET_REG_BIT(reg,bit)	Sets bit in register	4-52

Table 3–2. Macros Defined in cache.h

Function	Description	Page
CACHE_BYPASS( )	Bypasses cache and get program data from EMIF	4-2
CACHE_DISABLE( )	Disables program memory cache	4-3
CACHE_ENABLE( )	Enables program memory cache	4-3
CACHE_FLUSH( )	Transitions from disabled to enabled state to clear	4-4
CACHE_FREEZE( )	Freezes cache state. Cache misses do not update	4-4
IDLE( )	Idles processor	4-22

Table 3–3. Macros and Functions Defined in dma.h and dma.c

Function	Description	Page
DMA_AUTO_START(chan)	Begins DMA autoinitialization operation on selected channel	4-5
DMA_DEST_ADDR_ADDR(chan)	Returns selected destination and register address	4-5
dma_global_init(gcr, gcra, gcrb, gndxa, gndxb, gaddra, gaddrb, gaddrc, gaddrd)	Sets registers in parameter list to passed in parameter values	4-6
dma_init(chan,pri_ctrl,sec_ctrl,src_addr, dst_addr,trans_ctr)	Sets registers for selected channel (chan) in argument list to passed in parameter values	4-7
DMA_PAUSE(chan)	Pauses DMA operation on selected channel	4-8
DMA_PRIMARY_CTRL_ADDR(chan)	Returns selected primary ctrl register address	4-9
dma_reset( )	Resets all DMA registers to their default values	4-10
DMA_RSYNC_CLR( )	Clears the read sync bit in the DMA secondary control register, selecting no synchronization	4-10
DMA_RSYNC_SET( )	Sets the read sync bit in the DMA secondary control register, selecting synchronization	4-11
DMA_SECONDARY_CTRL_ADDR(chan)	Returns selected secondary ctrl register address	4-11
DMA_SRC_ADDR_ADDR(chan)	Returns selected source and address register address	4-12
DMA_START(chan)	Begins DMA operation on selected channel	4-12
DMA_STOP(chan)	Stops DMA operation on selected channel	4-13
DMA_WSYNC_CLR( )	Clears the write sync bit in the DMA secondary control register, selecting no synchronization	4-13
DMA_WSYNC_SET( )	Sets the write sync bit in the DMA secondary control register, selecting syncnronization	4-14
DMA_XFER_COUNTER__ADDR(chan)	Returns selected transfer counter address register address	4-14

Table 3–4. Macros and Functions Defined in emif.c and emif.h

Function	Description	Page
EMIF_GET_MAP_MODE( )	Returns value of MAP bit in EMIF global control register	4-50
emif_init(g_ctrl,ce0_ctrl,ce1_ctrl,ce2_ctrl, ce3_ctrl,sdram_ctrl,sdram_refresh)	Sets registers in parameter list to values passed in arguments	4-16
SDRAM_INIT( )	Initializes SDRAM in each CE space configured for SDRAM	4-49
SDRAM_REFRESH_DISABLE( )	Disables SDRAM refresh	4-50
SDRAM_REFRESH_ENABLE( )	Enables SDRAM refresh	4-50
SDRAM_REFRESH_PERIOD(val)	Sets SDRAM refresh period	4-51

Table 3–5. Macros and Functions Defined in hpi.h

Function	Description	Page
HPI_GET_DSPINT( )	Returns value of DSP interrupt	4-20
HPI_GET_HINT( )	Returns value of host interrupt	4-20
HPI_RESET_DSPINT( )	Resets DSP interrupt flag generated by host	4-21
HPI_SET_HINT( )	Generates HPI interrupt to host	4-21

Table 3–6. Macros and Functions Defined in intr.h and intr.c

Function	Description	Page
INTR_CHECK_FLAG(bit)	Returns value of bit in IFR	4-22
INTR_CLR_FLAG(bit)	Manually clears indicated interrupt by writing 1 to ICR	4-23
INTR_DISABLE(bit)	Clears corresponding bit in IER	4-23
INTR_ENABLE(bit)	Sets corresponding bit in IER	4-24
INTR_EXT_POLARITY(bit,val)	Assigns external interrupt polarity. [val = 0 (normal), val = 1 (inverted)]	4-24
intr_get_cpu_intr(int isn)	Returns CPU interrupt number corresponding to given interrupt selection number (isn) found in the interrupt multiplexor registers. ERROR is returned if isn is not found	4-25

Table 3–6. Macros and Functions Defined in intr.h and intr.c (Continued)

Function	Description	Page
INTR_GET_ISN(intr,sel)	Returns interrupt source corresponding to the CPU interrupt specified by intr. Sel is used to select between the low and high interrupt multiplexer registers (0 = low, 1 = high)	4-25
INTR_GLOBAL_DISABLE( )	Globally disables all masked interrupts by clearing the GIE bit	4-26
INTR_GLOBAL_ENABLE( )	Globally enables all masked interrupts by setting the GIE bit	4-26
intr_init(void)	Initializes the ISTP based upon the global vicinity which is resolved at link time.	4-27
intr_isn(int cpu_intr)	Returns interrupt source number corresponding to the CPU interrupt specified by cpu_intr	4-28
intr_hook(void(*fp)(void),cpu_intr)	Places the address of the function in the parameter list in the isr_jump_table at the indicated offset (CPU interrupt number)	4-27
intr_map(int cpu_intr,int isn)	Maps interrupt source (isn) to the indicated CPU interrupt	4-28
INTR_MAP_RESET( )	Resets the interrupt multiplexer maps to their default values	4-29
INTR_RETURN_ISN(intr,sel)	Returns the interrupt source number corresponding to the CPU interrupt specified by intr.	4-29
INTR_SET_FLAG(bit)	Manually sets indicated interrupt by writing to ISR	4-30
INTR_SET_MAP(intr,val,sel)	Maps a CPU interrupt specified by intr to the interrupt source specified by value. Sel is used to select between the low and high interrupt multiplexer registers (0 = low, 1 = high)	4-30

Table 3–7. Macros and Functions Defined in mcbsp.h

Function	Description	Page
MCBSP_ADDR(port)	Returns the base address of the control register block for the specified MCBSP port	4-32
MCBSP_BYTES_PER_WORD(wdlen)	Returns the number of bytes required to hold the number of bits indicated by wdlen	4-33
MCBSP_DRR_ADDR(chan)	Returns selected data receive register address	4-33
MCBSP_DXR_ADDR(chan)	Returns selected data transmit register address	4-34
MCBSP_ENABLE(port_no,type)	Enables operation of the selected channels: transmitter, receiver, or both. (type = 1,2,3; tx_rx_both)	4-34
MCBSP_FRAME_SYNC_ ENABLE(port_no)	Enables generation of frame sync signal for selected port	4-35
MCBSP_FRAME_SYNC_RESET(port_no)	Resets frame sync generation logic for selected port	4-35
mcbsp_init(port_no,spcr_ctrl,rcr_ctrl, xcr_ctrl,srgr_ctrl,mcr_ctrl,rcer_ctrl, xcer_ctrl,pcr_ctrl)	Initializes registers in the parameter list to their given argument values	4-36
MCBSP_IO_DISABLE(port_no)	Takes selected port out of general-purpose I/O mode	4-37
MCBSP_IO_ENABLE(port_no)	Places selected port in general purpose I/O mode	4-37
MCBSP_LOOPBACK_DISABLE(port_no)	Disables loopback mode for selected port	4-38
MCBSP_LOOPBACK_ENABLE(port_no)	Enables loopback mode for selected port	4-38
MCBSP_MCR_ADDR(chan)	Returns selected multichannel control register address	4-39
MCBSP_PCR_ADDR(chan)	Returns selected pin control register address	4-39
MCBSP_RCER_ADDR(chan)	Returns selected receive channel enable register address	4-40
MCBSP_RCR_ADDR(chan)	Returns selected receive control register address	4-40
MCBSP_READ(port_no)	Reads selected channels DRR	4-41
MCBSP_RRDY(port_no)	Returns value of RRDY for selected port	4-41
MCBSP_RX_RESET(port_no)	Resets receive side of serial port	4-42
MCBSP_SAMPLE_RATE_ ENABLE(port_no)	Enables sample rate generator for selected port	4-42
MCBSP_SAMPLE_RATE_ RESET(port_no)	Resets sample rate generator for selected port	4-43
MCBSP_SPCR_ADDR(chan)	Returns selected serial port control register address	4-43

Table 3–7. Macros and Functions Defined in mcbsp.h (Continued)

Function	Description	Page
MCBSP_SRGR_ADDR(chan)	Returns selected sample rate generator register address	4-44
MCBSP_TX_RESET(port_no)	Resets transmit side of serial port	4-44
MCBSP_WRITE(port_no,data)	Writes to selected channels DXR	4-45
MCBSP_XCER_ADDR(chan)	Returns selected transmit channel enable register address	4-45
MCBSP_XCR_ADDR(chan)	Returns selected transmit control register address	4-46
MCBSP_XRDY(port_no)	Returns value of XRDY for selected port	4-46

Table 3–8. Macros and Functions Defined in timer.h and timer.c

Function	Description	Page
TIMER_AVAILABLE(chan)	Checks for availability of the specified timer channel	4-53
TIMER_CLK_EXTERNAL(chan)	Selects timer driven by external clock source for specified channel	4-53
TIMER_CLK_INTERNAL(chan)	Selects the timer driven by the internal clock source for the specified channel	4-54
TIMER_COUNTER_ADDR(chan)	Returns selected timer counter register address	4-54
TIMER_CTRL_ADDR(chan)	Returns selected timer control register address	4-55
timer_delay(num_timer_periods)	Delays for a specified number of timer periods	4-55
TIMER_GET_COUNT(chan)	Returns the value of the timer counter register for the specified channel	4-56
TIMER_GET_PERIOD(chan)	Returns the value of the timer period register for the specified channel	4-56
TIMER_GET_TSTAT(chan)	Returns the value of the timer status bit, TSTAT, in the timer control register of the specified channel	4-57
TIMER_INIT(chan,ctrl,per,cnt)	Initializes timer registers	4-57
TIMER_MODE_SELECT(chan,mode)	Selects between PULSE and CLOCK modes	4-58
TIMER_PERIOD_ADDR(chan)	Returns selected period register address	4-58
TIMER_READ(chan)	Reads value of timer counter register	4-59
TIMER_RESET(chan)	Resets timer to condition defined by device reset	4-59

Table 3–8. Macros and Functions Defined in timer.h and timer.c(Continued)

Function	Description	Page
TIMER_RESUME(chan)	Negates (sets) the HOLD bit to resume counting without resetting the counter register	4-60
TIMER_SET_COUNT(chan,val)	Sets the value of the timer counter register for the specified channel	4-60
TIMER_SET_PERIOD(chan,val)	Sets the value of the timer period register for the specified channel	4-61
TIMER_START(chan)	Sets both GO and HOLD bit in timer control registers, which resets the timer counter register and enables counting on the next clock	4-61
TIMER_STOP(chan)	Asserts (clears) the HOLD bit in the timer control register	4-62
TINP_GET(chan)	Returns value of TINP pin	4-62
TOUT_ASSERT(chan)	Asserts TOUT pin (high).	4-63
TOUT_DISABLE(chan)	Configures TOUT as a timer pin	4-63
TOUT_ENABLE(chan)	Configures TOUT as general-purpose output pin	4-63
TOUT_NEGATE(chan)	Negates TOUT pin (low).	4-64
TOUT_VAL(chan,val)	Assigns val to the TOUT pin when TOUT is configured as a general-purpose output	4-64

Chapter 4

Macros and Functions Description

This chapter provides an alphabetical list of all functions and macros provided in the ’C6x peripheral support library. Each entry gives the complete syntax and description information for the named function or macro, shows where it is defined, and provides a code example. Macros are denoted by uppercase text and functions are denoted by lowercase.

ASSIGN_BIT_VAL

Assign bit to value

Syntax

#include <regs.h>
#define **ASSIGN_BIT_VAL(**addr,bit,val**)**

Defined in

regs.h as a macro

Description

ASSIGN_BIT_VAL sets or clears the indicated bit in the register or memory location specified. It uses the following parameters:

- addr: address of register or memory location
- val: value to assign bit field (val = 0 clears, val ≠ 0 sets)
- bit: relative bit position in register or memory word

Example

```
#include <regs.h>
/* SET BIT 8 of memory location 0x1000 to 1        */
ASSIGN_BIT_VAL((unsigned int *)0x1000,8,1);
```

CACHE_BYPASS

Bypass program memory cache

Syntax

#include <cache.h>
#define **CACHE_BYPASS( )**

Defined in

cache.h as a macro

Description

CACHE_BYPASS places the internal program memory in the cache bypass state, where the cache retains its current state. A program read to a bypassed cache causes the fetch packets to be fetched from the EMIF.

Example

```
#include <cache.h>
CACHE_BYPASS( );
```

CACHE_DISABLE

Disable program memory cache

Syntax

#include <cache.h>
#define **CACHE_DISABLE( )**

Defined in

cache.h as a macro

Description

CACHE_DISABLE sets the internal program memory to mapped mode, in which program fetches to an internal program memory address return the fetch packet at that address.

Example

```
#include <cache.h>
CACHE_DISABLE( );
```

CACHE_ENABLE

Enable program memory cache

Syntax

#include <cache.h>
#define **CACHE_ENABLE( )**

Defined in

cache.h as a macro

Description

CACHE_ENABLE enables the internal program memory to be used as a program cache.

Example

```
#include <cache.h>
CACHE_ENABLE( );
```

CACHE_FLUSH — *Flush program memory cache*

Syntax

#include <cache.h>
#define **CACHE_FLUSH( )**

Defined in

cache.h as a macro

Description

CACHE_FLUSH flushes the cache by transitioning internal program memory modes from mapped to enabled. This mode transition is the only mechanism that flushes the cache.

Example

```
#include <cache.h>
CACHE_FLUSH( );
```

CACHE_FREEZE — *Freeze program memory cache*

Syntax

#include <cache.h>
#define **CACHE_FREEZE( )**

Defined in

cache.h as a macro

Description

CACHE_FREEZE places the internal program memory in the cache freeze state, in which the cache retains its current state. A program read to a frozen cache is identical to a read from an enabled cache, with the exception that on a cache miss the data read from the EMIF is not stored in the cache.

Example

```
#include <cache.h>
CACHE_FREEZE( );
```

DMA_AUTO_START

DMA start operation for autoinitialization

Syntax

#include <dma.h>
#define **DMA_AUTO_START(**chan**)**

Defined in

dma.h as a macro

Description

DMA_AUTO_START starts the DMA operation of the selected channel by setting the START field of the selected channel's primary control register (PCR) to a value of 11b. After completion of a block transfer, the DMA channel is restarted and the selected DMA channel registers are reloaded. It uses the following parameter:

- chan: channel selector (0,1)

Example

```
#include <dma.h>
/*Start DMA channel 2 operation using autoinitialization*/
DMA_AUTO_START(2);
```

DMA_DEST_ADDR_ADDR

Selects DMA desgination address register address

Syntax

#include <dma.h>
#define **DMA_DEST_ADDR_ADDR(**chan**)**

Defined in

dma.h as a macro

Description

DMA_DEST_ADDR_ADDR returns the address of the DMA destination address register for the selected channel. It uses the following parameter:

- chan: channel selector (0,1)

Example

```
#include <dma.h>
/* ----------------------------------------------------------*/
/*Set destination address for DMA channel register 2*/
/* ----------------------------------------------------------*/
* (unsigned int *) DMA_DEST_ADDR_ADDR(2) = (unsigned
int *)0X00400000u;
```

dma_global_init *Initialize DMA global registers*

Syntax

```
#include <dma.h>
void dma_global_init (unsigned int gcr,
                      unsigned int gra,
                      unsigned int grb,
                      unsigned int gndxa,
                      unsigned int gndxb,
                      unsigned int gaddra,
                      unsigned int gaddrb,
                      unsigned int gaddrc,
                      unsigned int gaddrd)
```

Defined in

dma.h as a static inline function
dma.c as a callable function

Description

dma_global_init assigns the values in the parameter list to their corresponding registers. It uses the following parameters:

- gcr: value to set DMA global control register
- gcra: value to set DMA global count reload register A
- gcrb: value to set DMA global count reload register B
- gndxa: value to set DMA global index register A
- gndxb: value to set DMA global index register B
- gaddra: value to set DMA global address register A
- gaddrb: value to set DMA global address register B
- gaddrc: value to set DMA global address register C
- gaddrd: value to set DMA global address register D

Example

```
#include <dma.h>
/*---------------------------------------------------------------------------*/
/* Initialize DMA global control registers                                   */
/* gcr = 0x1 set aux ctrl register chan pri to give chan 2 priority          */
/* gcra = 0x256 set element count in global count register A                 */
/* gndxa = 0x1 set index increment address by 1                              */
/* gaddrb = 0x400000 set split address in global address reg B               */
/* gaddro = 0x200000 set source reload address in global addr reg C          */
/*---------------------------------------------------------------------------*/
dma_global_init
(0x1, 0x256, 0x0, 0x1, 0x0, 0x0, 0x00400000, 0x00200000, 0x0)
```

dma_init — *Initialize DMA channel-specific registers*

Syntax

```
#include <dma.h>
void dma_init (unsigned int chan,
               unsigned int pri_ctrl,
               unsigned int sec_ctrl,
               unsigned int src_addr,
               unsigned int dst_addr,
               unsigned int trans_ctr)
```

Defined in

dma.h as a static inline function
dma.c as a callable function

Description

dma_init initializes a DMA channel by assigning the values in the parameter list to their corresponding registers. It uses the following parameters:

- chan: channel selector (0,1)
- pri_ctrl: value to set DMA primary control register
- sec_ctrl: value to set DMA secondary control register
- src_addr: value to set DMA source address register
- dst_addr: value to set DMA destination address register
- trans_ctr: value to set DMA transfer counter register

Example

```
#include <dma.h>
/*----------------------------------------------------------------------------*/
/* Initialize control registers for DMA channel 2                             */
/* channel = 2 initialize registers for DMA channel 2                         */
/* pri_ctrl = 0x2A00A10U use global index A, element size 8 bits,             */
/* use global address B as split address, enable interrupts                   */
/* pause during emulation halts, reload source address                        */
/* from global address reg C                                                  */
/* sec_ctrl = 0x0000000Au enable interrupt on completion of each frame        */
/*----------------------------------------------------------------------------*/
extern unsigned short outbuf [ ];
dma_init(2, 0x2A000A10u, 0x0000000Au, (unsigned int) & outbuf,0x00400000u,
0x00050080u);
```

DMA_PAUSE *DMA pause transfer*

Syntax

#include <dma.h>
#define **DMA_PAUSE(**chan**)**

Defined in

dma.h as a macro

Description

DMA_PAUSE pauses the DMA transfer for the indicated channel. Any write transfers whose read transfer requests are complete is also completed. If the DMA channel has all of the necessary read synchronizations for the next element, one more element transfer is allowed to complete. DMA_PAUSE uses the following parameter:

- chan: channel selector (0–3)

Example

```
#include <dma>h>
/* ------------------------------------------------------------*/
/* Suspend DMA channel 2 operation                             */
/* ------------------------------------------------------------*/
DMA_PAUSE (2);
```

DMA_PRIMARY_CTRL_ADDR

Select DMA primary control register address

Syntax

#include <dma.h>
#define **DMA_PRIMARY_CTRL_ADDR(**chan**)**

Defined in

dma.h as a macro

Description

DMA_PRIMARY_CTRL_ADDR returns the address of the DMA primary control register for the selected channel. It uses the following parameter:

- chan: channel selector (0–3)

Example

```
#include <dma.h>
/*----------------------------------------------------------------------------*/
/* Set DMA channel 2 primary control:                                         */
/* dst reload     = 00b no reload of destination address for autotint         */
/* source reload  = 10b reload source address from global reg C               */
/* emod           = 1b pause DMA during emulation halts                       */
/* fs             = 0b no frame synchronization                               */
/* toint          = 1b transfer controller interrupt enable                   */
/* pri            = 0b CPU access has priority over DMA                       */
/* wsync          = 00000b no synchorization                                  */
/* rsync          = 00000b no synchorization                                  */
/* index          = 0b use global index register A as programmable index      */
/* cnt reload     = 0b reload with global count register A                    */
/* split          = 10b use DMA global address register B as split address*/
/* esize          = 10b set element transfer size to 8 bits                   */
/* dst directory  = 00b no modification, writing data to external device      */
/* src directory  = 01b adjust using global index register (reg A)            */
/* status         = xx status bits read only                                  */
/* start          = 00b hold DMA until control registers have been written */
/* Full mask      = 0010 1010 0000 0000 0000 1010 0001 0000b, 0x2A000A10      */
/*----------------------------------------------------------------------------*/
*(unsigned int *) DMA_PRIMARY_CTRL_ADDR(2) = (unsigned int *)0X2A000A10U;
```

dma_reset

Reset DMA registers to default state

Syntax

#include <dma.h>
void **dma_reset(**void**)**

Defined in

dma.h as a static inline function
dma.c as a callable function

Description

DMA_RESET resets all DMA registers to their power-on reset state.

Example

```
#include <dma.h>
/* ------------------------------------------------------- */
/* Reset DMA                                               */
/* ------------------------------------------------------- */
dma_reset( );
```

DMA_RSYNC_CLR

Clear read sync bit in DMA secondary control register

Syntax

#include <dma.h>
#define **DMA_RSYNC_CLR(**chan**)**

Defined in

dma.h as a macro

Description

DMA_RSYNC_CLR clears the read sync bit in the DMA secondary control register for the specified channel, turning off read synchronization. It uses the following parameter:

- chan: channel selector (0–3)

Example

```
#include <dma.h>
DMA_RSYNC_CLR(0);
```

DMA_RSYNC_SET

Set read sync bit in DMA secondary control register

Syntax

#include <dma.h>
#define **DMA_RSYNC_SET(**chan**)**

Defined in

dma.h as a macro

Description

DMA_RSYNC_SET sets the read sync bit in the DMA secondary control register for the specified channel, enabling read synchronization. It uses the following parameter:

- chan: channel selector (0–3)

Example

```
#include <dma.h>
DMA_RSYNC_SET(0);
```

DMA_SECONDARY_CTRL_ADDR

Select DMA secondary control register address

Syntax

#include <dma.h>
#define **DMA_SECONDARY_CTRL_ADDR(**chan**)**

Defined in

dma.h as a macro

Description

DMA_SECONDARY_CTRL_ADDR returns the address of the DMA secondary control register for the selected channel. It uses the following parameter:

- chan: channel selector (0–3)

Example

```
#include <dma.h>
/*-------------------------------------------------------------------------*/
/* Set secondary control register for DMA channel 2:                       */
/* frame ie = 1b enable interrupt on completion of each frame xfr          */
/* sx ie = 1b enable interrupt on split transmit overun                    */
/* Full mask = 0000 0000 0000 0000 0000 0000 0000 1010b, 0x0000000A        */
/* Value may be set either by using direct assignment of full mask or using*/
/* MASK_BIT macros to facilitate code maintainability/readibility          */
/*-------------------------------------------------------------------------*/
*(unsigned int *) DMA_SECONDARY_CTRL_ADDR (2) = (unsigned int *)MASK_BIT
(FRAME_IE) | MASK_BIT(SX_IE);
```

DMA_SRC_ADDR_ADDR

Select DMA source address register address

Syntax

#include <dma.h>
#define **DMA_SRC_ADDR_ADDR(**chan**)**

Defined in

dma.h as a macro

Description

DMA_SRC_ADDR_ADDR returns the address of the DMA source address register for the selected channel. It uses the following parameter:

- chan: channel selector (0,1)

Example

```
#include <dma.h>
extern unsigned short outbuf [ ];
/*---------------------------------------------------------------------- */
/* Set source address for DMA channel 2                                  */
/*---------------------------------------------------------------------- */
* (unsigned int *) DMA_SRC_ADDR_ADDR (2) = (unsigned int) & outbuf;
```

DMA_START

DMA manual start operation

Syntax

#include <dma.h>
#define **DMA_START(**chan**)**

Defined in

dma.h as a macro

Description

DMA_START manually starts the DMA operation of the selected channel by setting the START field of the selected channel's primary control register (PCR) to a value of 01. It uses the following parameter:

- chan: channel selector (0–3)

Example

```
include <dma.h>
/* ------------------------------------------------------------*/
/* Start DMA channel 2 operation                               */
/*  ------------------------------------------------------------*/
DMA_START(DMA_CH2);
```

DMA_STOP

DMA stop transfer

Syntax

#include <dma.h>
#define **DMA_STOP(**chan**)**

Defined in

dma.h as a macro

Description

DMA_STOP immediately stops the DMA operation on the selected channel and discards any data from completed read transfers that is held internally. It uses the following parameter:

- chan: channel selector (0–3)

Example

```
#include <dma.h>
DMA_STOP(DMA_CH2);
```

DMA_WSYNC_CLR

Clear write sync bit in DMA secondary control register

Syntax

#include <dma.h>
#define **DMA_WSYNC_CLR(**chan**)**

Defined in

dma.h as a macro

Description

DMA_WSYNC_CLR clears the write sync bit in the DMA secondary control register for the specified channel, turning off write synchronization. It uses the following parameter:

- chan: channel selector (0–3)

Example

```
#include <dma.h>
DMA_WSYNC_CLR(0);
```

DMA_WSYNC_SET

Set write sync bit in DMA secondary control register

Syntax

#include <dma.h>
#define **DMA_WSYNC_SET(**chan**)**

Defined in

dma.h as a macro

Description

DMA_WSYNC_SET sets the write sync bit in the DMA secondary control register for the specified channel, enabling write synchronization. It uses the following parameter:

- chan: channel selector (0–3)

Example

```
#include <dma.h>
DMA_WSYNC_SET(0);
```

DMA_XFER_COUNTER_ADDR

Select DMA transfer counter register address

Syntax

#include <dma.h>
#define **DMA_XFER_COUNTER_ADDR(**chan**)**

Defined in

dma.h as a macro

Description

DMA_XFER_COUNTER_ADDR returns the address of the DMA transfer counter register for the selected channel.

- chan: channel selector (0–3)

Example

```
#include <dma.h>
/* Get address of transfer counter for DMA channel 2 */
unsigned int *xfer_counter = (unsigned int *)
DMA_XFER_COUNTER_ADDR(2);
```

EMIF_GET_MAP_MODE

Return value of MAP bit in EMIF global control register

Syntax

#include <emif.h>
#define **EMIF_GET_MAP_MODE( )**

Defined in

emif.h as a macro

Description

EMIF_GET_MAP_MODE returns the value of the MAP bit in the EMIF global control register. The MAP bit determines whether internal or external memory is mapped at address 0.

Example

```
#include <emif.h>
unsigned short map = EMIF_GET_MAP_MODE( );
```

emif_init

Initialize EMIF registers

Syntax

```
#include <emif.h>
void emif_init(unsigned int g_ctrl,
               unsigned int ce0_ctrl,
               unsigned int ce1_ctrl,
               unsigned int ce2_ctrl
               unsigned int ce3_ctrl,
               unsigned int sdram_ctrl,
               unsigned int sdram_refresh)
```

Defined in

emif.h as a static inline function
emif.c as a callable function

Description

emif_init assigns the values in the parameter list to their corresponding registers. It uses the following parameters:

- g_ctrl: value to set EMIF global control register
- ce0_ctrl: value to set CE0 space control register
- ce1_ctrl: value to set CE1 space control register
- ce2_ctrl: value to set CE2 space control register
- ce3_ctrl: value to set CE3 space control register
- sdram_ctrl: value to set SDRAM control register
- sdram_refresh: value to set SDRAM refresh period register

Example

```
/* ----------------------------------------------------------*/
/* Initialize external memory interface for 8-bit    */
/* ROM                                                       */
/*                                                           */
/* g_ctrl –0x00000090 CLKC1EN, NOHOLD                */
/* ce0_ctrl – 0x20920201  READ/WRITE setup 2 cycles  */
/*                        READ/WRITE strobe 2 cycles */
/*                        READ/WRITE hold 1 cycle    */
/* ----------------------------------------------------------*/
#include <emif.h>
emif_init(0x00000090,  0x20920201,  0x0,  0x0,  0x0,  0x0,
0x0);
```

GET_BIT *Return value of bit*

Syntax

#include <regs.h>
#define **GET_BIT(**addr,bit**)**

Defined in

regs.h as a macro

Description

This macro returns the value of the indicated bit in the register whose address is given by the parameter addr. It uses the following parameters:

- addr: address of peripheral control register or other memory word
- bit: indicates the relative position in the word, the bit's value is returned

Example

```
/*---------------------------------------------------------------------------*/
/* Include header for multichannel buffered serial port                      */
/* Inclusion of mcbsp.h automatically includes regs.h                        */
/*---------------------------------------------------------------------------*/
#include <mcbsp.h>
/*---------------------------------------------------------------------------*/
/* Test RRDY bit in multichannel buffered serial port 0                      */
/* control register                                                          */
/*---------------------------------------------------------------------------*/
unsigned int xx;
if (GET_BIT(MCBSP_SPCR_ADDR(0),RRDY) xx = MCBSP0_DRR;
```

GET_FIELD — *Return value of bits in bit field*

Syntax

#include <regs.h>
#define **GET_FIELD(**addr,bit,length**)**

Defined in

regs.h as a macro

Description

GET_FIELD returns the value of the bits in the indicated bit field within the register whose address is given by addr. GET_FIELD uses the following parameters:

- addr: address of peripheral control register or memory word
- bit: starting bit location of desired bit field
- length: length of bit field

Example

```
/*-------------------------------------------------------------------------------*/
/* Include DMA register, macro, and function definitions                         */
/* Including dma.h, automatically includes regs.h                                */
/*-------------------------------------------------------------------------------*/
#include <dma.h>                                                                 /*
-------------------------------------------------------------------------------*/
/* Get value of number elements per frame from DMA transfer                      */
/* counter register                                                              */
/* register                                                                      */
/*-------------------------------------------------------------------------------*/
ushort no_elements_per_frame =
GET_FIELD(DMA0_XFR_COUNTER_ADDR,ELEMENT_COUNT,ELEMENT_COUNT_SZ);
```

GET_REG — *Return value in register*

Syntax

#include <regs.h>
#define **GET_REG(**reg**)**

Defined in

regs.h as a macro

Description

GET_REG returns the value in the named register. It uses the following parameter:

- reg: name of the peripheral control register (TIMER0_CTRL, TIMER1_PERIOD, etc.)

Example

```
/*-------------------------------------------------------------------------------*/
/* Include timer.h to gain access to timer control regs,                         */
/* macros, and function definitions. Including timer.h                           */
/* automatically includes regs.h                                                 */
/*-------------------------------------------------------------------------------*/
#include <intr.h>
unsigned int clock_tics = GET_REG(TIMER0_COUNTER);
```

GET_REG_BIT — *Return value of bit in named register*

Syntax

#include <regs.h>
#define **GET_REG_BIT(**reg,bit**)**

Defined in

regs.h as a macro

Description

GET_REG_BIT returns the value of the indicated bit in the named register. It uses the following parameters:

- reg: name of register (DMA_GCTRL, DMA0_PRIMARY_CTRL, TIMER0_CTRL, etc.)
- bit: bit in register whose value is to be returned.

Example

```
/*---------------------------------------------------------------------------*/
/* Include mcbsp.h to gain access to multi channel                           */
/* buffered serial port registers, macros, and functions                     */
/* Including mcbsp.h, automatically includes regs.h                          */
/*---------------------------------------------------------------------------*/
#include <mcbsp.h>
while (!(GET_REG_BIT(MCBSP0_SPCR,RRDY));
/* Wait for serial port read ready                                           */
```

GET_REG_FIELD — *Return value of bit field in named register*

Syntax

#include <regs.h>
#define **GET_REG_FIELD(**reg,bit,length**)**

Defined in

regs.h as a macro

Description

GET_REG_FIELD returns the value of the indicated bit field in the named register. It uses the following parameters:

- reg: name of register to access
- bit: starting position of bit field in register
- length: length of bit field

Example

```
/*---------------------------------------------------------------------------*/
/* Include emif.h to gain access to the external                             */
/* Memory interface registers, macros, and functions                         */
/* Including emif.h, automatically includes, regs.h                          */
/*---------------------------------------------------------------------------*/
#include <emif.h>
unsigned short strobe_width = GET_REG_FIELD (EMIF_CEO_CTRL, READ_STROBE,
READ_STROBE_SZ);
```

HPI_GET_DSPINT

Return state of DSP interrupt

Syntax

#include <hpi.h>
#define **HPI_GET_DSPINT( )**

Defined in

hpi.h as a macro

Description

HPI_GET_DSPINT returns the value of the DSPINT bit in the HPI control register.

Example

```
/*-------------------------------------------------------------------------*/
/* Including hpi.h to gain access to host port interface                   */
/* registers, macros, and functions.   *                                   */
/*-------------------------------------------------------------------------*/
while (!(HPI_GET_DSPINT( ))) write_to_hpi_mem( );
/* Write to HPI mem until HOST interrupt */
```

HPI_GET_HINT

Return state of host interrupt

Syntax

#include <hpi.h>
#define **HPI_GET_HINT( )**

Defined in

hpi.h as a macro

Description

HPI_GET_HINT returns the value of the HINT bit in the HPI control register.

Example

```
/*-------------------------------------------------------------------------*/
/* Include hpi.h to gain access to host port interface                     */
/* registers, macros, and functions                                        */
/*-------------------------------------------------------------------------*/
while (! (HPI_GET_HINT( )));
/* pause until HOST has cleared DSP to HOST interrupt                      */
```

HPI_RESET _DSPINT

Reset/clear DSP interrupt

Syntax

#include <hpi.h>
#define **HPI_RESET_DSPINT( )**

Defined in

hpi.h as a macro

Description

HPI_RESET_DSPINT resets/clears the DSPINT signal by writing a 1 to the DSPINT bit in the HPI control register.

Example

```
/*---------------------------------------------------------------------------*/
/* Include hpi.h to gain access to host port interface                       */
/* registers, macros, and functions                                          */
/*---------------------------------------------------------------------------*/
HPI_RESET_DSPINT( );
```

HPI_SET_HINT

Generate host interrupt

Syntax

#include <hpi.h>
#define **HPI_SET_HINT( )**

Defined in

hpi.h as a macro

Description

HPI_SET_HINT sets the HINT bit in the HPI control register that generates a host interrupt.

Example

```
/*---------------------------------------------------------------------------*/
/* Include hpi.h to gain access to host port interface                       */
/* registers, macros, and functions.                                         */
/*---------------------------------------------------------------------------*/
#include <hpi.h>
HPI_SET_HINT( );

/* Send interrupt to host processor                                          */
```

IDLE

Idle the processor

Syntax

#include <cache.h>
#define **IDLE( )**

Defined in

cache.h as a macro

Description

IDLE issues the assembly instruction IDLE, which performs a multicycle NOP. The idle is broken when an interrupt is serviced.

Example

```
/* ------------------------------------------------------------ */
/* Idle CPU                                                     */
/* ------------------------------------------------------------ */
#include <cache.h>
IDLE( );
```

INTR_CHECK_FLAG

Check value of bit in IFR

Syntax

#include <intr.h>
#define **INTR_CHECK_FLAG(**bit**)**

Defined in

intr.h as a macro

Description

INTR_CHECK_FLAG returns the value of the indicated bit in the Interrupt flag register (IFR). It uses the following parameter:

- bit: bit position in interrupt flag register to poll

Example

```
#include <intr.h>
while (INTR_CHECK_FLAG (CPU_INT14));
```

INTR_CLR _FLAG

Clear interrupt manually

Syntax

#include <intr.h>
#define **INTR_CLR_FLAG(**bit**)**

Defined in

intr.h as a macro

Description

INTR_CLR_FLAG manually clears the selected interrupt by writing a 1 to the specified bit in the ICR. Writing a 1 to IC4–IC15 in the ICR causes the corresponding bit in the IFR to be cleared. INTR_CLR_FLAG uses the following parameter:

- bit: CPU interrupt (value 0–15) that must be cleared

Example

```
#include <intr.h>
INTR_CLR_FLAG (CPU_INT14);
```

INTR_DISABLE

Disable interrupt

Syntax

#include <intr.h>
#define **INTR_DISABLE(**bit**)**

Defined in

intr.h as a macro

Description

intr_disable disables the specified interrupt by clearing the indicated bit in the interrupt enable register (IER). It uses the following parameter:

- bit: CPU interrupt (value 0–15) that disables the specified interrupt

Example

```
#include <intr.h>
INTR_DISABLE (CPU_14);
/* Disable CPU 14 interrupt */
```

INTR_ENABLE

Enable interrupt

Syntax

#include <intr.h>
#define **INTR_ENABLE(**bit**)**

Defined in

intr.h as a macro

Description

intr_enable (bit) enables the specified interrupt by setting the indicated bit in the interrupt enable register (IER). It uses the following parameter:

- bit: CPU interrupt (value 0–15) that enables the specified interrupt

Example

```
#include <intr.h>
INTR_ENABLE (CPU_INT14);
/* Enable CPU 14 interrupt */
```

INTR_EXT_POLARITY

Set polarity for external interrupts

Syntax

#include <intr.h>
#define **INTR_EXT_POLARITY(**bit,val**)**

Defined in

intr.h as a macro

Description

INTR_EXT_POLARITY sets polarity of the four external interrupts by writing to the external interrupt polarity register. It uses the following parameters:

- bit: bit in external polarity register (0 – affects external interrupt-4 polarity, 1 – affects external interrupt-5 polarity, 2 – affects external interrupt-6 polarity, 3 – affects external interrupt-7 polarity)
- val: 0/1 clears/sets (respectively) the external polarity for the specified external interrupt

Example

```
#include <intr.h>
INTR_EXT_POLARITY (XIP4; 1);
/* Invert external polarity for external interrupt #4*/
```

intr_get_cpu_intr

Return CPU interrupt corresponding to given interrupt source

Syntax

#include <intr.h>
int **intr_get_cpu_intr(**int isn**)**

Defined in

intr.c as a callable function

Description

intr_get_cpu_intr returns the interrupt number corresponding to the indicated interrupt selection number (isn). It uses the following parameter:

- isn: interrupt selector number (0–15). If the isn is not currently mapped, ERROR is returned.

Example

```
#include <intr.h>
int cpuint = intr_get_cpu_intr(ISN_TINT0);
/*Returns the CPU interrupt number t0 which timer interrupt 0 is mapped*/
```

INTR_GET_ISN

Return interrupt source corresponding to CPU interrupt

Syntax

#include <intr.h>
int **INTR_GET_ISN(**int cpu_intr**)**

Defined in

intr.h as a macro

Description

INTR_GET_ISN returns the isn value corresponding to the indicated CPU interrupt from the appropriate interrupt multiplexer register. It uses the following parameter:

- cpu_intr: CPU interrupt (0–15)

Example

```
#include <intr.h>
unsigned short interrupt = (unsigned short)
int interrupt = INTR_GET_ISN (CPU_INT4);
/* Get interrupt selector associated with CPU interrupt #4  */
```

INTR_GLOBAL_DISABLE

Globally disable all maskable interrupts

Syntax

#include <intr.h>
#define **INTR_GLOBAL_DISABLE( )**

Defined in

intr.h as a macro

Description

INTR_GLOBAL_DISABLE globally disables all maskable interrupts by clearing the GIE bit in the control status register (CSR). Interrupts may also be disabled by the INTR_DISABLE macro, which clears the corresponding interrupt bit in the IER.

Example

```
#include <intr.h>
INTR_GLOBAL_DISABLE( );

/* Globally disable all interrupts*/
```

INTR_GLOBAL_ENABLE

Globally enable all maskable interrupts

Syntax

#include <intr.h>
#define **INTR_GLOBAL_ENABLE( )**

Defined in

intr.h as a macro

Description

INTR_GLOBAL_ENABLE globally enables all maskable interrupts by setting the GIE bit in the control status register (CSR). For interrupts to be processed, the corresponding bit in the Interrupt enable register (IER) must also be set.

Example

```
#include <intr.h>
INTR_GLOBAL_ENABLE( );

/* Globally enable all interrupts                */
```

intr_hook

Hook an interrupt service function to an interrupt

Syntax

#include <intr.h>
void **intr_hook(**void(*fp)(void),intr_num**)**

Defined in

intr.c as callable function

Description

intr_hook places the function pointer indicated by fp into isr_jump_table[ ] at the location specified by intr_num. It uses the following parameters:

- ❏ fp: vector location for interrupt (given as absolute offset from base vector address). Note that fp is a pointer to an ISR declared in C by the interrupt keyword.
- ❏ intr_num: interrupt service routine to invoke when servicing this interrupt

Example

```
#include <intr.h>
void extern_1 (void);
/* Prototype for interrupt service routine          */
intr_hook (extern_1, CPU_INT3)
```

intr_init

Initialize interrupt processing

Syntax

#include <intr.h>
void **intr_init(**void**)**

Defined in

intr.c as callable function

Description

intr_init initializes the interrupt service table pointer (ISTP) with the address of the global label vec_table, which is resolved at link time. The address vec_table is defined in intr_.asm and its value is determined at link time.

Example

```
#include <intr.h>
intr_init();
/* Place base address of vector table in ISTP       */
```

intr_isn

Return interrupt source corresponding to CPU interrupt

Syntax

#include <intr.h>
int **intr_isn(**int cpu_intr**)**

Defined in

intr.c as a callable function

Description

intr_isn returns the interrupt source number corresponding to the CPU interrupt specified by cpu_intr. It uses the following parameter:

- cpu_intr: CPU interrupt (0–15)

Example

```
#include <intr.h>
/*Return interrupt number mapped to CPU interrupt number 4                */
int isn = intr_isn(CPU_INT4);
```

intr_map

Map interrupt source to CPU interrupt

Syntax

#include <intr.h>
void **intr_map(**int cpu_intr,int isn**)**

Defined in

intr.c as a callable function

Description

intr_map places the indicated ISN value in the appropriate field of the appropriate interrupt multiplexer register. It uses the following parameters:

- cpu_intr: selects the CPU interrupt (0–15) to which to map the corresponding interrupt given by isn
- isn: interrupt selector number to map to indicated CPU interrupt

Example

```
/* Map timer 0 interrupt selector to CPU interrupt 4 */
intr_map(CPU_INT4,TIMER0_INT);
```

INTR_MAP_RESET

Reset interrupt multiplexer registers

Syntax

#include <intr.h>
#define **INTR_MAP_RESET( )**

Defined in

intr.h as a macro

Description

INTR_MAP_RESET resets the interrupt multiplexer high and interrupt multiplexer low registers to their default values.

Example

```
#include <intr.h>
INTR_MAP_RESET( );
/* Set low and high interrupt multiplexer registers to default values     */
```

INTR_RETURN_ISN

Return interrupt source corresponding to CPU interrupt

Syntax

#include <intr.h>
#define **INTR_RETURN_ISN(**intr,sel**)**

Defined in

intr.h as a macro

Description

INTR_RETURN_ISN returns the interrupt source number corresponding to the CPU interrupt specified by intr. It uses the following parameters:

- ❏ intr: specifies the CPU interrupt
- ❏ sel: selects between the low and high interrupt multiplexer registers

Example

```
#include <intr.h>
int interrupt_source = INTR_RETURN_ISN(CPU_INT4, 0);
```

INTR_SET_FLAG

Set interrupt manually

Syntax

#include <intr.h>
#define **INTR_SET_FLAG(**bit**)**

Defined in

intr.h as a macro

Description

INTR_SET_FLAG manually sets the selected interrupt by writing a 1 to the specified bit in the ISR. Writing a 1 to IS4–IS15 in the ISR causes the corresponding bit in the IFR to be set. INTR_SET_FLAG uses the following parameter:

- bit: CPU interrupt to mask (0–15)

Example

```
#include <intr.h>
INTR_SET_FLAG(CPU_INT4);
```

INTR_SET_MAP

Map interrupt source to a CPU interrupt

Syntax

#include <intr.h>
#define **INTR_SET_MAP(**intr,val,sel**)**

Defined in

intr.h as a macro

Description

INTR_SET_MAP maps an interrupt source specified by val to the CPU intr specified by intr. It uses the following parameters:

- intr: CPU interrupt to map
- val: interrupt source
- sel: 0/1 selects interrupt multiplexer register low or high, respectively

Example

```
#include <intr.h>
INTR_SET_MAP (CPU_INT4, ISN_EXT_INT4, 0);
/* Map CPU interrupt 4 to external interrupt 4 (as source)                */
```

LOAD_FIELD *Assign value of bits in bit field*

Syntax

#include <regs.h>
#define **LOAD_FIELD(**addr,val,bit,length**)**

Defined in

regs.h as a macro

Description

LOAD_FIELD assigns the bits within the indicated bit field of the register whose address is given by addr to val. It uses the following parameters:

- addr: address of peripheral control register/memory word to access
- bit: starting bit position of the bit field
- length: length of bit field (number of bits) defined in regs.h
- val: value to assign to the bit field

Example

```
/*-------------------------------------------------------------------------*/
/* Include emif.h to gain access to the external                           */
/* Memory interface registers, macros, and functions                       */
/* Including emif.h, automatically includes regs.h                         */
/*-------------------------------------------------------------------------*/
LOAD_FIELD (EMIF_CE0_CTRL__ADDR, READ_STROBE, READ_STROBE_SZ, 4);
/* Set read strobe width to 4 cycles                                       */
```

MASK_BIT *Create bit mask*

Syntax

#include <regs.h>
#define **MASK_BIT(**bit**)**

Defined in

regs.h as a macro

Description

MASK_BIT returns a bit mask for the specified bit. A bit mask is defined as a 1 in the indicated bit position with all other bits 0.

- bit: bit to mask in word

Example

```
/* ----------------------------------------------------------*/
/* Globally enable interrupts by setting                     */
/* GIE bit in Control Status Register                        */
/* ----------------------------------------------------------*/
#include <regs.h>
CSR = MASK_BIT(GIE);
```

MASK_FIELD — *Return bit field mask*

Syntax

#include <regs.h>
#define **MASK_FIELD(**bit,length**)**

Defined in

regs.h as a macro

Description

MASK_FIELD returns a bit field mask at the indicated bit with length number of bits masked. A bit field mask is defined as 1s in the bit locations comprising the field and 0s elsewhere. It uses the following parameters:

- bit: starting bit position of field to mask
- length: length of bit field (number of bits)

Example

```
/*---------------------------------------------------------------------------*/
/* Get CPU ID from control status register                                   */
/*---------------------------------------------------------------------------*/
#include <regs.h>
unsigned short cpu_id |= MASK_FIELD (CPU_ID, CPU_ID_SZ) >> CPU_ID;
```

MCBSP_ADDR — *Return base address*

Syntax

#include <mcbsp.h>
#define **MCBSP_ADDR(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_ADDR returns the base address of the block of control registers for the specified MCBSP port. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
#include <mcbsp.h>
unsigned int *port0_base_addr = (unsigned int *)
MCBSP_ADDR(0);
```

MCBSP_BYTES_PER_WORD

Return bytes to hold bits indicated by wdlen

Syntax

#include <mcbsp.h>
#define **MCBSP_BYTES_PER_WORD(**wdlen**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_BYTES_PER_WORD returns the number of bytes required to hold the number of bits indicated by wdlen. It uses the following parameter:

- wdlen: number of bits

Example

```
#include
unsigned short bytes = MCBSP_BYTES_PER_WORD(30);
```

MCBSP_DRR_ADDR

Select MCBSP data receive register address

Syntax

#include <mcbsp.h>
#define **MCBSP_DRR_ADDR(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_DRR_ADDR returns the address of the MCBSP data receive register for the selected channel. It uses the following parameters:

- port_no: indicates the selected port (0,1)

Example

```
/* ----------------------------------------------------*/
/* Get address of multi channel buffered               */
/* serial port 1, receive address reg                  */
/* ----------------------------------------------------*/
#include <mcbsp.h>
unsigned int *ptr = (unsigned int *) MCBSP_DRR_ADDR
(1);
```

MCBSP_DXR_ADDR

Select MCBSP data transmit register address

Syntax

#include <mcbsp.h>
#define **MCBSP_DXR_ADDR(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_DXR_ADDR returns the address of the MCBSP data transmit register for the selected channel. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ----------------------------------------------------------*/
/* Get address of data transmit register                     */
/* for MCBSP port 0.                                         */
/* ----------------------------------------------------------*/
#include <mcbsp.h>
unsigned int *ptr = (unsigned int *)MCBSP_DXR_ADDR (0);
```

MCBSP_ENABLE

Enable multichannel buffered serial port

Syntax

#include <mcbsp.h>
#define **MCBSP_ENABLE(**port_no,type**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_ENABLE enables either the receive, transmit or both sections of the selected multichannel buffered serial port. It uses the following parameters:

- port_no: selects which McBSP port to enable
- type: specifies the transfer mode to enable for the selected port. Macros defined in mcbsp.h such as MCBSP_TX should be used to specify the transfer mode.

Example

```
#include <mcbsp.h>
/* enable mcbsp 0 transmit                                   */
MCBSP_ENABLE(0,MCBSP_TX)
```

MCBSP_FRAME_SYNC_ENABLE

Enable frame sync generation logic for selected port

Syntax

#include <mcbsp.h>
#define **MCBSP_FRAME_SYNC_ENABLE(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_FRAME_SYNC_ENABLE enables the frame sync generation logic for the selected port. Note that the sample rate generator must also be enabled for the frame sync signal FSG to become active. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ------------------------------------------------------*/
/* Enable frame sync for MCBSP 1                         */
/* ------------------------------------------------------*/
#include <mcbsp.h>
SET_BIT(MCBSP_SPCR_ADDR(1),GRST);
/* Enable Sample Rate Generator                          */
MCBSP_FRAME_SYNC_ENABLE(1);
```

MCBSP_FRAME_SYNC_RESET

Reset/disable frame sync generation logic for selected port

Syntax

#include <mcbsp.h>
#define **MCBSP_FRAME_SYNC_RESET(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_FRAME_SYNC_RESET resets/disables the frame sync generation logic for the selected port. The internal frame sync signal, FSG, is driven inactive low. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ------------------------------------------------------*/
/*Disable frame sync for MCBSP 0 port                    */
/* ------------------------------------------------------*/
#include <mcbsp.h>
MCBSP_FRAME_SYNC_RESET(0);
```

mcbsp_init — *Initialize multichannel buffered serial port*

Syntax

#include <mcbsp.h>
void **mcbsp_init**(port_no,spcr_ctrl,rcr_ctrl,xcr_ctrl,srgr_ctrl,mcr_ctrl,
rcer_ctrl,xcer_ctrl,pcr_ctrl)

Defined in

mcbsp.h as a static inline function
mcbsp.c as a callable function

Description

mcbsp_init initializes the registers indicated in the argument list to their corresponding parameter values. It uses the following parameters:

- port_no: selects port (0,1)
- spcr_ctrl: value to set serial port control register
- rcr_ctrl: value to set receive control register
- xcr_ctrl: value to se transmit control register
- srgr_ctrl: value to set sample rate generator register
- mcr_ctrl: value to set multi–channel control register
- rcer_ctrl: value to set receive channel enable register
- xcer_ctrl: value to set transmit channel enable register
- pcr_ctrl: value to set pin control register

Example

```
#include
/* ---------------------------------------------------------------------- */
/* Set control for buffered serial port control register                  */
/*                                                                        */
/* port  = 1                                                              */
/* spcr_ctrl= 0x00002000u right justify zero – fill                       */
/* rcr_ctrl= 0x00047F00u Pulse after first frame sync ignored             */
/* Frame length 128 words                                                 */
/* Word size 8 bits                                                       */
/* xcr_ctrl = 0x00047F00u  Pulse after first frame sync ignored           */
/* Frame length 128 words                                                 */
/* Word size 8 bits                                                       */
/* srgr_ctrl   = 0xc4000400u  Frame period 1024 clkg cycles               */
/* Frame sync pulse width 4 clkg cycles                                   */
/* mcr_ctrl = 0x00070001u Disable all but selected channels               */
/* enable blk 0 for receive                                               */
/* enable blk 1 for transmit                                              */
/* rcer_ctrl   = 0x00000001u   enable partitian A chan 0 for rec          */
/* xcer_ctrl   = 0x00010000u   Enable Partitian B chan 0 for xmit         */
/* pcr_ctrl = 0x0000030cu   Receive/Transmit driven externally            */
/*----------------------------------------------------------------------- */
mcbsp_init(1, 0x00002000u, 0x00047F00u, 0x00047F00u, 0xC4000400u,
0x00070001u, 0x00000001u, 0x00010000u, 0x00000030u );
```

MCBSP_IO_DISABLE

Take selected port out of general-purpose I/O mode

Syntax

#include <mcbsp.h>
#define **MCBSP_IO_DISABLE(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_IO_DISABLE restores the selected port to McBSP operational mode from the general-purpose I/O mode. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ----------------------------------------------------------*/
/* Take MCBSP 0 out of general purpose I/O                   */
/* mode                                                      */
/* ----------------------------------------------------------*/
#include <mcbsp.h>
MCBSP_IO_DISABLE(0);
```

MCBSP_IO_ENABLE

Place selected port in general-purpose I/O mode

Syntax

#include <mcbsp.h>
#define **MCBSP_IO_ENABLE(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_IO_ENABLE places the selected port in general purpose I/O mode. DR and CLKS are general purpose input pins; DX is a general purpose output pin. FS(R/X), CLK(R/X) are general-purpose I/O pins. MCBSP_IO_ENABLE uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ----------------------------------------------------------*/
/* Place MCBSP 0 port in general purpose I/O                 */
/* mode                                                      */
/* ----------------------------------------------------------*/
#include <mcbsp.h>
MCBSP_IO_ENABLE (0);
```

MCBSP_LOOPBACK_DISABLE

Take selected serial port out of internal loopback mode

Syntax

#include <mcbsp.h>
#define **MCBSP_LOOPBACK_DISABLE(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_LOOPBACK_DISABLE takes the selected serial port out of internal loopback mode by resetting the DLB bit in the appropriate serial port configuration register. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ---------------------------------------------------------*/
/* Disable loopback mode mode for MCBSP 1                   */
/* ---------------------------------------------------------*/
#include <mcbsp.h>
MCBSP_LOOPBACK_DISABLE(1);
```

MCBSP_LOOPBACK_ENABLE

Place selected serial port in internal loopback mode

Syntax

#include <mcbsp.h>
#define **MCBSP_LOOPBACK_ENABLE(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_LOOPBACK_ENABLE places the selected serial port in internal loopback mode by setting the DLB bit in the appropriate serial port configuration register. During DLB mode, the DR, FSR, and CLKR are internally connected to the DX, FSX and CLKX pins, respectively. MCBSP_LOOPBACK_ENABLE uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ---------------------------------------------------------*/
/* Place MCBSP 1 digital loopback mode                      */
/* ---------------------------------------------------------*/
#include <mcbsp.h>
MCBSP_LOOPBACK_ENABLE(1);
```

MCBSP_MCR_ADDR

Select MCBSP multichannel control register address

Syntax

#include <mcbsp.h>
#define **MCBSP_MCR_ADDR(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_MCR_ADDR returns the address of the MCBSP multichannel control register for the selected channel. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ----------------------------------------------------------*/
/* Get address of MCBSP 1 multi-channel                      */
/* control register                                          */
/* ----------------------------------------------------------*/
#include <mcbsp.h>
unsigned int *ptr = (unsigned int *)MCBSP_MCR_ADDR(1);
```

MCBSP_PCR_ADDR

Select MCBSP pin control register address

Syntax

#include <mcbsp.h>
#define **MCBSP_PCR_ADDR(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_PCR_ADDR returns the address of the MCBSP pin control register for the selected channel. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ----------------------------------------------------------*/
/* Get address of MCBSP 1 pin control reg                    */
/* ----------------------------------------------------------*/
#include <mcbsp.h>
unsigned int *ptr = (unsigned int *)MCBSP_PCR_ADDR(1);
```

MCBSP_RCER_ADDR

Select MCBSP receive channel enable register address

Syntax

#include <mcbsp.h>
#define **MCBSP_RCER_ADDR(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_RCER_ADDR returns the address of the MCBSP receive channel enable register for the selected channel. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ----------------------------------------------------------*/
/*Get address of MCBSP 1 receive channel enable reg */
/* ----------------------------------------------------------*/
#include <mcbsp.h>
unsigned int *ptr = (unsigned int *)MCBSP_RCER_ADDR(1);
```

MCBSP_RCR_ADDR

Select MCBSP receive control register address

Syntax

#include <mcbsp.h>
#define **MCBSP_RCR_ADDR(**port_no**)**

Defined in

mcbsp.h as a macro

Description

This macro function returns the address of the MCBSP receive control register for the selected channel. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ----------------------------------------------------------*/
/*Get address of MCBSP 1 receive control register   */
/* ----------------------------------------------------------*/
#include <mcbsp.h>
unsigned int *ptr = (unsigned int *)MCBSP_RCR_ADDR(1);
```

MCBSP_READ *Read data receive register (DRR)*

Syntax

#include <mcbsp.h>
#define **MCBSP_READ(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_READ returns the value in the selected port's data receive register (DRR). It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
#include <mcbsp.h>
/*---------------------------------------------------------------------------*/
/* Read value from data, receive register of serial port 1                   */
/*---------------------------------------------------------------------------*/
unsigned int data = MCBSP_READ (1);
```

MCBSP_RRDY *Return state of RRDY bit for selected port*

Syntax

#include <mcbsp.h>
#define **MCBSP_RRDY(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_RRDY returns the value of the RRDY bit for the selected port. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/*-------------------------------------------------------------------------------*/
/* Poll status of RDDY for MCBSP 0                                               */
/*-------------------------------------------------------------------------------*/
#include <mcbsp.h>
while(!(MCBSP_RRDY(0));
```

MCBSP_RX_RESET

Reset multichannel buffered serial port

Syntax

#include <mcbsp.h>
#define **MCBSP_RX_RESET(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_RX_RESET resets the receiver of the selected port. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
#include <mcbsp.h>
/* ----------------------------------------------------------*/
/* Resets receive side of serial port 1                      */
/* ----------------------------------------------------------*/
MCBSP_RX_RESET (1);
```

MCBSP_SAMPLE_RATE_ENABLE

Enable sample rate generator for selected port

Syntax

#include <mcbsp.h>
#define **MCBSP_SAMPLE_RATE_ENABLE(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_SAMPLE_RATE_ENABLE enables the sample rate generator for the selected port. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ----------------------------------------------------------*/
/* Enable SRGR for MCBSP 1                                   */
/* ----------------------------------------------------------*/
#include <mcbsp.h>
MCBSP_SAMPLE_RATE_ENABLE(1);
```

MCBSP_SAMPLE_RATE_RESET

Reset/disable sample rate generator for selected port

Syntax

#include <mcbsp.h>
#define **MCBSP_SAMPLE_RATE_RESET(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_SAMPLE_RATE_RESET resets/disables the sample rate generator for the selected port. The FSG and CLKG signals are driven inactive low. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ----------------------------------------------------------*/
/* Disable SRGR for MCBSP 1                                  */
/* ----------------------------------------------------------*/
#include <mcbsp.h>
MCBSP_SAMPLE_RATE_RESET(1);
```

MCBSP_SPCR_ADDR

Select MCBSP serial port control register address

Syntax

#include <mcbsp.h>
#define **MCBSP_SPCR_ADDR(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_SPCR_ADDR returns the address of the MCBSP serial port control register for the selected channel. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ----------------------------------------------------------*/
/* Get address of MCBSP 0 sample rate generator reg */
/* ----------------------------------------------------------*/
#include <mcbsp.h>
unsigned int *ptr = (unsigned int *)MCBSP_SPCR_ADDR
(0);
```

MCBSP_SRGR_ADDR

Select MCBSP sample rate generator register address

Syntax

#include <mcbsp.h>
#define **MCBSP_SRGR_ADDR(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_SRGR_ADDR returns the address of the MCBSP sample rate generator register for the selected channel. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ------------------------------------------------------*/
/* Get address of MCBSP 0 sample rate generator reg */
/* ------------------------------------------------------*/
#include <mcbsp.h>
unsigned int *ptr = (unsigned int *)MCBSP_SRGR_ADDR
(0);
```

MCBSP_TX_RESET

Reset multichannel buffered serial port

Syntax

#include <mcbsp.h>
#define **MCBSP_TX_RESET(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_TX_RESET resets the transmitter of the selected port. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
#include <mcbsp.h>
/* ------------------------------------------------------*/
/* Reset transmit side of serial port 1                  */
/* ------------------------------------------------------*/
MCBSP_TX_RESET(1);
```

MCBSP_WRITE — *Write data transmit register (DXR)*

Syntax

#include <mcbsp.h>
#define **MCBSP_WRITE (**port_no,data**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_WRITE loads data into the selected port's data transmit register (DXR). It uses the following parameters:

- port_no: indicates the selected port (0,1)
- data: data to be transmitted

Example

```
#include <mcbsp.h>
MCBSP_WRITE (1, (unsigned int) 'A');
```

MCBSP_XCER_ADDR — *Select MCBSP transmit channel enable register address*

Syntax

#include <mcbsp.h>
#define **MCBSP_XCER_ADDR(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_XCER_ADDR returns the address of the MCBSP transmit channel enable register for the selected channel. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/*-------------------------------------------------------------------------*/
/* Get address of MCBSP 1 transmit control register                        */
/*-------------------------------------------------------------------------*/
#include <mcbsp.h>
unsigned int xcer = MCBSP_XCER_ADDR(1);
```

MCBSP_XCR_ADDR

Select MCBSP transmit control register address

Syntax

#include <mcbsp.h>
#define **MCBSP_XCR_ADDR(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_XCR_ADDR returns the address of the MCBSP transmit control register for the selected channel. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ----------------------------------------------------------*/
/* Get address of MCBSP 1 transmit control register */
/* ----------------------------------------------------------*/
#include <mcbsp.h>
unsigned int *ptr = (unsigned int *)MCBSP_XCR_ADDR (1);
```

MCBSP_XRDY

Return state of XRDY bit for selected port

Syntax

#include <mcbsp.h>
#define **MCBSP_XRDY(**port_no**)**

Defined in

mcbsp.h as a macro

Description

MCBSP_XRDY returns the value of the XRDY bit for the selected port. It uses the following parameter:

- port_no: indicates the selected port (0,1)

Example

```
/* ----------------------------------------------------------*/
/* Poll status of XRDY for MCBSP 0                           */
/* ----------------------------------------------------------*/
#include <mcbsp.h>
while (!(MCBSP_XRDY(0));
```

REG_READ — *Read memory-mapped register*

Syntax

#include <regs.h>
#define **REG_READ(**addr**)**

Defined in

regs.h as a macro

Description

REG_READ returns the contents of the register whose address is given by addr. It uses the following parameter:

- addr: address of memory-mapped register to be read.

Example

```
/*-------------------------------------------------------------------------- */
/* Include timer.h to access timer control register                         */
/* Definitions, macros, and functions, including timer.h                    */
/* Automatically includes regs.h                                            */
/*-------------------------------------------------------------------------- */
#include <timer.h>
unsigned int count = REG_READ(TIMER0_COUNTER_ADDR);
/*Get value of timer counter register                                       */
```

REG_WRITE — *Write to memory-mapped register*

Syntax

#include <regs.h>
#define **REG_WRITE(**addr,val**)**

Defined in

regs.h as a macro

Description

REG_WRITE writes val to the register whose address is given by addr. It uses the following parameters:

- addr: address of memory-mapped register to write to
- val: value to write to register

Example

```
/*-------------------------------------------------------------------------- */
/* Include timer.h to access timer control register                         */
/* Definitions, macros and functions, including timer.h                     */
/* Automatically includes, regs.h                                           */
/*-------------------------------------------------------------------------- */
#include <timer.h>
REG_WRITE(TIMER0_PERIOD_ADDR,256);
/* Set TIMER 0 period register                                              */
```

RESET_BIT *Reset/clear bit*

Syntax

#include <regs.h>
#define **RESET_BIT(**addr,bit**)**

Defined in

regs.h as a macro

Description

RESET_BIT resets (clears) the indicated bit in the register whose address is given by addr. It uses the following parameters:

- addr: address of memory-mapped register/word to access
- bit: bit to reset/clear

Example

```
/*-------------------------------------------------------------------------*/
/* Include hpi.h to gain access to Host Port Interface                     */
/* Register definitions, macros, and functions                             */
/* Including hpi.h, automatically includes, regs.h                         */
/*-------------------------------------------------------------------------*/
RESET_BIT (HPIC_ADDR, DSPINT);

/*Clear HOST to DSP/DMA interrupt bit                                      */
```

RESET_FIELD *Resets/clears bits in bit field*

Syntax

#include <regs.h>
#define **RESET_FIELD(**addr,bit,length**)**

Defined in

regs.h as a macro

Description

RESET_FIELD clears the bits within the indicated bit field in the register whose address is given by addr. It uses the following parameters:

- addr: address of memory-mapped register
- bit: starting bit of field to reset/clear
- length: length of bit field (number of bits)

Example

```
/*-------------------------------------------------------------------------*/
/* Include intr.h to access interrupt handling macros                      */
/* Functions and register definitions. Including                           */
/* intr.h automatically includes, regs.h                                   */
/*-------------------------------------------------------------------------*/
#include <intr.h>

RESET_FIELD(INTR_MULTIPLEX_LOW_ADDR,INTSEL6, INTSEL_SZ);
/* Reset interrupt selector interrupt 6                                    */
```

RESET_REG_BIT

Reset/clear bit in named register

Syntax

#include <regs.h>
#define **RESET_REG_BIT(**reg,bit**)**

Defined in

regs.h as a macro

Description

RESET_REG_BIT sets the indicated bit in the named register. It uses the following parameters:

- reg: register name (CSR, MCBSP0_SPCR, TIMER0_CTRL, DMA_GCTRL)
- bit: bit in register to reset/clear

Example

```
/*---------------------------------------------------------------------------*/
/* Include intr.h to access interrupt handling macros                        */
/* Functions and register definitions. Including                             */
/* intr.h automatically includes, regs.h                                     */
/*---------------------------------------------------------------------------*/
#include <intr.h>
RESET_REG_BIT(IFR,IF4);
/* Reset/clear interrupt flag for CPU interrupt #4                           */
```

SDRAM_INIT

Initialize SDRAM

Syntax

#include <emif.h>
#define **SDRAM_INIT( )**

Defined in

emif.h as a macro

Description

SDRAM_INIT causes the EMIF to perform the necessary functions to initialize SDRAM if any of the CE spaces are configured for SDRAM.

Example

```
#include <emif.h>
SDRAM_INIT( );
```

SDRAM_REFRESH_DISABLE

Disable SDRAM refresh cycles

Syntax

#include <emif.h>
#define **SDRAM_REFRESH_DISABLE( )**

Defined in

emif.h as a macro

Description

SDRAM_REFRESH_DISABLE disables SDRAM refresh cycles for all CE spaces that specify an SDRAM memory type in the MTYPE field of the associated CE space control register.

Example

```
#include <emif.h>
SDRAM_REFRESH_DISABLE( );
```

SDRAM_REFRESH_ENABLE

Enable SDRAM refresh cycles

Syntax

#include <emif.h>
#define **SDRAM_REFRESH_ENABLE( )**

Defined in

emif.h as a macro

Description

SDRAM_REFRESH_ENABLE enables SDRAM refresh cycles for all CE spaces that specify an SDRAM memory type in the MTYPE field of the associated CE space control register.

Example

```
#include <emif.h>
SDRAM_REFRESH_ENABLE( );
```

SDRAM_REFRESH_PERIOD

Set SDRAM refresh period

Syntax

#include <emif.h>
#define **SDRAM_REFRESH_PERIOD(**val**)**

Defined in

emif.h as a macro

Description

SDRAM_REFRESH_PERIOD sets the SDRAM refresh period by assigning the val parameter to the PERIOD field of the SDRAM timing register. It uses the following parameter:

- ❏ val: value to set SDRAM refresh period in SDRAM timing register. Val indicates the number of CLKOUT2 periods (1/2 the CPU clock rate) in the refresh cycle.

Example

```
#include <emif.h>
SDRAM_REFRESH_PERIOD(2048);
```

SET_BIT

Set bit

Syntax

#include <regs.h>
#define **SET_BIT(**addr,bit**)**

Defined in

regs.h as a macro

Description

SET_BIT sets the indicated bit to 1 in the register whose address is given by addr. It uses the following parameter:

- ❏ addr: address of memory-mapped register
- ❏ bit: bit in register to set

Example

```
/* HPI.H automatically includes regs.h              */
#include <hpi.h>
SET_BIT(HPIC_ADDR,HRDY);
```

SET_REG — *Set value of named register*

Syntax

#include <regs.h>
#define **SET_REG(**reg,val**)**

Defined in

regs.h as a macro

Description

SET_REG sets the value of the named register. It uses the following parameters:

- reg: name of memory-mapped register
- val: value to assign to register

Example

```
#include <timer.h>
/* timer.h automatically includes regs.h          */
SET_REG(TIMER0_PERIOD,256);
```

SET_REG_BIT — *Set bit in named register*

Syntax

#include <regs.h>
#define **SET_REG_BIT(**reg,bit**)**

Defined in

regs.h as a macro

Description

SET_REG_BIT sets the indicated bit in the named register.

- reg: name of peripheral/memory-mapped control register
- bit: bit in register to be set

Example

```
#include <hpi.h>
/* hpi.h automatically includes regs.h            */
SET_REG_BIT(HPIC,HRDY);
```

TIMER_AVAILABLE

Checks for availability of timer channel

Syntax

#include <timer.h>
#define **TIMER_AVAILABLE(**chan**)**

Defined in

timer.h as a macro

Description

TIMER_AVAILABLE checks the status of the specified channel to see if it is available for use. It uses the following parameter:

- chan: channel selector (0,1)

Example

```
#include <timer.h>
if (TIMER_AVAILABLE(0)) timer_init(0, 0x1, 0xffff);
```

TIMER_CLK_EXTERNAL

Select external clock source timer for channel

Syntax

#include <timer.h>
#define **TIMER_CLK_EXTERNAL(**chan**)**

Defined in

timer.h as a macro

Description

TIMER_CLK_EXTERNAL selects the external clock source for the specified timer channel. It uses the following parameter:

- chan: channel selector (0,1)

Example

```
#include <timer.h>
TIMER_CLK_EXTERNAL(1);  /* TIMER 1 driven by external clock source */
```

TIMER_CLK_INTERNAL

Select internal clock source

Syntax

#include <timer.h>
#define **TIMER_CLK_INTERNAL(**chan**)**

Defined in

timer.h as a macro

Description

TIMER_CLK_INTERNAL selects the internal clock source for the specified timer channel. It uses the following parameter:

- ❏ chan: channel selector (0,1)

Example

```
#include <timer.h>
TIMER_CLK_INTERNAL(0);  /* TIMER 0 driven by internal clock source */
```

TIMER_COUNTER_ADDR

Select timer counter register address

Syntax

#include <timer.h>
#define **TIMER_COUNTER_ADDR(**chan**)**

Defined in

timer.h as a macro

Description

TIMER_COUNTER_ADDR returns the address of the timer counter register for the selected channel. It uses the following parameter:

- ❏ chan: channel selector (0,1)

Example

```
/*-------------------------------------------------------------------------*/
/* Get address of timer 1 counter register                                 */
/*-------------------------------------------------------------------------*/
#include <timer.h>
unsigned int *ptr = (unsigned int *)TIMER_COUNTER_ADDR (1);
```

TIMER_CTRL_ADDR

Select timer control register address

Syntax

#include <timer.h>
#define **TIMER_CTRL_ADDR(**chan**)**

Defined in

timer.h as a macro

Description

TIMER_CTRL_ADDR returns the address of the timer control register for the selected channel. It uses the following parameter:

- chan: channel selector (0,1)

Example

```
/* ------------------------------------------------------*/
/* get address of timer 0 control register              */
/* ------------------------------------------------------*/
#include <timer.h>
unsigned int *timer_ctrl = (unsigned int*)
TIMER_CTRL_ADDR(0);
```

timer_delay

Delay for specified number of timer periods

Syntax

#include <timer.h>
#define **timer_delay(**short num_timer_periods**)**

Defined in

timer.c as a callable function

Description

Delays for specified number of clock ticks. The argument is used to set the value of the timer period register. It uses the following parameter:

- num_timer_periods: value to set timer period

Example

```
#include <timer.h>
timer_delay(100);   /* delay 100 clock cycles*/
```

TIMER_GET_COUNT

Return value of timer counter register

Syntax

#include <timer.h>
#define **TIMER_GET_COUNT(**chan**)**

Defined in

timer.h as a macro

Description

TIMER_GET_COUNT returns the value of the timer counter register for the specified channel It uses the following parameter:

- chan: channel selector (0,1)

Example

```
#include <timer.h>
unsigned int count = TIMER_GET_COUNT(0);
```

TIMER_GET_PERIOD

Return value of timer period register

Syntax

#include <timer.h>
#define **TIMER_GET_PERIOD(**chan**)**

Defined in

timer.h as a macro

Description

TIMER_GET_PERIOD returns the value of the timer period register for the specified timer channel. It uses the following parameter:

- chan: channel selector (0,1)

Example

```
#include <timer.h>
unsigned int period = TIMER_GET_PERIOD(0);
```

TIMER_GET_TSTAT

Return value of status bit

Syntax

#include <timer.h>
#define **TIMER_GET_TSTAT(**chan**)**

Defined in

timer.h as a macro

Description

TIMER_GET_TSTAT returns the value of the timer status bit, TSTAT, in the timer control register of the specified channel. TSTAT reflects the value of the timer output. It uses the following parameter:

- chan: channel selector (0,1)

Example

```
#include <timer.h>
unsigned int timer_output = TIMER_GET_TSTAT(0);
```

TIMER_INIT

Initialize timer registers

Syntax

#include <timer.h>
#define **TIMER_INIT(**chan,ctrl,per,cnt**)**

Defined in

timer.h as a macro

Description

TIMER_INIT initializes the timer control, timer period, and timer counter registers to the specified values for the specified channel. It uses the following parameters:

- chan: channel selector (0,1)
- ctrl: mask to set timer control register
- per: value to set timer period register
- cnt: value to set timer counter register

Example

```
#include <timer.h>
/*------------------------------------------------------------------------- */
/* Configure timer 0 as timer pin                                           */
/*------------------------------------------------------------------------- */
      TIMER_INIT (0, 0x1, 0xFFFF, 0);
/* Configure timer 0 as timer pin and set timer period to 0xFFFF     */
```

TIMER_MODE_SELECT

Select timer mode

Syntax

#include <timer.h>
#define **TIMER_MODE_SELECT(**chan,mode**)**

Defined in

timer.h as a macro

Description

TIMER_MODE_SELECT sets the mode of the timer to either pulse or clock mode. Macro defines in timer.h TIMER_CLOCK_MODE and TIMER_PULSE_MODE should be used for the mode argument. TIMER_MODE_SELECT uses the following parameters:

- ❏ chan: channel selector (0,1)
- ❏ mode: mode to set for specified timer (TIMER_CLOCK_MODE, TIMER_PULSE_MODE)

Example

```
#include <timer.h>
TIMER_CODE_SELECT(1,TIMER_CLOCK_MODE);
```

TIMER_PERIOD_ADDR

Select timer period register address

Syntax

#include <timer.h>
#define **TIMER_PERIOD_ADDR(**chan**)**

Defined in

timer.h as a macro

Description

TIMER_PERIOD_ADDR returns the address of the timer period register for the selected channel. It uses the following parameters:

- ❏ chan: channel selector (0,1)

Example

```
/* ------------------------------------------------------*/
/* get address of timer 1 period register               */
/* ------------------------------------------------------*/
#include  <timer.h>
unsigned int *ptr = (unsigned int *)TIMER_PERIOD_ADDR
(1);
```

TIMER_READ — *Read value in timer counter register*

Syntax

#include <timer.h>
#define **TIMER_READ(**chan**)**

Defined in

timer.h as a macro

Description

TIMER_READ returns the value to the selected channel's timer counter register. It uses the following parameter:

- chan: channel selector (0,1)

Example

```
#include <timer.h>
/* ------------------------------------------------------*/
/* Read timer counter value                              */
/* ------------------------------------------------------*/
unsigned int x = TIMER_READ (1);
```

TIMER_RESET — *Reset timer registers*

Syntax

#include <timer.h>
#define **TIMER_RESET(**chan**)**

Defined in

timer.h as a macro

Description

TIMER_RESET resets the timer control register, timer period register, and timer counter register of the specified channel to their default values of 0. It uses the following parameter:

- chan: channel selector (0,1)

Example

```
#include <timer.h>
/* ------------------------------------------------------*/
/* Reset timer 1                                         */
/* ------------------------------------------------------*/
TIMER_RESET (1);
```

TIMER_RESUME

Resume timer

Syntax

#include <timer.h>
#define **TIMER_RESUME(**chan**)**

Defined in

timer.h as a macro

Description

TIMER_RESUME negates (sets) the HLD bit in the specified channel's timer control register, which enables the timer to proceed from its previous state. It uses the following parameter:

- chan: channel selector (0,1)

Example

```
#include <timer.h>
TIMER_RESUME(0);
```

TIMER_SET_COUNT

Set timer counter register

Syntax

#include <timer.h>
#define **TIMER_SET_COUNT(**chan,val**)**

Defined in

timer.h as a macro

Description

TIMER_SET_COUNT sets the value of the timer counter register for the specified channel. It uses the following parameters:

- chan: channel selector (0,1)
- val: value to write to the counter register

Example

```
#include <timer.h>
TIMER_SET_COUNT(0,256);
```

TIMER_SET_PERIOD

Set timer period register

Syntax

#include <timer.h>
#define **TIMER_SET_PERIOD(**chan,val**)**

Defined in

timer.h as a macro

Description

TIMER_SET_PERIOD sets the value of the timer period register for the specified timer channel to the value given by val. It uses the following parameter:

- chan: channel selector (0,1)
- val: value to set register

Example

```
#include <timer.h>
TIMER_SET_PERIOD(1,1024);
```

TIMER_START

Start timer

Syntax

#include <timer.h>
#define **TIMER_START(**chan**)**

Defined in

timer.h as a macro

Description

TIMER_START sets both the GO and HLD bits in the timer control register of the specified channel. This causes the timer counter to be reset to 0 and counting to be enabled. It uses the following parameters:

- chan: channel selector (0,1)

Example

```
#include <timer.h>
/* ------------------------------------------------------*/
/* Start timer 1                                         */
/* ------------------------------------------------------*/
TIMER_START (1);
```

TIMER_STOP — *Stop timer*

Syntax

#include <timer.h>
#define **TIMER_STOP(**chan**)**

Defined in

timer.h as a macro

Description

TIMER_STOP asserts (clears) the HLD bit in the specified channel's timer control register which disables counting. It uses the following parameters:

- ❏ chan: channel selector (0,1)

Example

```
#include <timer.h>
/* ------------------------------------------------------*/
/* Stop timer                                            */
/* ------------------------------------------------------*/
TIMER_STOP (0);
```

TINP_GET — *Get value of TINP pin*

Syntax

#include <timer.h>
#define **TINP_GET(**chan**)**

Defined in

timer.h as a macro

Description

TINP_GET returns the value on the TINP pin. This macro returns a valid value regardless of the whether the TINP pin is being used as an external clock source or as a general-purpose input. It uses the following parameters:

- ❏ chan: channel selector (0,1)

Example

```
/* ------------------------------------------------------*/
/* Get value of timer 0 TINP pin                         */
/* ------------------------------------------------------*/
#include <timer.h>
unsigned int tinp = TINP_GET (0);
```

TOUT_ASSERT

Asserts 1 on TOUT pin

Syntax

#include <timer.h>
#define **TOUT_ASSERT(**chan**)**

Defined in

timer.h as a macro

Description

TOUT_ASSERT writes a 1 to the TOUT pin of the specified channel. It uses the following parameter:

- chan: channel selector (0,1)

Example

```
#include <timer.h>
TOUT_ASSERT(0);       /*Write a 1 to the TOUT pin of
timer 0 */
```

TOUT_DISABLE

Configure TOUT as timer pin

Syntax

#include <timer.h>
#define **TOUT_DISABLE(**chan**)**

Defined in

timer.h as a macro

Description

TOUT_DISABLE configures TOUT as a timer pin for the specified timer channel. It uses the following parameter:

- chan: channel selector (0,1)

Example

```
#include <timer.h>
TOUT_DISABLE(1);
```

TOUT_ENABLE

Configure TOUT pin as general-purpose input

Syntax

#include <timer.h>
#define **TOUT_ENABLE(**chan**)**

Defined in

timer.h as a macro

Description

TOUT_ENABLE sets the TOUT pin of the selected channel to be a general-purpose output by setting the FUNC bit in the timer control register. When TOUT is enabled, TOUT_ASSERT and TOUT_NEGATE may be used to control the signal level on the pin. TOUT_ENABLE uses the following parameter:

- chan: channel selector (0,1)

Example

```
#include <timer.h>
TOUT_ENABLE (0);
```

TOUT_NEGATE — *Asserts 0 on TOUT pin*

Syntax

#include <timer.h>
#define **TOUT_NEGATE(**chan**)**

Defined in

timer.h as a macro

Description

TOUT_NEGATE writes a 0 to the TOUT pin of the specified channel. It uses the following parameter:

- ❏ chan: channel selector (0,1)

Example

```
#include <timer.h>
TOUT_NEGATE (0);      /* Write a 0 to the TOUT pin of
timer 0 */
```

TOUT_VAL — *Assigns value to TOUT*

Syntax

#include <timer.h>
#define **TOUT_VAL(**chan,val**)**

Defined in

timer.h as a macro

Description

TOUT_VAL writes a value to the TOUT pin of the specified channel. It uses the following parameters:

- ❏ chan: channel selector (0,1)
- ❏ val: value to be written

Example

```
#include <timer.h>
TOUT_VAL(0,1);  /* Write a 1 to TOUT pin of timer 0 */
```

For:dbrune
Printed on:Fri, Apr 10, 1998 04:52:33
From book:Appendices
Document:68114k
Last saved on:Fri, Apr 10, 1998 04:39:47

Appendix A

Source File Listing

This appendix provides the entire code listing for each source file contained in the ’C6x peripheral support library. However, the code provided to you may be a more recent version than the one that is listed here. For more information on the source files and the macros and functions that comprise them, see Chapter 2.

Topic **Page**

A.1 Header Files

The following sections contain the header files that are included in the ’C6x peripheral support library.

A.1.1 cache.h

```
/*****************************************************************************/
/*  CACHE.H – TMS320C6x Peripheral Support Library Program Cache Support      */
/*                                                                            */
/*     This file provides the header for program memory cache support.        */
/*                                                                            */
/*                                                                            */
/*  MACRO FUNCTIONS:                                                          */
/*     CACHE_ENABLE() – Enables program memory cache                          */
/*     CACHE_DISALBE()– Disables program memory cache (memory-mapped)         */
/*     CACHE_FREEZE() – Freeze program memory cache                           */
/*     CACHE_BYPASS() – Bypass program memory cache                           */
/*     CACHE_FLUSH()  – Flushes program memory cache                          */
/*     CACHE_CLEAR()  – Clears program memory cache                           */
/*     IDLE()         – Put processor in IDLE state                           */
/*                                                                            */
/*  FUNCTIONS:                                                                */
/*    None.                                                                   */
/*                                                                            */
/*****************************************************************************/
#ifndef _CACHE_H_
#define _CACHE_H_
/*---------------------------------------------------------------------------*/
/* INCLUDES                                                                  */
/*---------------------------------------------------------------------------*/
#include ”regs.h”
/*---------------------------------------------------------------------------*/
/* DEFINES AND MACROS                                                        */
/*---------------------------------------------------------------------------*/
#define CACHE_ENABLE() \
        {CSR &= 0xffffff1f; CSR |= 0x40;}
#define CACHE_DISABLE() \
        {CSR &= 0xffffff1f;}
#define CACHE_FREEZE() \
        {CSR &= 0xffffff1f; CSR |= 0x60;}
#define CACHE_BYPASS() \
        {CSR &= 0xffffff1f; CSR |= 0x80;}
#define CACHE_FLUSH() \
        {CACHE_DISABLE(); CACHE_ENABLE();}
#define IDLE() \
        { asm(”\tidle”);}
/*---------------------------------------------------------------------------*/
/* GLOBAL VARIABLES                                                          */
/*---------------------------------------------------------------------------*/
/*---------------------------------------------------------------------------*/
/* FUNCTIONS                                                                 */
/*---------------------------------------------------------------------------*/
#endif
```

A.1.2 dma.h

```
/************************************************************************/
/*  DMA.H – TMS320C6x Peripheral Support Library DMA Support            */
/*                                                                      */
/*     This file provides the header for the DSP DMA controller support.*/
/*                                                                      */
/*                                                                      */
/*  MACRO FUNCTIONS:                                                    */
/*     DMA_START         – Manually start selected channel              */
/*     DMA_AUTO_START()  – Begins DMA autoinitialization on selected channel */
/*     DMA_STOP          – Stop selected channel                        */
/*     DMA_PAUSE         – Pause selected channel                       */
/*     DMA_RSYNC_CLR()   – Clear DMA read sync bit                      */
/*     DMA_WSYNC_CLR()   – Clear DMA write sync bit                     */
/*     DMA_RSYNC_SET()   – Set DMA read sync bit                        */
/*     DMA_WSYNC_SET()   – Set DMA write sync bit                       */
/*                                                                      */
/*  FUNCTIONS:                                                          */
/*     dma_init()        – Initialize channel specific control registers*/
/*     dma_global_init() – Initialize global control registers          */
/*     dma_reset()       – Resets all four DMA channels to default      */
/*                                                                      */
/************************************************************************/
#ifndef _DMA_H_
#define _DMA_H_
#include ”regs.h”
#ifdef _INLINE
#define __INLINE static inline
#else
#define __INLINE
#endif
/************************************************************************/
/***************************** DMA REGISTERS ****************************/
/* Register Addresses */
/* DMA Channel 0 */
#define DMA0_PRIMARY_CTRL_ADDR       0x01840000   /* DMA 0 PRI CTRL REG*/
#define DMA0_SECONDARY_CTRL_ADDR     0x01840008   /* DMA 0 SEC CTRL REG*/
#define DMA0_SRC_ADDR_ADDR           0x01840010   /* DMA 0 SRC ADDR    */
#define DMA0_DEST_ADDR_ADDR          0x01840018   /* DMA 0 DEST ADDR   */
#define DMA0_XFER_COUNTER_ADDR       0x01840020   /* DMA 0 TRANS CNT   */
/* DMA Channel 1 */
#define DMA1_PRIMARY_CTRL_ADDR       0x01840040   /* DMA 1 PRI CTRL REG*/
#define DMA1_SECONDARY_CTRL_ADDR     0x01840048   /* DMA 1 SEC CTRL REG*/
#define DMA1_SRC_ADDR_ADDR           0x01840050   /* DMA 1 SRC ADDR    */
#define DMA1_DEST_ADDR_ADDR          0x01840058   /* DMA 1 DEST ADDR   */
#define DMA1_XFER_COUNTER_ADDR       0x01840060   /* DMA 1 TRANS CNT   */
/* DMA Channel 2 */
#define DMA2_PRIMARY_CTRL_ADDR       0x01840004   /* DMA 2 PRI CTRL REG*/
#define DMA2_SECONDARY_CTRL_ADDR     0x0184000C   /* DMA 2 SEC CTRL REG*/
#define DMA2_SRC_ADDR_ADDR           0x01840014   /* DMA 2 SRC ADDR    */
#define DMA2_DEST_ADDR_ADDR          0x0184001C   /* DMA 2 DEST ADDR   */
#define DMA2_XFER_COUNTER_ADDR       0x01840024   /* DMA 2 TRANS CNT   */
```

```
/* DMA Channel 3 */
#define DMA3_PRIMARY_CTRL_ADDR       0x01840044   /* DMA 3 PRI CTRL REG*/
#define DMA3_SECONDARY_CTRL_ADDR     0x0184004C   /* DMA 3 SEC CTRL REG*/
#define DMA3_SRC_ADDR_ADDR           0x01840054   /* DMA 3 SRC ADDR    */
#define DMA3_DEST_ADDR_ADDR          0x0184005C   /* DMA 3 DEST ADDR   */
#define DMA3_XFER_COUNTER_ADDR       0x01840064   /* DMA 3 TRANS CNT   */
/* DMA Auxiliary Control */
#define DMA_AUX_CTRL_ADDR            0x01840070   /* AUX DMA CTRL REG  */
/* DMA Global Registers */
#define DMA_GCR_A_ADDR               0x01840028   /* GLOBAL CNT RELOADA*/
#define DMA_GCR_B_ADDR               0x0184002c   /* GLOBAL CNT RELOADB*/
#define DMA_GNDX_A_ADDR              0x01840030   /* GLOBAL INDEX REG A*/
#define DMA_GNDX_B_ADDR              0x01840034   /* GLOBAL INDEX REG B*/
#define DMA_GADDR_A_ADDR             0x01840038   /* GLOBAL ADDR REG A */
#define DMA_GADDR_B_ADDR             0x0184003C   /* GLOBAL ADDR REG B */
#define DMA_GADDR_C_ADDR             0x01840068   /* GLOBAL ADDR REG C */
#define DMA_GADDR_D_ADDR             0x0184006C   /* GLOBAL ADDR REG D */
#define DMA_GCTRL_ADDR               0x01840070   /* GLOBAL CTRL REG   */
/* Register Contents */
/* DMA Channel 0 */
#define DMA0_PRIMARY_CTRL    *(volatile unsigned int *)DMA0_PRIMARY_CTRL_ADDR
#define DMA0_SECONDARY_CTRL  *(volatile unsigned int *)DMA0_SECONDARY_CTRL_ADDR
#define DMA0_SRC_ADDR        *(volatile unsigned int *)DMA0_SRC_ADDR_ADDR
#define DMA0_DEST_ADDR       *(volatile unsigned int *)DMA0_DEST_ADDR_ADDR
#define DMA0_XFER_COUNTER    *(volatile unsigned int *)DMA0_XFER_COUNTER_ADDR
/* DMA Channel 1 */
#define DMA1_PRIMARY_CTRL    *(volatile unsigned int *)DMA1_PRIMARY_CTRL_ADDR
#define DMA1_SECONDARY_CTRL  *(volatile unsigned int *)DMA1_SECONDARY_CTRL_ADDR
#define DMA1_SRC_ADDR        *(volatile unsigned int *)DMA1_SRC_ADDR_ADDR
#define DMA1_DEST_ADDR       *(volatile unsigned int *)DMA1_DEST_ADDR_ADDR
#define DMA1_XFER_COUNTER    *(volatile unsigned int *)DMA1_XFER_COUNTER_ADDR
/* DMA Channel 2 */
#define DMA2_PRIMARY_CTRL    *(volatile unsigned int *)DMA2_PRIMARY_CTRL_ADDR
#define DMA2_SECONDARY_CTRL  *(volatile unsigned int *)DMA2_SECONDARY_CTRL_ADDR
#define DMA2_SRC_ADDR        *(volatile unsigned int *)DMA2_SRC_ADDR_ADDR
#define DMA2_DEST_ADDR       *(volatile unsigned int *)DMA2_DEST_ADDR_ADDR
#define DMA2_XFER_COUNTER    *(volatile unsigned int *)DMA2_XFER_COUNTER_ADDR
/* DMA Channel 3 */
#define DMA3_PRIMARY_CTRL    *(volatile unsigned int *)DMA3_PRIMARY_CTRL_ADDR
#define DMA3_SECONDARY_CTRL  *(volatile unsigned int *)DMA3_SECONDARY_CTRL_ADDR
#define DMA3_SRC_ADDR        *(volatile unsigned int *)DMA3_SRC_ADDR_ADDR
#define DMA3_DEST_ADDR       *(volatile unsigned int *)DMA3_DEST_ADDR_ADDR
#define DMA3_XFER_COUNTER    *(volatile unsigned int *)DMA3_XFER_COUNTER_ADDR
/* DMA Global Registers */
#define DMA_GCR_A            *(volatile unsigned int *)DMA_GCR_A_ADDR
#define DMA_GCR_B            *(volatile unsigned int *)DMA_GCR_B_ADDR
#define DMA_GNDX_A           *(volatile unsigned int *)DMA_GNDX_A_ADDR
#define DMA_GNDX_B           *(volatile unsigned int *)DMA_GNDX_B_ADDR
#define DMA_GADDR_A          *(volatile unsigned int *)DMA_GADDR_A_ADDR
#define DMA_GADDR_B          *(volatile unsigned int *)DMA_GADDR_B_ADDR
#define DMA_GADDR_C          *(volatile unsigned int *)DMA_GADDR_C_ADDR
#define DMA_GADDR_D          *(volatile unsigned int *)DMA_GADDR_D_ADDR
#define DMA_GCTRL            *(volatile unsigned int *)DMA_GCTRL_ADDR
```

```
#define DMA_REG_SELECT(base,chan) \
        (((unsigned int)(base)) + (((chan) & 1) * 0x40) + (((chan) & 2) * 2))
#define DMA_PRIMARY_CTRL_ADDR(chan) \
        (DMA_REG_SELECT(DMA0_PRIMARY_CTRL_ADDR,chan))
#define DMA_SECONDARY_CTRL_ADDR(chan) \
        (DMA_REG_SELECT(DMA0_SECONDARY_CTRL_ADDR,chan))
#define DMA_SRC_ADDR_ADDR(chan) \
        (DMA_REG_SELECT(DMA0_SRC_ADDR_ADDR,chan))
#define DMA_DEST_ADDR_ADDR(chan) \
        (DMA_REG_SELECT(DMA0_DEST_ADDR_ADDR,chan))
#define DMA_XFER_COUNTER_ADDR(chan) \
        (DMA_REG_SELECT(DMA0_XFER_COUNTER_ADDR,chan))
/** DMA Register Bitfields **/
/** Primary Control Register **/
#define START                  0
#define START_SZ               2
#define STATUS                 2
#define STATUS_SZ              2
#define SRC_DIR                4
#define SRC_DIR_SZ             2
#define DST_DIR                6
#define DST_DIR_SZ             2
#define ESIZE                  8
#define ESIZE_SZ               2
#define SPLIT                 10
#define SPLIT_SZ               2
#define CNT_RELOAD            12
#define INDEX                 13
#define RSYNC                 14
#define RSYNC_SZ               5
#define WSYNC                 19
#define WSYNC_SZ               5
#define PRI                   24
#define TCINT                 25
#define FS                    26
#define EMOD                  27
#define SRC_RELOAD            28
#define SRC_RELOAD_SZ          2
#define DST_RELOAD            30
#define DST_RELOAD_SZ          2
/** Secondary Control Register **/
#define SX_COND                0
#define SX_IE                  1
#define FRAME_COND             2
#define FRAME_IE               3
#define LAST_COND              4
#define LAST_IE                5
#define BLOCK_COND             6
#define BLOCK_IE               7
#define RDROP_COND             8
#define RDOPR_IE               9
#define WDROP_COND            10
#define WDROP_IE              11
```

```
#define RSYNC_STAT            12
#define RSYNC_CLR             13
#define WSYNC_STAT            14
#define WSYNC_CLR             15
#define DMAC_EN               16
#define DMAC_EN_SZ             3
/* DMA Channel Transfer Counter Register */
#define ELEMENT_COUNT          0
#define ELEMENT_COUNT_SZ      16
#define FRAME_COUNT           16
#define FRAME_COUNT_SZ        16
/* DMA Global Count Reload Register Bits */
#define ELEMENT_COUNT_RELOAD        0
#define ELEMENT_COUNT_RELOAD_SZ    16
#define FRAME_COUNT_RELOAD         16
#define FRAME_COUNT_RELOAD_SZ      16
/* DMA Global Index Register Bits */
#define ELEMENT_INDEX               0
#define ELEMENT_INDEX_SZ           16
#define FRAME_INDEX                16
#define FRAME_INDEX_SZ             16
/* DMA Global Address Register Bits */
#define SPLIT_ADDRESS               3
#define SPLIT_ADDRESS_SZ           29
/* DMA Auxiliary Control Register Bits */
#define CH_PRI                      0
#define CH_PRI_SZ                   4
#define AUXPRI                      4
/*-------------------------------------------------------------------------*/
/* DEFINES                                                                 */
/*-------------------------------------------------------------------------*/
#define DMA_CH0              0x00
#define DMA_CH1              0x01
#define DMA_CH2              0x02
#define DMA_CH3              0x03
/** BITFIELD VALUES **/
/* DMA Channel Primary Control Register bitfield values */
/* START */
#define DMA_STOP_VAL        0x00
#define DMA_START_VAL       0x01
#define DMA_PAUSE_VAL       0x02
#define DMA_AUTO_START_VAL  0x03
/* SRC DIR, DST DIR */
#define DMA_ADDR_NO_MOD      0x00
#define DMA_ADDR_INC         0x01
#define DMA_ADDR_DEC         0x02
#define DMA_ADDR_INDX        0x03
/* Synchronization Event Numbers */
#define SEN_NONE             0x00
#define SEN_TINT0            0x01
#define SEN_TINT1            0x02
#define SEN_SD_INT           0x03
#define SEN_EXT_INT4         0x04
```

```
#define SEN_EXT_INT5         0x05
#define SEN_EXT_INT6         0x06
#define SEN_EXT_INT7         0x07
#define SEN_DMA_INT0         0x08
#define SEN_DMA_INT1         0x09
#define SEN_DMA_INT2         0x0A
#define SEN_DMA_INT3         0x0B
#define SEN_XEVT0            0x0C
#define SEN_REVT0            0x0D
#define SEN_XEVT1            0x0E
#define SEN_REVT1            0x0F
#define SEN_DSPINT           0x10
/* ESIZE defines */
#define DMA_ESIZE32          0x00
#define DMA_ESIZE16          0x01
#define DMA_ESIZE8           0x02
/* PRI defines */
#define DMA_CPU_PRI          0x00
#define DMA_DMA_PRI          0x01
/* SPLIT mode operation defines */
#define DMA_SPLIT_DIS        0x00
#define DMA_SPLIT_GARA       0x01
#define DMA_SPLIT_GARB       0x02
#define DMA_SPLIT_GARC       0x03
/* CNT RELOAD defines */
#define DMA_CNT_RELOADA      0x00
#define DMA_CNT_RELOADB      0x01
/* INDEX defines */
#define DMA_INDX2            0x00
#define DMA_INDX3            0x01
/* EMULATION MODE response defines */
#define DMA_NO_EM_HALT       0x00
#define DMA_EM_HALT          0x01
/* SRC/DST RELOAD defines */
#define DMA_RELOAD_NONE      0x00
#define DMA_RELOAD_GARB      0x01
#define DMA_RELOAD_GARC      0x02
#define DMA_RELOAD_GARD      0x03
/* DMA Channel Primary Control Register bitfield values */
/* DMAC EN Control */
#define DMAC_LO              0x00
#define DMAC_HI              0x01
#define DMAC_RSYNC_STAT      0x02
#define DMAC_WSYNC_STAT      0x03
#define DMAC_FRAME_COND      0x04
#define DMAC_BLOCK_COND      0x05
/*-----------------------------------------------------------------------------*/
/* MACRO DEFINITIONS                                                           */
/*-----------------------------------------------------------------------------*/
#define DMA_START(chan) \
        LOAD_FIELD(DMA_PRIMARY_CTRL_ADDR(chan),DMA_START_VAL,START,START_SZ)
#define DMA_AUTO_START(chan) \
        LOAD_FIELD(DMA_PRIMARY_CTRL_ADDR(chan),DMA_AUTO_START_VAL,START,START_SZ)
```

```
#define DMA_STOP(chan) \
        LOAD_FIELD(DMA_PRIMARY_CTRL_ADDR(chan),DMA_STOP_VAL,START,START_SZ)
#define DMA_PAUSE(chan) \
        LOAD_FIELD(DMA_PRIMARY_CTRL_ADDR(chan),DMA_PAUSE_VAL,START,START_SZ)
#define DMA_RSYNC_CLR(chan) \
        LOAD_FIELD(DMA_SECONDARY_CTRL_ADDR(chan),2,RSYNC_STAT,2)
#define DMA_WSYNC_CLR(chan) \
        LOAD_FIELD(DMA_SECONDARY_CTRL_ADDR(chan),2,WSYNC_STAT,2)
#define DMA_RSYNC_SET(chan) \
        LOAD_FIELD(DMA_SECONDARY_CTRL_ADDR(chan),1,RSYNC_STAT,2)

#define DMA_WSYNC_SET(chan) \
        LOAD_FIELD(DMA_SECONDARY_CTRL_ADDR(chan),1,WSYNC_STAT,2)

/*-------------------------------------------------------------------------*/
/* FUNCTION DEFINITIONS                                                    */
/*-------------------------------------------------------------------------*/
/***************************************************************************/
/* DMA_INIT -  Initialize channel specific control registers.              */
/*                                                                         */
/*     This function is responsible for setting the primary control register, */
/*     secondary control register, source address, destination address and */
/*     transfer count for the specified DMA channel.                       */
/*                                                                         */
/***************************************************************************/
__INLINE void    dma_init(              /*RET: OK or ERROR (invalid channel)  */
             unsigned short channel    /*IN: DMA channel number              */
            ,unsigned int pri_ctrl     /*IN: Value to set primary cntl reg   */
            ,unsigned int sec_ctrl     /*IN: Value to set sec cntl reg       */
            ,unsigned int src_addr     /*IN: Value to set source addr reg    */
            ,unsigned int dst_addr     /*IN: Value to set dest addr reg      */
            ,unsigned int trans_ctr    /*IN: Value to set transfer counter   */
               );
/***************************************************************************/
/* DMA_GLOBAL_INIT -  Initialize global control registers.                 */
/*                                                                         */
/*     This function is responsible for setting the DMA global control     */
/*     register, global count reload registers A & B, global index registers  */
/*     A & B, and global address registers A-D.                            */
/*                                                                         */
/***************************************************************************/
__INLINE void     dma_global_init(       /*RET: VOID function                 */
             unsigned int gcr          /*IN:Value for Global Control Register*/
            ,unsigned int gcra         /*IN:Value for Global Cnt Reload Reg A*/
            ,unsigned int gcrb         /*IN:Value for Global Cnt Reload Reg B*/
            ,unsigned int gndxa        /*IN:Value for Global Idx Reg A       */
            ,unsigned int gndxb        /*IN:Value for Global Idx Reg B       */
            ,unsigned int gaddra       /*IN:Value for Global Addr Reg A      */
            ,unsigned int gaddrb       /*IN:Value for Global Addr Reg B      */
            ,unsigned int gaddrc       /*IN:Value for Global Addr Reg C      */
            ,unsigned int gaddrd       /*IN:Value for Global Addr Reg D      */
                      );
```

```
/*****************************************************************************/
/* DMA_RESET –  Reset DMA channels.                                          */
/*                                                                           */
/*     This function resets the specified DMA channels by initializing       */
/*     channel control registers to their default values                     */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
__INLINE void     dma_reset(void);                                          */
#if _INLINE
static inline void   dma_init(
             unsigned short channel    /*IN: DMA channel number             */
            ,unsigned int pri_ctrl     /*IN: Value to set primary cntl reg  */
            ,unsigned int sec_ctrl     /*IN: Value to set sec cntl reg      */
            ,unsigned int src_addr     /*IN: Value to set source addr reg   */
            ,unsigned int dst_addr     /*IN: Value to set dest addr reg     */
            ,unsigned int trans_ctr    /*IN: Value to set transfer counter  */
                )
{
  *((unsigned int *)(DMA_PRIMARY_CTRL_ADDR(channel)))   = pri_ctrl;
  *((unsigned int *)(DMA_SECONDARY_CTRL_ADDR(channel))) = sec_ctrl;
  *((unsigned int *)(DMA_SRC_ADDR_ADDR(channel)))       = src_addr;
  *((unsigned int *)(DMA_DEST_ADDR_ADDR(channel)))      = dst_addr;
  *((unsigned int *)(DMA_XFER_COUNTER_ADDR(channel)))   = trans_ctr;
  return;
}
static inline void dma_global_init(
             unsigned int gcr          /*IN:Value for Global Control Register*/
            ,unsigned int gcra         /*IN:Value for Global Cnt Reload Reg A*/
            ,unsigned int gcrb         /*IN:Value for Global Cnt Reload Reg B*/
            ,unsigned int gndxa        /*IN:Value for Global Idx Reg A       */
            ,unsigned int gndxb        /*IN:Value for Global Idx Reg B       */
            ,unsigned int gaddra       /*IN:Value for Global Addr Reg A      */
            ,unsigned int gaddrb       /*IN:Value for Global Addr Reg B      */
            ,unsigned int gaddrc       /*IN:Value for Global Addr Reg C      */
            ,unsigned int gaddrd)      /*IN:Value for Global Addr Reg D      */
{
  DMA_GCR_A     = gcra;
  DMA_GCR_B     = gcrb;
  DMA_GNDX_A    = gndxa;
  DMA_GNDX_B    = gndxb;
  DMA_GADDR_A   = gaddra;
  DMA_GADDR_B   = gaddrb;
  DMA_GADDR_C   = gaddrc;
  DMA_GADDR_D   = gaddrd;
  return;
}
static inline void dma_reset( void )
{
  int chan;
  for (chan= 0;chan < DMA_CH3; chan++)
```

```
  {
    *(unsigned int *)(DMA_PRIMARY_CTRL_ADDR(chan))        = 0x00;
    *(unsigned int *)(DMA_SECONDARY_CTRL_ADDR(chan))      = 0x00;
    *(unsigned int *)(DMA_SRC_ADDR_ADDR(chan))            = 0x00;
    *(unsigned int *)(DMA_DEST_ADDR_ADDR(chan))           = 0x00;
    *(unsigned int *)(DMA_XFER_COUNTER_ADDR(chan))        = 0x00;
  }
  DMA_GCR_A     = 0x00;
  DMA_GCR_B     = 0x00;
  DMA_GNDX_A    = 0x00;
  DMA_GNDX_B    = 0x00;
  DMA_GADDR_A   = 0x00;
  DMA_GADDR_B   = 0x00;
  DMA_GADDR_C   = 0x00;
  DMA_GADDR_D   = 0x00;
  DMA_GCTRL     = 0x00;
}
#endif /* _INLINE */
#ifdef __INLINE
#undef __INLINE
#endif
#endif /* _DMA_H_ */
```

A.1.3 emif.h

```
/*****************************************************************************/
/*  EMIF.H – TMS320C6x Peripheral Support Library EMIF Support               */
/*                                                                           */
/*     This file provides the header for the DSP's EMIF support.             */
/*     on the TMS320C6x DSP.                                                 */
/*                                                                           */
/*  MACRO FUNCTIONS:                                                         */
/*    SDRAM_REFRESH_ENABLE()  – Enable SDRAM refresh cycles                  */
/*    SDRAM_REFRESH_DISABLE() – Disable SDRAM refresh cycles                 */
/*    SDRAM_REFRESH_PERIOD()  – Assigns refresh period for SDRAM             */
/*    SDRAM_INIT()            – Perform initialization sequence for SDRAM    */
/*    EMIF_GET_MAP_MODE()     – Return value of MAP bit in EMIF global ctrl  */
/*                                                                           */
/*  FUNCTIONS:                                                               */
/*    emif_init()          – Sets all EMIF registers to parameter values     */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
#ifndef _EMIF_H_
#define _EMIF_H_
#include "regs.h"        /* EMIF Register Addresses and bitfield definitions */
definitions */
#if _INLINE
#define __INLINE static inline
#else
#define __INLINE
#endif
/*****************************************************************************/
/****************************** EMIF REGISTERS *******************************/
#define EMIF_GCTRL_ADDR         0x01800000
#define EMIF_CE0_CTRL_ADDR      0x01800008
#define EMIF_CE1_CTRL_ADDR      0x01800004
#define EMIF_CE2_CTRL_ADDR      0x01800010
#define EMIF_CE3_CTRL_ADDR      0x01800014
#define EMIF_SDRAM_CTRL_ADDR    0x01800018
#define EMIF_SDRAM_REF_ADDR     0x0180001C
#define EMIF_GCTRL              (*(volatile unsigned int *)EMIF_GCTRL_ADDR)
#define EMIF_CE0_CTRL           (*(volatile unsigned int *)EMIF_CE0_CTRL_ADDR)
#define EMIF_CE1_CTRL           (*(volatile unsigned int *)EMIF_CE1_CTRL_ADDR)
#define EMIF_CE2_CTRL           (*(volatile unsigned int *)EMIF_CE2_CTRL_ADDR)
#define EMIF_CE3_CTRL           (*(volatile unsigned int *)EMIF_CE3_CTRL_ADDR)
#define EMIF_SDRAM_CTRL         (*(volatile unsigned int *)EMIF_SDRAM_CTRL_ADDR)
#define EMIF_SDRAM_REF          (*(volatile unsigned int *)EMIF_SDRAM_REF_ADDR)
/* EMIF Global Control Register Bits */
#define MAP                 0
#define RBTR8               1
#define SSCRT               2
#define CLK2EN              3
#define CLK1EN              4
#define SSCEN               5
#define SDCEN               6
```

```
#define NOHOLD             7
#define HOLDA              8
#define HOLD               9
#define ARDY              10
#define CLK2INV           12
#define SDCINV            13
/* EMIF CE0/1/2/3 Control Register Bits */
#define READ_HOLD          0
#define READ_HOLD_SZ       2
#define MTYPE              4
#define MTYPE_SZ           3
#define READ_STROBE        8
#define READ_STROBE_SZ     6
#define TA                14
#define TA_SZ              2
#define READ_SETUP        16
#define READ_SETUP_SZ      4
#define WRITE_HOLD        20
#define WRITE_HOLD_SZ      2
#define WRITE_STROBE      22
#define WRTIE_STROBE_SZ    6
#define WRITE_SETUP       28
#define WRITE_SETUP_SZ     4
/* EMIF SDRAM Control Register Bits */
#define TRC               12
#define TRC_SZ             4
#define TRP               16
#define TRP_SZ             4
#define TRCD              20
#define TRCD_SZ            4
#define INIT              24
#define RFEN              25
#define SDWID             26
/* EMIF SDRAM Timing Register Bits */
#define PERIOD             0
#define PERIOD_SZ         12
#define COUNTER           12
#define COUNTER_SZ        12
/* EMIF Global Control Register Bitfield Values */
/* EMIF CE Space Control Register Bitfield Values */
#define MTYPE_8ROM          0x00     /* 8 bit wide ROM                       */
#define MTYPE_16ROM         0x01     /* 16 bit wide ROM                      */
#define MTYPE_32ASYNC       0x02     /* 32 bit asynchronous interface        */
#define MTYPE_32SDRAM       0x03     /* 32 bit SDRAM                         */
#define MTYPE_32SBSRAM      0x04     /* 32 bit SBSRAM                        */
/*----------------------------------------------------------------------------*/
/* MACRO FUNCTIONS                                                            */
/*----------------------------------------------------------------------------*/
#define SDRAM_REFRESH_ENABLE() \
        SET_BIT(EMIF_SDRAM_CTRL_ADDR,RFEN)
#define SDRAM_REFRESH_DISABLE() \
        RESET_BIT(EMIF_SDRAM_CTRL_ADDR,RFEN)
```

```
#define SDRAM_REFRESH_PERIOD(val) \
        LOAD_FIELD(EMIF_SDRAM_REF_ADDR,val)
#define SDRAM_INIT() \
        SET_BIT(EMIF_SDRAM_CTRL_ADDR,INIT)
#define EMIF_GET_MAP_MODE() \
        GET_BIT(EMIF_GCTRL_ADDR,MAP)
__INLINE
void emif_init(unsigned int g_ctrl,
               unsigned int ce0_ctrl,
               unsigned int ce1_ctrl,
               unsigned int ce2_ctrl,
               unsigned int ce3_ctrl,
               unsigned int sdram_ctrl,
               unsigned int sdram_refresh
              );
#if _INLINE
__INLINE
void emif_init(unsigned int g_ctrl,
               unsigned int ce0_ctrl,
               unsigned int ce1_ctrl,
               unsigned int ce2_ctrl,
               unsigned int ce3_ctrl,
               unsigned int sdram_ctrl,
               unsigned int sdram_refresh
              )
{
  REG_WRITE(EMIF_GCTRL_ADDR, g_ctrl);
  REG_WRITE(EMIF_CE0_CTRL_ADDR, ce0_ctrl);
  REG_WRITE(EMIF_CE1_CTRL_ADDR, ce1_ctrl);
  REG_WRITE(EMIF_CE2_CTRL_ADDR, ce2_ctrl);
  REG_WRITE(EMIF_CE3_CTRL_ADDR, ce3_ctrl);
  REG_WRITE(EMIF_SDRAM_CTRL_ADDR, sdram_ctrl);
  REG_WRITE(EMIF_SDRAM_REF_ADDR, sdram_refresh);
}
#endif  /* _INLINE  */
#ifdef __INLINE
#undef __INLINE
#endif
#endif  /* _EMIF_H_ */
```

A.1.4 hpi.h

```
/*****************************************************************************/
/*  HPI.H – TMS320C6x Peripheral Support Library EMIF Support                */
/*                                                                           */
/*     This file provides the header for the DSP's HPI support.              */
/*     Interface on the TMS320C6x DSP.                                       */
/*                                                                           */
/*  MACRO FUNCTIONS:                                                         */
/*    HPI_SET_HINT()                                                         */
/*    HPI_RESET_DSPINT()                                                     */
/*    HPI_GET_HINT()                                                         */
/*    HPI_GET_DSPINT()                                                       */
/*                                                                           */
/*  FUNCTIONS:                                                               */
/*    None.                                                                  */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
#ifndef _HPI_H_
#define _HPI_H_
/*---------------------------------------------------------------------------*/
/* INCLUDES                                                                  */
/*---------------------------------------------------------------------------*/
#include ”regs.h”
/*---------------------------------------------------------------------------*/
/* DEFINES AND MACROS                                                        */
/*---------------------------------------------------------------------------*/
/*****************************************************************************/
/***************************** HPI REGISTERS *********************************/
#define HPIC_ADDR          0x01880000                /* HPI Ctrl Reg Addr   */
#define HPIC               *(volatile unsigned int *)HPIC_ADDR  /* HPI Ctrl   */
/* HPIC Register bits */
#define HWOB               0
#define DSPINT             1
#define HINT               2
#define HRDY               3
#define FETCH              4
#define HPI_SET_HINT() \
        REG_WRITE(HPIC_ADDR,(REG_READ(HPIC_ADDR) & 0XFFFFFFFD) | 4))
#define HPI_RESET_DSPINT() \
        REG_WRITE(HPIC_ADBR,(REG_READ(HPIC_ADDR) & 0XFFFFFFFB) | 2))
#define HPI_GET_HINT() \
        (GET_BIT(HPIC_ADDR,HINT))
#define HPI_GET_DSPINT() \
        (GET_BIT(HPIC_ADDR,DSPINT))
/*---------------------------------------------------------------------------*/
/* GLOBAL VARIABLES                                                          */
/*---------------------------------------------------------------------------*/
/*---------------------------------------------------------------------------*/
/* FUNCTIONS                                                                 */
/*---------------------------------------------------------------------------*/
#endif
```

A.1.5 intr.h

```
/*****************************************************************************/
/*  INTR.H – TMS320C5x Peripheral Support Library Interrupt Support          */
/*                                                                           */
/*     This file provides the header for the DSP's interupt support.         */
/*     on the TMS320C6x DSP.                                                 */
/*                                                                           */
/*  MACRO FUNCTIONS:                                                         */
/*     INTR_GLOBAL_ENABLE()   – Enable global interrupts (GIE)               */
/*     INTR_GLOBAL_DISABLE()  – Disable global interrupts (GIE)              */
/*     INTR_ENABLE()          – Enable interrupt (set bit in IER)            */
/*     INTR_DISABLE()         – Disable interrupt (clear bit in IER)         */
/*     INTR_CHECK_FLAG()      – Check interrupt bit in IFR                   */
/*     INTR_SET_FLAG()        – Set interrupt by writing to ISR bit          */
/*     INTR_CLR_FLAG()        – Clear interrupt by writin to ICR bit         */
/*     INTR_SET_MAP()         – Map CPU interrupt to interrupt selector      */
/*     INTR_GET_ISN()         – Get ISN of selected interrupt                */
/*     INTR_MAP_RESET()       – Reset interrupt multiplexor map to defaults  */
/*     INTR_EXT_POOLARITY()   – Assign external interrupt's polarity         */
/*                                                                           */
/*  FUNCTIONS:                                                               */
/*     intr_reset()           – Reset interrupt registers to default values  */
/*     intr_init()            – Interrupt initialization                     */
/*     intr_isn()             – Assign ISN to CPU interrupt                  */
/*     intr_get_cpu_intr()    – Return CPU interrupt assigned to ISN         */
/*     intr_map()             – Place ISN in interrupt multiplexor register  */
/*                                                                           */
/*****************************************************************************/
#ifndef _INTR_H_
#define _INTR_H_
#if _INLINE
#define __INLINE static inline
#else
#define __INLINE
#endif
/*---------------------------------------------------------------------------*/
/* INCLUDES                                                                  */
/*---------------------------------------------------------------------------*/
#include ”regs.h”
/*---------------------------------------------------------------------------*/
/* DEFINES AND MACROS                                                        */
/*---------------------------------------------------------------------------*/
/*****************************************************************************/
/******************** INTERRUPT SELECTOR REGISTERS ***************************/
#define INTR_MULTIPLEX_HIGH_ADDR     0x019c0000
#define INTR_MULTIPLEX_LOW_ADDR      0x019c0004
#define EXTERNAL_INTR_POL_ADDR       0x019c0008
#define INTSEL4             0
#define INTSEL_SZ           4
#define INTSEL5             5
#define INTSEL6            10
#define INTSEL7            16
```

```
#define INTSEL8            21
#define INTSEL9            26
#define INTSEL10            0
#define INTSEL11            5
#define INTSEL12           10
#define INTSEL13           16
#define INTSEL14           21
#define INTSEL15           26
/* External Interrupt Polarity Register */
#define XIP4                0
#define XIP5                1
#define XIP6                2
#define XIP7                3
/* CPU Interrupt Numbers       */
#define CPU_INT_RST          0x00
#define CPU_INT_NMI          0x01
#define CPU_INT_RSV1         0x02
#define CPU_INT_RSV2         0x03
#define CPU_INT4             0x04
#define CPU_INT5             0x05
#define CPU_INT6             0x06
#define CPU_INT7             0x07
#define CPU_INT8             0x08
#define CPU_INT9             0x09
#define CPU_INT10            0x0A
#define CPU_INT11            0x0B
#define CPU_INT12            0x0C
#define CPU_INT13            0x0D
#define CPU_INT14            0x0E
#define CPU_INT15            0x0F
/* Interrupt Selection Numbers */
#define ISN_DSPINT           0x00
#define ISN_TINT0            0x01
#define ISN_TINT1            0x02
#define ISN_SD_INT           0x03
#define ISN_EXT_INT4         0x04
#define ISN_EXT_INT5         0x05
#define ISN_EXT_INT6         0x06
#define ISN_EXT_INT7         0x07
#define ISN_DMA_INT0         0x08
#define ISN_DMA_INT1         0x09
#define ISN_DMA_INT2         0x0A
#define ISN_DMA_INT3         0x0B
#define ISN_XINT0            0x0C
#define ISN_RINT0            0x0D
#define ISN_XINT1            0x0E
#define ISN_RINT1            0x0F
#define IML_SEL              0x00       /* Interrupt Multiplexor Low Select   */
#define IMH_SEL              0x01       /* Interrupt Multiplexor High Select  */
#define IML_RESET_VAL        0x250718A4
#define IMH_RESET_VAL        0x08202D4B
```

```
/*-------------------------------------------------------------------------*/
/* MACRO FUNCTIONS                                                         */
/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/
/* INTR_GLOBAL_ENABLE – enables all masked interrupts by setting the GIE   */
/*              bit (bit 0) in the CSR                                     */
/*-------------------------------------------------------------------------*/
#define INTR_GLOBAL_ENABLE() \
        SET_REG_BIT(CSR, GIE)
/*-------------------------------------------------------------------------*/
/* INTR_GLOBAL_DISABLE – disables all masked interrupts by clearing the GIE*/
/*               (bit 0) in the CSR.                                       */
/*-------------------------------------------------------------------------*/
#define INTR_GLOBAL_DISABLE() \
        CLR_REG_BIT(CSR, GIE)
/*-------------------------------------------------------------------------*/
/* INTR_ENABLE – enable interrupt by setting flag in IER                   */
/*-------------------------------------------------------------------------*/
#define INTR_ENABLE(bit) \
        SET_REG_BIT(IER,bit)
/*-------------------------------------------------------------------------*/
/* INTR_DISABLE – disable interrupt by clearing flag in IER                */
/*-------------------------------------------------------------------------*/
#define INTR_DISABLE(bit) \
        RESET_REG_BIT(IER,bit)
/*-------------------------------------------------------------------------*/
/* INTR_CHECK_FLAG – checks status of indicated interrupt bit in IFR       */
/*-------------------------------------------------------------------------*/
#define INTR_CHECK_FLAG(bit) \
        (IFR & MASK_BIT(bit) ? 1 : 0)
/*-------------------------------------------------------------------------*/
/* INTR_SET_FLAG – manually sets indicated interrupt by writing to ISR     */
/*-------------------------------------------------------------------------*/
#define INTR_SET_FLAG(bit) \
        (ISR |= MASK_BIT(bit))
/*-------------------------------------------------------------------------*/
/* INTR_CLR_FLAG – manually clears indicated interrupt by writing 1 to ICR */
/*-------------------------------------------------------------------------*/
#define INTR_CLR_FLAG(bit) \
        (ICR |= MASK_BIT(bit))
/*-------------------------------------------------------------------------*/
/* INTR_SET_MAP  – maps a cpu interrupt specified by intr to the interrupt src*/
/*            specified by val.  Sel is used to select between the low and */
/*            high interrupt_multiplexor registers.                        */
/*-------------------------------------------------------------------------*/
#define INTR_SET_MAP(intsel,val,sel) \
        (sel ? LOAD_FIELD(INTR_MULTIPLEX_HIGH_ADDR,val,intsel,INTSEL_SZ) : \
               LOAD_FIELD(INTR_MULTIPLEX_LOW_ADDR, val,intsel,INTSEL_SZ ))
```

```
/*-------------------------------------------------------------------------*/
/* INTR_GET_ISN – returns the ISN value in the selected Interrupt Multiplexor */
/*                register for the interrupt selected by intsel            */
/*-------------------------------------------------------------------------*/
#define INTR_GET_ISN(intsel,sel) \
        (sel ? GET_FIELD(INTR_MULTIPLEX_HIGH_ADDR,intsel,INTSEL_SZ) : \
               GET_FIELD(INTR_MULTIPLEX_LOW_ADDR, intsel, INTSEL_SZ))
/*-------------------------------------------------------------------------*/
/* INTR_MAP_RESET – resets the interrupt multiplexor maps to their default val*/
/*-------------------------------------------------------------------------*/
#define INTR_MAP_RESET() \
        {CONTENTS_OF(INTR_MULTIPLEX_HIGH_ADDR) = IMH_RESET_VAL; \
         CONTENTS_OF(INTR_MULTIPLEX_LOW_ADDR)  = IML_RESET_VAL; }
/*-------------------------------------------------------------------------*/
/* INTR_EXT_POLARITY – assigns external interrupt external priority.       */
/*                   val = 0 (normal), val = 1 (inverted)                  */
/*-------------------------------------------------------------------------*/
#define INTR_EXT_POLARITY(bit,val) \
        (val ? SET_BIT(EXTERNAL_INTR_POL_ADDR,bit) : \
               CLEAR_BIT(EXTERNAL_INTR_POL_ADDR,bit))
/*-------------------------------------------------------------------------*/
/* GLOBAL VARIABLES                                                        */
/*-------------------------------------------------------------------------*/
extern unsigned int istb;
/*-------------------------------------------------------------------------*/
/* FUNCTIONS                                                               */
/*-------------------------------------------------------------------------*/
extern void interrupt c_int00(void);
void intr_reset(void);
void intr_init(void);
void intr_hook(void (*fp)(void),int intr_num);
__INLINE void intr_map(int cpu_intr,int isn);
__INLINE int intr_isn(int cpu_intr);
__INLINE int intr_get_cpu_intr(int isn);
#if _INLINE
/* intr_map() – place isn value in Interrupt Multiplexer Register in INTSEL   */
/*              field indicated by cpu_intr.                               */

static inline void intr_map(int cpu_intr,int isn)
{
  int intsel;
  int sel;
  if (cpu_intr > CPU_INT9)
    sel=1;
  else
    sel= 0;
  intsel= ((cpu_intr – CPU_INT4) * INTSEL_SZ) – (sel * 30);
  if (intsel > INTSEL6)
    intsel++;
  INTR_SET_MAP(intsel,isn,sel);
  return;
}
```

```
/* intr_isn() – return isn in interrupt selector corresponding to cpu_intr    */
static inline int intr_isn(int cpu_intr)
{
  int intsel;
  int sel;
  if (cpu_intr > CPU_INT9)
    sel= 1;
  else
    sel= 0;
  intsel= ((cpu_intr – CPU_INT4) * INTSEL_SZ) – (sel * 30);
  if (intsel > INTSEL6)
    intsel++;
  return(INTR_GET_ISN(intsel,sel));
}
/* intr_get_cpu_intr() – return cpu interrupt corresponding to isn in         */
/*                       interrupt selecter register.  If the isn is not      */
/*                       mapped, return –1                                    */
static inline int intr_get_cpu_intr(int isn)
{
  int i;
  for (i= CPU_INT4;i<=CPU_INT15;i++)
  {
    if (intr_get_isn(i) == isn)
      return(i);
  }
  return(–1);
}
#endif /* _INLINE */
#undef __INLINE
#endif /* _INTR_H_ */
```

A.1.6 mcbsp.h

```
/*****************************************************************************/
/*   MCBSP.H – TMS320C6x Peripheral Support Library McBSP Support            */
/*                                                                           */
/*     This file provides the header for the DSP's McBSP support.            */
/*                                                                           */
/*  MACRO FUNCTIONS:                                                         */
/*     MCBSP_BYTES_PER_WORD()  – Return bytes required for word length       */
/*     MMCBSP_ENABLE()         – Enables McBSP transit, receive or both      */
/*     MCBSP_TX_RESET()        – Reset McBSP transmitter                     */
/*     MCBSP_RX_RESET()        – Reset McBSP receiver                        */
/*     MCBSP_READ()            – Read data value from McBSP receive          */
/*     MCBSP_WRITE()           – Write data value from McBSP transmit        */
/*     MCBSP_IO_ENABLE()       – Place McBSP in general-purpose I/O mode     */
/*     MCBSP_IO_DISABLE()      – Remove McBSP from general-purpose I/O mode  */
/*     MCBSP_FRAME_SYNC_ENABLE() – Enables McBSP frame sync generation logic */
/*     MCBSP_FRAME_SYNC_RESET()  – Resets McBSP frame sync generation logic  */
/*     MCBSP_SAMPLE_RATE_ENABLE() – Enables McBSP sample rate generator      */
/*     MCBSP_SAMPLE_RATE_RESET() – Resets McBSP sample rate generator        */
/*     MCBSP_RRDY()           – Returns McBSP receiver ready status          */
/*     MCBSP_XRDY()           – Returns McBSP transmitter ready status       */
/*     MCBSP_LOOPBACK_ENABLE()– Configures McBSP in digital loopback         */
/*     MCBSP_LOOPBACK_DISABLE()– Disables McBSP digital loopback mode        */
/*                                                                           */
/*                                                                           */
/*  FUNCTIONS:                                                               */
/*     mcbsp_init()            – Initializes McBSP registers                 */
/*                                                                           */
/*                                                                           */
/*  GLOBAL VARIABLES                                                         */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
#ifndef _MCBSP_H_
#define _MCBSP_H_
#ifdef _INLINE
#define __INLINE static inline
#else
#define __INLINE
#endif
/*---------------------------------------------------------------------------*/
/* INCLUDES                                                                  */
/*---------------------------------------------------------------------------*/
#include ”regs.h”
/*---------------------------------------------------------------------------*/
/* DEFINES AND MACROS                                                        */
/*---------------------------------------------------------------------------*/
/*****************************************************************************/
/***************************** MCSP REGISTERS ********************************/
/* Multi–Channel Buffered Serial Port Control Registers & Bits               */
#define MCBSP_ADDR(port)         (0x018C0000 + ((port) * 0x40000))
#define MCBSP_DRR_ADDR(port)     (MCBSP_ADDR(port))
```

```
#define MCBSP_DXR_ADDR(port)     ((MCBSP_ADDR(port)) + 0x04)
#define MCBSP_SPCR_ADDR(port)    ((MCBSP_ADDR(port)) + 0x08)
#define MCBSP_RCR_ADDR(port)     ((MCBSP_ADDR(port)) + 0x0c)
#define MCBSP_XCR_ADDR(port)     ((MCBSP_ADDR(port)) + 0x10)
#define MCBSP_SRGR_ADDR(port)    ((MCBSP_ADDR(port)) + 0x14)
#define MCBSP_MCR_ADDR(port)     ((MCBSP_ADDR(port)) + 0x18)
#define MCBSP_RCER_ADDR(port)    ((MCBSP_ADDR(port)) + 0x1c)
#define MCBSP_XCER_ADDR(port)    ((MCBSP_ADDR(port)) + 0x20)
#define MCBSP_PCR_ADDR(port)     ((MCBSP_ADDR(port)) + 0x24)
#define MCBSP0_DRR               *(volatile unsigned int *)(MCBSP_DRR_ADDR(0))
#define MCBSP0_DXR               *(volatile unsigned int *)(MCBSP_DXR_ADDR(0))
#define MCBSP0_SPCR              *(volatile unsigned int *)(MCBSP_SPCR_ADDR(0))
#define MCBSP0_RCR               *(volatile unsigned int *)(MCBSP_RCR_ADDR(0))
#define MCBSP0_XCR               *(volatile unsigned int *)(MCBSP_XCR_ADDR(0))
#define MCBSP0_SRGR              *(volatile unsigned int *)(MCBSP_SRGR_ADDR(0))
#define MCBSP0_MCR               *(volatile unsigned int *)(MCBSP_MCR_ADDR(0))
#define MCBSP0_RCER              *(volatile unsigned int *)(MCBSP_RCER_ADDR(0))
#define MCBSP0_XCER              *(volatile unsigned int *)(MCBSP_XCER_ADDR(0))
#define MCBSP0_PCR               *(volatile unsigned int *)(MCBSP_PCR_ADDR(0))
#define MCBSP1_DRR               *(volatile unsigned int *)(MCBSP_DRR_ADDR(1))
#define MCBSP1_DXR               *(volatile unsigned int *)(MCBSP_DXR_ADDR(1))
#define MCBSP1_SPCR              *(volatile unsigned int *)(MCBSP_SPCR_ADDR(1))
#define MCBSP1_RCR               *(volatile unsigned int *)(MCBSP_RCR_ADDR(1))
#define MCBSP1_XCR               *(volatile unsigned int *)(MCBSP_XCR_ADDR(1))
#define MCBSP1_SRGR              *(volatile unsigned int *)(MCBSP_SRGR_ADDR(1))
#define MCBSP1_MCR               *(volatile unsigned int *)(MCBSP_MCR_ADDR(1))
#define MCBSP1_RCER              *(volatile unsigned int *)(MCBSP_RCER_ADDR(1))
#define MCBSP1_XCER              *(volatile unsigned int *)(MCBSP_XCER_ADDR(1))
#define MCBSP1_PCR               *(volatile unsigned int *)(MCBSP_PCR_ADDR(1))
/* Multi-channel Serial Port Control Register Bits                          */
#define RRST           0
#define RRDY           1
#define RFULL          2
#define RSYNC_ERR      3
#define RINTM          4
#define RINTM_SZ       2
#define CLKSTP        10
#define CLKSTP_SZ      3
#define RJUST         13
#define RJUST_SZ       2
#define DLB           15
#define XRST          16
#define XRDY          17
#define XEMPTY        18
#define XSYNC_ERR     19
#define XINTM         20
#define XINTM_SZ       2
#define GRST          22
#define FRST          23
/* Multi-channel Serial Port Pin Control Reg Bits                           */
#define CLKRP          0
#define CLKXP          1
#define FSRP           2
```

```
#define FSXP           3
#define DR_STAT        4
#define DX_STAT        5
#define CLKS_STAT      6
#define CLKRM          8
#define CLKXM          9
#define FSRM          10
#define FSXM          11
#define RIOEN         12
#define XIOEN         13
/* Multi-channel Serial Port RX & TX Ctrl Reg Bits                        */
#define RWDLEN1        5
#define RWDLEN1_SZ     3
#define RFRLEN1        8
#define RFRLEN1_SZ     7
#define RDATDLY       16
#define RDATDLY_SZ     2
#define RFIG          18
#define RCOMPAND      19
#define RCOMPAND_SZ    2
#define RWDLEN2       21
#define RWDLEN2_SZ     3
#define RFRLEN2       24
#define RFRLEN2_SZ     7
#define RPHASE        31
#define XWDLEN1        5
#define XWDLEN1_SZ     3
#define XFRLEN1        8
#define XFRLEN1_SZ     7
#define XDATDLY       16
#define XDATDLY_SZ     2
#define XFIG          18
#define XCOMPAND      19
#define XCOMPAND_SZ    2
#define XWDLEN2       21
#define XWDLEN2_SZ     3
#define XFRLEN2       24
#define XFRLEN2_SZ     7
#define XPHASE        31
/* Multi-channel Serial Port Sample Rate Gen Reg Bits                     */
#define CLKGDV         0
#define CLKGDV_SZ      8
#define FWID           8
#define FWID_SZ        8
#define FPER          16
#define FPER_SZ       12
#define FSGM          28
#define CLKSM         29
#define CLKSP         30
#define GSYNC         31
```

```
/* Multi–Channel Buffered Serial Port Control Registers & Bits                */
#define RMCM           0
#define RCBLK          2
#define RCBLK_SZ       3
#define RPABLK         5
#define RPABLK_SZ      2
#define RPBBLK         7
#define RPBBLK_SZ      2
#define XMCM          16
#define XMCM_SZ        2
#define XCBLK         18
#define XCBLK_SZ       3
#define XPABLK        21
#define XPABLK_SZ      2
#define XPBBLK        23
#define XPBBLK_SZ      2
/* Multi–channel Serial Port Rec Enable Register Bits                         */
#define RCEA0          0
#define RCEA1          1
#define RCEA2          2
#define RCEA3          3
#define RCEA4          4
#define RCEA5          5
#define RCEA6          6
#define RCEA7          7
#define RCEA8          8
#define RCEA9          9
#define RCEA10        10
#define RCEA11        11
#define RCEA12        12
#define RCEA13        13
#define RCEA14        14
#define RCEA15        15
#define RCEB0         16
#define RCEB1         17
#define RCEB2         18
#define RCEB3         19
#define RCEB4         20
#define RCEB5         21
#define RCEB6         22
#define RCEB7         23
#define RCEB8         24
#define RCEB9         25
#define RCEB10        26
#define RCEB11        27
#define RCEB12        28
#define RCEB13        29
#define RCEB14        30
#define RCEB15        31
/* Multi–channel Serial Port TX Enable Register Bits                          */
#define XCEA0          0
#define XCEA1          1
#define XCEA2          2
```

```
#define XCEA3          3
#define XCEA4          4
#define XCEA5          5
#define XCEA6          6
#define XCEA7          7
#define XCEA8          8
#define XCEA9          9
#define XCEA10        10
#define XCEA11        11
#define XCEA12        12
#define XCEA13        13
#define XCEA14        14
#define XCEA15        15
#define XCEB0         16
#define XCEB1         17
#define XCEB2         18
#define XCEB3         19
#define XCEB4         20
#define XCEB5         21
#define XCEB6         22
#define XCEB7         23
#define XCEB8         24
#define XCEB9         25
#define XCEB10        26
#define XCEB11        27
#define XCEB12        28
#define XCEB13        29
#define XCEB14        30
#define XCEB15        31
#define MCBSP_RX     1
#define MCBSP_TX     2
#define MCBSP_BOTH   3
/* CONFIGURATION REGISTER BIT and BITFIELD values */
/* Serial Port Control Register SPCR */
#define INTM_RDY            0x00     /* R/X INT driven by R/X RDY               */
#define INTM_BLOCK          0x01     /* R/X INT driven by new multichannel blk  */
#define INTM_FRAME          0x02     /* R/X INT driven by new frame sync        */
#define INTM_SYNCERR        0x03     /* R/X INT generated by R/X SYNCERR        */
#define DLB_ENABLE          0x01     /* Enable Digital Loopback Mode            */
#define DLB_DISABLE         0x00     /* Disable Digital Loopback Mode           */
#define RXJUST_RJZF         0x00     /* Receive Right Justify Zero Fill         */
#define RXJUST_RJSE         0x01     /* Receive Right Justify Sign Extend       */
#define RXJUST_LJZF         0x02     /* Receive Left Justify Zero Fill          */
/* Pin Control Register PCR */
#define CLKR_POL_RISING     0x01     /* R Data Sampled on Rising Edge of CLKR */
#define CLKR_POL_FALLING    0x00     /* R Data Sampled on Falling Edge of CLKR*/
#define CLKX_POL_RISING     0x00     /* X Data Sent on Rising Edge of CLKX    */
#define CLKX_POL_FALLING    0x01     /* X Data Sent on Falling Edge of CLKX   */
#define FSYNC_POL_HIGH      0x00     /* Frame Sync Pulse Active High          */
#define FSYNC_POL_LOW       0x01     /* Frame Sync Pulse Active Low           */
#define CLK_MODE_EXT        0x00     /* Clock derived from external source    */
#define CLK_MODE_INT        0x01     /* Clock derived from internal source    */
```

```
#define FSYNC_MODE_EXT      0x00     /* Frame Sync derived from external src  */
#define FSYNC_MODE_INT      0x01     /* Frame Sync dervived from internal src */
/* Transmit Receive Control Register XCR/RCR */
#define SINGLE_PHASE        0x00     /* Selects single phase frames           */
#define DUAL_PHASE          0x01     /* Selects dual phase frames             */
#define MAX_FRAME_LENGTH    0x7f     /* maximum number of words per frame     */
#define WORD_LENGTH_8       0x00     /* 8 bit word length (requires filling)  */
#define WORD_LENGTH_12      0x01     /* 12 bit word length       ""           */
#define WORD_LENGTH_16      0x02     /* 16 bit word length       ""           */
#define WORD_LENGTH_20      0x03     /* 20 bit word length       ""           */
#define WORD_LENGTH_24      0x04     /* 24 bit word length       ""           */
#define WORD_LENGTH_32      0x05     /* 32 bit word length (matches DRR DXR sz*/
#define MAX_WORD_LENGTH     WORD_LENGTH_32
#define NO_COMPAND_MSB_1ST  0x00     /* No Companding, Data XFER starts w/MSb */
#define NO_COMPAND_LSB_1ST  0x01     /* No Companding, Data XFER starts w/LSb */
#define COMPAND_ULAW        0x02     /* Compand ULAW, 8 bit word length only  */
#define COMPAND_ALAW        0x03     /* Compand ALAW, 8 bit word length only  */
#define FRAME_IGNORE        0x01     /* Ignore frame sync pulses after 1st    */
#define NO_FRAME_IGNORE     0x00     /* Utilize frame sync pulses             */
#define DATA_DELAY0         0x00     /* 1st bit in same clk period as fsync   */
#define DATA_DELAY1         0x01     /* 1st bit 1 clk period after fsync      */
#define DATA_DELAY2         0x02     /* 1st bit 2 clk periods after fsync     */
/* Sample Rate Generator Register SRGR */
#define MAX_SRG_CLK_DIV     0xFF     /* max value to divide Sample Rate Gen Cl*/
#define MAX_FRAME_WIDTH     0xFF     /* maximum FSG width in CLKG periods     */
#define MAX_FRAME_PERIOD    0x0FFF   /* FSG period in CLKG periods            */
#define FSX_DXR_TO_XSR      0x00     /* Transmit FSX due to DXR to XSR copy   */
#define FSX_FSG             0x01     /* Transmit FSX due to FSG               */
#define CLKS_POL_FALLING    0x00     /* falling edge generates CLKG and FSG   */
#define CLKS_POL_RISING     0x01     /* rising edge generates CLKG and FSG    */
#define GSYNC_OFF           0x00     /* CLKG always running                   */
#define GSYNC_ON            0x01     /* CLKG and FSG synched to FSR           */
/*****************************************************************/
/* MCBSP_BYTES_PER_WORD – return # of bytes required to hold #    */
/*                        of bits indicated by wdlen             */
/*****************************************************************/
#define MCBSP_BYTES_PER_WORD(wdlen) \
       ( (wdlen) == WORD_LENGTH_32 ? 4 : (int)(((wdlen) + 2) / 2) )
/*****************************************************************/
/* MCBSP_ENABLE(unsigned short port_no, unsigned short type) –    */
/*            starts serial port receive and/or transmit         */
/*            type= 1 rx, type= 2 tx, type= 3 both               */
/*****************************************************************/
#define MCBSP_ENABLE(port_no,type)\
     (*(unsigned int *)MCBSP_SPCR_ADDR(port_no) |= \
     ((type % 2) * MASK_BIT(RRST)) | ((type/2) * MASK_BIT(XRST)))
/*****************************************************************/
/* MCBSP_TX_RESET() – reset transmit side of serial port          */
/*                                                                */
/*****************************************************************/
#define MCBSP_TX_RESET(port_no)\
     (*(unsigned int *)MCBSP_SPCR_ADDR(port_no) &= ~MASK_BIT(XRST)
```

```
/*****************************************************************/
/* MCBSP_RX_RESET() – reset receive side of serial port          */
/*                                                               */
/*****************************************************************/
#define MCBSP_RX_RESET(port_no)\
    (*(unsigned int *)MCBSP_SPCR_ADDR(port_no) &= ~MASK_BIT(RRST))
/*****************************************************************/
/* MCBSP_READ() – read data value from serial port               */
/*****************************************************************/
#define MCBSP_READ(port_no)\
     (*(unsigned int *)(MCBSP_DRR_ADDR(port_no)))
/*****************************************************************/
/* MCBSP_WRITE() – write data value to serial port transmit reg  */
/*****************************************************************/
#define MCBSP_WRITE(port_no, data)\
     (*(unsigned int *)(MCBSP_DXR_ADDR(port_no)) = (unsigned int) data)
/*****************************************************************/
/* MCBSP_IO_ENABLE() – place port in general purpose I/O mode    */
/*****************************************************************/
#define MCBSP_IO_ENABLE(port_no) \
        { MCBSP_TX_RESET(port_no); MCBSP_RX_RESET(port_no); \
          RESET_FIELD(MCBSP_PCR_ADDR(port_no),RIOEN,2); }
/*****************************************************************/
/* MCBSP_IO_DISABLE() – take port out of general purpose I/O mode */
/*****************************************************************/
#define MCBSP_IO_DISABLE(port_no) \
        SET_FIELD(MCBSP_PCR_ADDR(port_no),RIOEN,2)
/*****************************************************************/
/* MCBSP_FRAME_SYNC_ENABLE – sets FRST bit in SPCR               */
/*****************************************************************/
#define MCBSP_FRAME_SYNC_ENABLE(port_no) \
        (SET_BIT(MCBSP_SPCR_ADDR(port_no),FRST))
/*****************************************************************/
/* MCBSP_FRAME_SYNC_RESET – clrs FRST bit in SPCR                */
/*****************************************************************/
#define MCBSP_FRAME_SYNC_RESET(port_no) \
        (RESET_BIT(MCBSP_SPCR_ADDR(port_no),FRST))
/*****************************************************************/
/* MCBSP_SAMPLE_RATE_ENABLE – sets GRST bit in SPCR              */
/*****************************************************************/
#define MCBSP_SAMPLE_RATE_ENABLE(port_no) \
        (SET_BIT(MCBSP_SPCR_ADDR(port_no),GRST))
/*****************************************************************/
/* MCBSP_SAMPLE_RATE_RESET – clrs GRST bit in SPCR               */
/*****************************************************************/
#define MCBSP_SAMPLE_RATE_RESET(port_no) \
        (RESET_BIT(MCBSP_SPCR_ADDR(port_no),GRST))
/*****************************************************************/
/* MCBSP_RRDY – returns selected ports RRDY                      */
/*****************************************************************/
#define MCBSP_RRDY(port_no) \
        (GET_BIT(MCBSP_SPCR_ADDR(port_no),RRDY))
/*****************************************************************/
```

```
/* MCBSP_XRDY – returns selected ports XRDY                          */
/*********************************************************************/
#define MCBSP_XRDY(port_no) \
        (GET_BIT(MCBSP_SPCR_ADDR(port_no),XRDY))
/*********************************************************************/
/* MCBSP_LOOPBACK_ENABLE – places selected port in loopback          */
/*********************************************************************/
#define MCBSP_LOOPBACK_ENABLE(port_no) \
        (SET_BIT(MCBSP_SPCR_ADDR(port_no),DLB))
/*********************************************************************/
/* MCBSP_LOOPBACK_DISABLE – takes port out of DLB                    */
/*********************************************************************/
#define MCBSP_LOOPBACK_DISABLE(port_no) \
        (RESET_BIT(MCBSP_SPCR_ADDR(port_no),DLB))
/*-------------------------------------------------------------------------*/
/* GLOBAL VARIABLES                                                        */
/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/
/* FUNCTIONS                                                               */
/*-------------------------------------------------------------------------*/
__INLINE void mcbsp_init(unsigned short port_no,
                         unsigned int spcr_ctrl,
                         unsigned int rcr_ctrl,
                         unsigned int xcr_ctrl,
                         unsigned int srgr_ctrl,
                         unsigned int mcr_ctrl,
                         unsigned int rcer_ctrl,
                         unsigned int xcer_ctrl,
                         unsigned int pcr_ctrl);
#ifdef _INLINE
/*********************************************************************/
/* mcbsp_init – initialize and start serial port operation           */
/*                                                                   */
/*********************************************************************/
static inline void mcbsp_init(unsigned short port_no,
                              unsigned int spcr_ctrl,
                              unsigned int rcr_ctrl,
                              unsigned int xcr_ctrl,
                              unsigned int srgr_ctrl,
                              unsigned int mcr_ctrl,
                              unsigned int rcer_ctrl,
                              unsigned int xcer_ctrl,
                              unsigned int pcr_ctrl)
```

```
{
   unsigned int *port = (unsigned int *)(MCBSP_ADDR(port_no));
   /*****************************************************************/
   /* Place port in reset – setting XRST & RRST to 0                */
   /*****************************************************************/
   *(port + 2)      &= ~(MASK_BIT(RRST) | MASK_BIT(XRST));
   /*****************************************************************/
   /* Set values of all control reigsters                           */
   /*****************************************************************/
   *(port + 3) = rcr_ctrl;
   *(port + 4) = xcr_ctrl;
   *(port + 5) = srgr_ctrl;
   *(port + 6) = mcr_ctrl;
   *(port + 7) = rcer_ctrl;
   *(port + 8) = xcer_ctrl;
   *(port + 9) = pcr_ctrl;
   *(port + 2) = ~(MASK_BIT(RRST) | MASK_BIT(XRST)) & (spcr_ctrl);
   *(port + 2) |= (MASK_BIT(RRST) | MASK_BIT(XRST)) & (spcr_ctrl);
}

#endif
#ifdef __INLINE
#undef __INLINE
#endif
#endif /* _MCBSP_H_ */
```

A.1.7 regs.h

```
/*****************************************************************************/
/*  REGS.H – TMS320C6x Peripheral Support Library CPU Register Support       */
/*                                                                           */
/*     This file provides the header for the DSP's register support.         */
/*                                                                           */
/*                                                                           */
/*  MACRO FUNCTIONS:                                                         */
/*     REG_READ          – Read register at specified address                */
/*     REG_WRITE         – Write to register at specified address            */
/*     RESET_BIT         – Clears bit in register.                           */
/*     GET_BIT           – Returns bit value in register.                    */
/*     SET_BIT           – Sets bit in register.                             */
/*     MASK_BIT          – Create (1's) mask for specified bit.              */
/*     ASSIGN_BIT_VAL    – Assign bit to specified value                     */
/*     RESET_FIELD       – Clears field in register                          */
/*     GET_FIELD         – Returns value of bit field in a register          */
/*     MASK_FIELD        – Create (1's) mask for specified field             */
/*     LOAD_FIELD        – Assigns bit field in register                     */
/*     GET_REG           – Returns value of non memory mapped register       */
/*     SET_REG           – Sets value of a non memory mapped register        */
/*     GET_REG_BIT       – Return bit value in non memory mapped register    */
/*     SET_REG_BIT       – Sets bit in non memory mapped register            */
/*     RESET_REG_BIT     – Resets given bit in non memory mapped register    */
/*     GET_REG_FIELD     – Return value of specified register field          */
/*     LOAD_REG_FIELD    – Set value of specified register field             */
/*                                                                           */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
#ifndef _REGS_H_
#define _REGS_H_
/*---------------------------------------------------------------------------*/
/* DEFINES                                                                   */
/*---------------------------------------------------------------------------*/
/*************************** CONTROL REGISTERS ******************************/
extern cregister volatile unsigned int AMR;     /* Address Mode Register     */
extern cregister volatile unsigned int CSR;     /* Control Status Register   */
extern cregister volatile unsigned int IFR;     /* Interrupt Flag Register   */
extern cregister volatile unsigned int ISR;     /* Interrupt Set Register    */
extern cregister volatile unsigned int ICR;     /* Interrupt Clear Register  */
extern cregister volatile unsigned int IER;     /* Interrupt Enable Register */
extern cregister volatile unsigned int ISTP;    /* Interrupt Service Tbl Ptr */
extern cregister volatile unsigned int IRP;     /* Interrupt Return Pointer  */
extern cregister volatile unsigned int NRP;     /* Non–maskable Int Return Ptr*/
extern cregister volatile unsigned int IN;      /* General Purpose Input Reg */
extern cregister volatile unsigned int OUT;     /* General Purpose Output Reg */
```

```
/* Control Register Bitfields */
/* AMR */
#define A4_MODE              0
#define A4_MODE_SZ           2
#define A5_MODE              2
#define A5_MODE_SZ           2
#define A6_MODE              4
#define A6_MODE_SZ           2
#define A7_MODE              6
#define A7_MODE_SZ           2
#define B4_MODE              8
#define B4_MODE_SZ           2
#define B5_MODE             10
#define B5_MODE_SZ           2
#define B6_MODE             12
#define B6_MODE_SZ           2
#define B7_MODE             14
#define B7_MODE_SZ           2
#define BK0                 16
#define BK0_SZ               5
#define BK1                 21
#define BK1_SZ               5
/* CSR */
#define GIE                  0
#define PGIE                 1
#define DCC                  2
#define DCC_SZ               3
#define PCC                  5
#define PCC_SZ               3
#define EN                   8
#define SAT                  9
#define PWRD                10
#define PWRD_SZ              6
#define REVISION_ID         16
#define REVISION_ID_SZ       8
#define CPU_ID              24
#define CPU_ID_SZ            8
/* Interrupt Enable Register (IER) */
#define NMIE                 1
#define IE4                  4
#define IE5                  5
#define IE6                  6
#define IE7                  7
#define IE8                  8
#define IE9                  9
#define IE10                10
#define IE11                11
#define IE12                12
#define IE13                13
#define IE14                14
#define IE15                15
```

```
/* Interrupt Flag Register (IFR) */
#define NMIF                  1
#define IF4                   4
#define IF5                   5
#define IF6                   6
#define IF7                   7
#define IF8                   8
#define IF9                   9
#define IF10                 10
#define IF11                 11
#define IF12                 12
#define IF13                 13
#define IF14                 14
#define IF15                 15
/* Interrupt Set register (ISR) */
#define IS4                   4
#define IS5                   5
#define IS6                   6
#define IS7                   7
#define IS8                   8
#define IS9                   9
#define IS10                 10
#define IS11                 11
#define IS12                 12
#define IS13                 13
#define IS14                 14
#define IS15                 15
/* Interrupt Clear Register (ICR) */
#define IC4                   4
#define IC5                   5
#define IC6                   6
#define IC7                   7
#define IC8                   8
#define IC9                   9
#define IC10                 10
#define IC11                 11
#define IC12                 12
#define IC13                 13
#define IC14                 14
#define IC15                 15
/* Interrupt Service Table Pointer (ISTP) */
#define ISTB                 10
#define ISTB_SZ              22
#define HPEINT                5
#define HPEINT_SZ             5
/*---------------------------------------------------------------------------*/
/* MACRO FUNCTIONS                                                           */
/*---------------------------------------------------------------------------*/
#define CONTENTS_OF(addr) \
        (*((volatile unsigned int *)(addr)))
#define LENGTH_TO_BITS(length) \
        (~(0xffffffff << (length)))
```

```
/* MACROS to SET, CLEAR and RETURN bits and bitfields in Memory Mapped        */
/* locations using the address of the specified register.                     */
#define REG_READ(addr) \
        (CONTENTS_OF(addr))
#define REG_WRITE(addr,val) \
        (CONTENTS_OF(addr) = (val))
#define MASK_BIT(bit) \
        (1 << (bit))
#define RESET_BIT(addr,bit) \
        (CONTENTS_OF(addr) &= (~MASK_BIT(bit)))
#define GET_BIT(addr,bit) \
        (CONTENTS_OF(addr) & (MASK_BIT(bit)) ? 1 : 0)
#define SET_BIT(addr,bit) \
        (CONTENTS_OF(addr) = (CONTENTS_OF(addr)) | (MASK_BIT(bit)))
#define ASSIGN_BIT_VAL(addr,bit,val) \
        ( (val) ? SET_BIT(addr,bit) : RESET_BIT(addr,bit) )
#define MASK_FIELD(bit,length) \
        (LENGTH_TO_BITS(length) << (bit))
#define RESET_FIELD(addr,bit,length) \
        ( CONTENTS_OF(addr) &= (~MASK_FIELD(bit,length)))
#define GET_FIELD(addr,bit,length) \
       ((CONTENTS_OF(addr) & MASK_FIELD(bit,length)) >> bit)
#define LOAD_FIELD(addr,val,bit,length) \
        (CONTENTS_OF(addr) = (CONTENTS_OF(addr) &
(~MASK_FIELD(bit,length))) | (val<<bit))
/* MACROS to SET, CLEAR and RETURN bits and bitfields in Memory Mapped        */
/* and Non-Memory Mapped using register names.                                */
#define GET_REG(reg) \
        (reg)
#define SET_REG(reg,val) \
        ((reg)= (val))
#define GET_REG_BIT(reg,bit) \
        ((reg) & MASK_BIT(bit) ? 1 : 0)
#define SET_REG_BIT(reg,bit) \
        ((reg) |= MASK_BIT(bit))
#define RESET_REG_BIT(reg,bit) \
        ((reg) &= (~MASK_BIT(bit)))
#define GET_REG_FIELD(reg,bit,length) \
        ((reg & MASK_FIELD(bit,length)) >> bit)
#define LOAD_REG_FIELD(reg,val,bit,length) \
        (reg &= (~MASK_FIELD(bit,length)) | (val<<bit))
#endif /* ifndef _REGS_H_ */
```

A.1.8 timer.h

```
/*****************************************************************************/
/*  TIMER.H – TMS320C6x Peripheral Support Library Timers Support            */
/*                                                                           */
/*     This file provides the header for the DSP’s timers support.           */
/*                                                                           */
/*  MACRO FUNCTIONS:                                                         */
/*    TIMER_CTRL_ADDR(chan)                                                  */
/*    TIMER_PERIOD_ADDR(chan)                                                */
/*    TIMER_COUNTER_ADDR(chan)                                               */
/*    TIMER_RESET(chan)                                                      */
/*    TIMER_INIT(chan,ctrl,per,cnt)                                          */
/*    TIMER_START(chan)                                                      */
/*    TIMER_STOP(chan)                                                       */
/*    TIMER_RESUME(chan)                                                     */
/*    TIMER_MODE_SELECT(chan,mode)                                           */
/*    TIMER_CLK_INTERNAL(chan)                                               */
/*    TIMER_CLK_EXTERNAL(chan)                                               */
/*    TOUT_ENABLE(chan)                                                      */
/*    TOUT_DISABLE(chan)                                                     */
/*    TOUT_VAL(chan,val)                                                     */
/*    TOUT_ASSERT(chan)                                                      */
/*    TOUT_NEGATE(chan)                                                      */
/*    TINP_GET(chan)                                                         */
/*    TIMER_READ(chan)                                                       */
/*    TIMER_GET_COUNT(chan)                                                  */
/*    TIMER_SET_COUNT(chan,val)                                              */
/*    TIMER_AVAILABLE(chan)                                                  */
/*    TIMER_SET_PERIOD(chan,val)                                             */
/*    TIMER_GET_PERIOD(chan)                                                 */
/*    TIMER_GET_TSTAT(chan)                                                  */
/*                                                                           */
/*  FUNCTIONS:                                                               */
/*    timer_delay() – delay specified number of timer periods                */
/*                                                                           */
/*****************************************************************************/
#ifndef _TIMER_H_
#define _TIMER_H_
/*---------------------------------------------------------------------------*/
/* INCLUDES                                                                  */
/*---------------------------------------------------------------------------*/
#include ”regs.h”
/*---------------------------------------------------------------------------*/
/* DEFINES AND MACROS                                                        */
/*---------------------------------------------------------------------------*/
/*****************************************************************************/
/************************** TIMER REGISTERS **********************************/
#define TIMER_BASE_ADDR 0x01940000
#define TIMER_CTRL_ADDR(chan) \
        (TIMER_BASE_ADDR + ((chan) * 0x40000))
#define TIMER_PERIOD_ADDR(chan) \
        (TIMER_BASE_ADDR + ((chan) * 0x40000) + 4)
```

```
#define TIMER_COUNTER_ADDR(chan) \
        (TIMER_BASE_ADDR + ((chan) *0x40000) + 8)
#define TIMER0_CTRL_ADDR      TIMER_CTRL_ADDR(0)
#define TIMER0_PERIOD_ADDR    TIMER_PERIOD_ADDR(0)
#define TIMER0_COUNTER_ADDR   TIMER_COUNTER_ADDR(0)
#define TIMER1_CTRL_ADDR      TIMER_CTRL_ADDR(1)
#define TIMER1_PERIOD_ADDR    TIMER_PERIOD_ADDR(1)
#define TIMER1_COUNTER_ADDR   TIMER_COUNTER_ADDR(1)
#define TIMER0_CTRL     *(volatile unsigned int *)(TIMER0_CTRL_ADDR)
#define TIMER0_PERIOD   *(volatile unsigned int *)(TIMER0_PERIOD_ADDR)
#define TIMER0_COUNTER  *(volatile unsigned int *)(TIMER0_COUNTER_ADDR)
#define TIMER1_CTRL     *(volatile unsigned int *)(TIMER1_CTRL_ADDR)
#define TIMER1_PERIOD   *(volatile unsigned int *)(TIMER1_PERIOD_ADDR)
#define TIMER1_COUNTER  *(volatile unsigned int *)(TIMER1_COUNTER_ADDR)
/* Timer Control Register Bitfield */
#define FUNC     0
#define INVOUT   1
#define DATOUT   2
#define DATIN    3
#define PWID     4
#define GO       6
#define HLD      7
#define C_P      8
#define CLKSRC   9
#define INVINP   10
#define TSTAT    11
#define TIMER_PULSE_MODE     0
#define TIMER_CLOCK_MODE     1
/*-------------------------------------------------------------------------*/
/* TIMER_RESET – reset timer to conditions defined by device reset         */
/*-------------------------------------------------------------------------*/
#define TIMER_RESET(chan) \
        { (*(unsigned int *)(TIMER_CTRL_ADDR(chan)) = 0);    \
          (*(unsigned int *)(TIMER_PERIOD_ADDR(chan)) = 0);  \
          (*(unsigned int *)(TIMER_COUNTER_ADDR(chan)) = 0); \
        }
/*-------------------------------------------------------------------------*/
/* TIMER_INIT – initialize timer registers                                 */
/*-------------------------------------------------------------------------*/
#define TIMER_INIT(chan,ctrl,per,cnt) \
        { (*unsigned int *) (TIMER_CTRL_ADDR(chan))    = ctrl); \
          (*(unsigned int *)(TIMER_PERIOD_ADDR(chan))  = per); \
          (*(unsigned int *)(TIMER_COUNTER_ADDR(chan)) = cnt); \
        }
/*-------------------------------------------------------------------------*/
/* TIMER_START – Sets both GO and HOLD bits in Timer Control Register which*/
/*               resets the Timer Counter Register and enables counting on */
/*               on the next clock.  (GO bit autoclears)                   */
/*-------------------------------------------------------------------------*/
#define TIMER_START(chan) \
        REG_WRITE(TIMER_CTRL_ADDR(chan), (REG_READ(TIMER_CTRL_ACCR(chan)) | 0xc0))
/*-------------------------------------------------------------------------*/
/* TIMER_STOP – Asserts (clears) the HOLD bit in the Timer Control Register*/
```

```
/*-------------------------------------------------------------------------*/
#define TIMER_STOP(chan) \
        RESET_BIT(TIMER_CTRL_ADDR(chan),HLD)
/*-------------------------------------------------------------------------*/
/* TIMER_RESUME – Negates (sets) the HOLD bit to resume counting without   */
/*                resetting the counter register                           */
/*-------------------------------------------------------------------------*/
#define TIMER_RESUME(chan) \
        SET_BIT(TIMER_CTRL_ADDR(chan),HLD)
/*-------------------------------------------------------------------------*/
/* TIMER_MODE_SELECT – selects between PULSE and CLOCK modes                */
/*-------------------------------------------------------------------------*/
#define TIMER_MODE_SELECT(chan,mode) \
        (mode == TIMER_CLOCK_MODE) ? SET_BIT(TIMER_CTRL_ADDR(chan),C–P) \
        : RESET_BIT(TIMER_CTRL_ADDR(chan),C–P)
/*-------------------------------------------------------------------------*/
/* TIMER_CLK_INTERNAL – sets CLKSRC to select CPU clock/4 as timer clock   */
/*-------------------------------------------------------------------------*/
#define TIMER_CLK_INTERNAL(chan) \
        SET_BIT(TIMER_CTRL_ADDR(chan),CLKSRC)
/*-------------------------------------------------------------------------*/
/* TIMER_CLK_EXTERNAL – clears CLKSRC to select TINP as timer clock        */
/*-------------------------------------------------------------------------*/
#define TIMER_CLK_EXTERNAL(chan) \
        RESET_BIT(TIMER_CTRL_ADDR(chan),CLKSRC)
/*-------------------------------------------------------------------------*/
/* TOUT_ENABLE – configures TOUT as general purpose output pin.            */
/*                                                                         */
/*-------------------------------------------------------------------------*/
#define TOUT_ENABLE(chan) \
        RESET_BIT(TIMER_CTRL_ADDR(chan),FUNC)
/*-------------------------------------------------------------------------*/
/* TOUT_DISABLE – configures TOUT as a timer pin; reflects value of TSTAT  */
/*                conditioned by INVOUT                                    */
/*-------------------------------------------------------------------------*/
#define TOUT_DISABLE(chan) \
        SET_BIT(TIMER_CTRL_ADDR(chan),FUNC)
/*-------------------------------------------------------------------------*/
/* TOUT_VAL – assigns val to TOUT pin when TOUT is enabled as general purpose*/
/*            output                                                       */
/*-------------------------------------------------------------------------*/
#define TOUT_VAL(chan,val) \
        ASSIGN_BIT_VAL(TIMER_CTRL_ADDR(chan),DATOUT,val)
/*-------------------------------------------------------------------------*/
/* TOUT_ASSERT – assigns 1 to TOUT pin when TOUT is enabled as general     */
/*               purpose output                                            */
/*-------------------------------------------------------------------------*/
#define TOUT_ASSERT(chan) \
        SET_BIT(TIMER_CTRL_ADDR(chan),DATOUT)
/*-------------------------------------------------------------------------*/
/* TOUT_NEGATE – assignes 0 to TOUT pin when TOUT is enables as general    */
/*               purpose output                                            */
/*-------------------------------------------------------------------------*/
```

```
#define TOUT_NEGATE(chan) \
        RESET_BIT(TIMER_CTRL_ADDR(chan),DATOUT)
/*-------------------------------------------------------------------------*/
/* TINP_GET – returns value on TINP input pin                              */
/*-------------------------------------------------------------------------*/
#define TINP_GET \
        GET_BIT(TIMER_CTRL_ADDR(chan),DATIN)
/*-------------------------------------------------------------------------*/
/* TIMER_READ – reads value of Timer Counter Register                      */
/*-------------------------------------------------------------------------*/
#define TIMER_READ(chan) \
        (REG_READ(TIMER_COUNTER_ADDR(chan)))
/*-------------------------------------------------------------------------*/
/* TIMER_GET_COUNT – reads value of Timer Counter Register                 */
/*-------------------------------------------------------------------------*/
#define TIMER_GET_COUNT(chan) \
        (REG_READ(TIMER_COUNTER_ADDR(chan)))
/*-------------------------------------------------------------------------*/
/* TIMER_SET_COUNT – reads value of Timer Counter Register                 */
/*-------------------------------------------------------------------------*/
#define TIMER_SET_COUNT(chan,val) \
        (REG_WRITE(TIMER_COUNTER_ADDR(chan),val))
/*-------------------------------------------------------------------------*/
/* TIMER_AVAILABLE – checks timer for availability; returns TRUE or FALSE  */
/*-------------------------------------------------------------------------*/
#define TIMER_AVAILABLE(chan) \
        (GET_BIT(TIMER_CTRL_ADDR(chan),HLD) ? 0 : 1)
/*-------------------------------------------------------------------------*/
/* TIMER_SET_PERIOD – sets value of Timer Period Register                  */
/*-------------------------------------------------------------------------*/
#define TIMER_SET_PERIOD(chan,val) \
        (REG_WRITE(TIMER_PERIOD_ADDR(chan),val))
/*-------------------------------------------------------------------------*/
/* TIMER_GET_PERIOD – returns value of Timer Period Register               */
/*-------------------------------------------------------------------------*/
#define TIMER_GET_PERIOD(chan) \
        (REG_READ(TIMER_PERIOD_ADDR(chan)))
/*-------------------------------------------------------------------------*/
/* TIMER_GET_TSTAT – returns value of TSTAT bit in Timer Ctrl Register     */
/*-------------------------------------------------------------------------*/
#define TIMER_GET_TSTAT(chan) \
        (GET_BIT(TIMER_CTRL_ADDR(chan),TSTAT))
/*-------------------------------------------------------------------------*/
/* GLOBAL VARIABLES                                                        */
/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/
/* FUNCTIONS                                                               */
/*-------------------------------------------------------------------------*/
int timer_delay(short num_timer_periods);
#endif /* ifndef _TIMER_H_ */
```

A.2 C and Assembly Files

The following sections contain the C and assembly source files that are included in the ’C6x peripheral support library.

A.2.1 dma.c

```
/*************************************************************************/
/*  DMA.C – TMS320C6x Peripheral Support Library DMA Support              */
/*                                                                        */
/*     This file provides the support for the TMS320C6x DSP’s DMA         */
/*     controller.                                                        */
/*                                                                        */
/*                                                                        */
/*  FUNCTIONS:                                                            */
/*     dma_init()        – Initialize channel specific control registers  */
/*     dma_global_init() – Initialize global control registers            */
/*     dma_reset()       – Reset all four DMA channels.                   */
/*                                                                        */
/*                                                                        */
/*  STATIC FUNCTIONS:                                                     */
/*     None.                                                              */
/*                                                                        */
/*  GLOBAL VARIABLES DEFINED                                              */
/*     None.                                                              */
/*                                                                        */
/*                                                                        */
/*************************************************************************/
/*-------------------------------------------------------------------------*/
/* INCLUDES                                                                */
/*-------------------------------------------------------------------------*/
#include ”dma.h”
/*-------------------------------------------------------------------------*/
/* LOCAL DEFINES                                                           */
/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/
/* FILE LOCAL (STATIC) VARIABLES                                           */
/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/
/* FILE LOCAL (STATIC) PROTOTYPES                                          */
/*-------------------------------------------------------------------------*/
/*-------------------------------------------------------------------------*/
/* FUNCTIONS                                                               */
/*-------------------------------------------------------------------------*/
/*************************************************************************/
/* DMA_INIT –  Initialize channel specific control registers.             */
/*                                                                        */
/*     This function is responsible for setting the primary control register, */
/*     secondary control register, source address, destination address and   */
/*     transfer count for the specified DMA channel.                      */
/*                                                                        */
/*************************************************************************/
```

```
void    dma_init(
             unsigned short channel    /*IN: DMA channel number              */
            ,unsigned int pri_ctrl     /*IN: Value to set primary cntl reg   */
            ,unsigned int sec_ctrl     /*IN: Value to set sec cntl reg       */
            ,unsigned int src_addr     /*IN: Value to set source addr reg    */
            ,unsigned int dst_addr     /*IN: Value to set dest addr reg      */
            ,unsigned int trans_ctr    /*IN: Value to set transfer counter   */
                )
{
  *((unsigned int *)(DMA_PRIMARY_CTRL_ADDR(channel)))   = pri_ctrl;
  *((unsigned int *)(DMA_SECONDARY_CTRL_ADDR(channel))) = sec_ctrl;
  *((unsigned int *)(DMA_SRC_ADDR_ADDR(channel)))       = src_addr;
  *((unsigned int *)(DMA_DEST_ADDR_ADDR(channel)))      = dst_addr;
  *((unsigned int *)(DMA_XFR_COUNTER_ADDR(channel)))    = trans_ctr;
  return;
}
/*****************************************************************************/
/* DMA_GLOBAL_INIT -  Initialize global control registers.                   */
/*                                                                           */
/*     This function is responsible for setting the DMA global control       */
/*     register, global count reload registers A & B, global index registers */
/*     A & B, and global address registers A-D.                              */
/*                                                                           */
/*****************************************************************************/
void     dma_global_init(
              unsigned int gcr          /*IN:Value for Global Control Register*/
             ,unsigned int gcra         /*IN:Value for Global Cnt Reload Reg A*/
             ,unsigned int gcrb         /*IN:Value for Global Cnt Reload Reg B*/
             ,unsigned int gndxa        /*IN:Value for Global Idx Reg A       */
             ,unsigned int gndxb        /*IN:Value for Global Idx Reg B       */
             ,unsigned int gaddra       /*IN:Value for Global Addr Reg A      */
             ,unsigned int gaddrb       /*IN:Value for Global Addr Reg B      */
             ,unsigned int gaddrc       /*IN:Value for Global Addr Reg C      */
             ,unsigned int gaddrd)      /*IN:Value for Global Addr Reg D      */
{
  DMA_GCR_A     = gcra;
  DMA_GCR_B     = gcrb;
  DMA_GNDX_A    = gndxa;
  DMA_GNDX_B    = gndxb;
  DMA_GADDR_A   = gaddra;
  DMA_GADDR_B   = gaddrb;
  DMA_GADDR_C   = gaddrc;
  DMA_GADDR_D   = gaddrd;
  return;
}
/*****************************************************************************/
/* DMA_RESET -  Reset all four DMA channel.                                  */
/*                                                                           */
/*     This function resets all four DMA channels by initializing            */
/*     channel control registers to their default values                     */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
```

```
void     dma_reset(void)
{
  int chan;
  for (chan= 0;chan < DMA_CH3; chan++)
  {
    *(unsigned int *)(DMA_PRIMARY_CTRL_ADDR(chan))   = 0x00;
    *(unsigned int *)(DMA_SECONDARY_CTRL_ADDR(chan)) = 0x00;
    *(unsigned int *)(DMA_SRC_ADDR_ADDR(chan))       = 0x00;
    *(unsigned int *)(DMA_DEST_ADDR_ADDR(chan))      = 0x00;
    *(unsigned int *)(DMA_XFR_COUNTER_ADDR(chan))    = 0x00;
  }
  DMA_GCR_A     = 0x00;
  DMA_GCR_B     = 0x00;
  DMA_GNDX_A    = 0x00;
  DMA_GNDX_B    = 0x00;
  DMA_GADDR_A   = 0x00;
  DMA_GADDR_B   = 0x00;
  DMA_GADDR_C   = 0x00;
  DMA_GADDR_D   = 0x00;
  DMA_GCTRL     = 0x00;
}
```

A.2.2 emif.c

```
/*************************************************************************/
/*  EMIF.C – TMS320C6x Peripheral Support Library EMIF Support           */
/*                                                                       */
/*     This file provides support for the TMS320C6x DSP's external memory*/
/*     interface.                                                        */
/*                                                                       */
/*                                                                       */
/*  FUNCTIONS:                                                           */
/*    emif_init() – Initialize EMIF registers                           */
/*                                                                       */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
/*-----------------------------------------------------------------------*/
/* INCLUDES                                                              */
/*-----------------------------------------------------------------------*/
#include "emif.h"
/*-----------------------------------------------------------------------*/
/* LOCAL DEFINES                                                         */
/*-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------*/
/* FILE LOCAL (STATIC) VARIABLES                                         */
/*-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------*/
/* FILE LOCAL (STATIC) PROTOTYPES                                        */
/*-----------------------------------------------------------------------*/
/*-----------------------------------------------------------------------*/
/* FUNCTIONS                                                             */
/*-----------------------------------------------------------------------*/
/*************************************************************************/
/* EMIF_INIT –  Initialize EMIF registers                                */
/*                                                                       */
/*************************************************************************/
void emif_init(unsigned int g_ctrl,
               unsigned int ce0_ctrl,
               unsigned int ce1_ctrl,
               unsigned int ce2_ctrl,
               unsigned int ce3_ctrl,
               unsigned int sdram_ctrl,
               unsigned int sdram_refresh
              )
{
  REG_WRITE(EMIF_GCTRL_ADDR, g_ctrl);
  REG_WRITE(EMIF_CE0_CTRL_ADDR, ce0_ctrl);
  REG_WRITE(EMIF_CE1_CTRL_ADDR, ce1_ctrl);
  REG_WRITE(EMIF_CE2_CTRL_ADDR, ce2_ctrl);
  REG_WRITE(EMIF_CE3_CTRL_ADDR, ce3_ctrl);
  REG_WRITE(EMIF_SDRAM_CTRL_ADDR, sdram_ctrl);
  REG_WRITE(EMIF_SDRAM_REF_ADDR, sdram_refresh);
}
```

A.2.3 intr.c

```
/*****************************************************************************/
/*  INTR.C – TMS320C6x Peripheral Support Library Interrupt Support          */
/*                                                                           */
/*     This module provides routines to control and initialize               */
/*     interrupt processing facilities on the C6x.                           */
/*                                                                           */
/*  FUNCTIONS:                                                               */
/*                                                                           */
/*    intr_reset()         – Reset interrupt registers to default            */
/*    intr_init()          – Interrupt initialization                        */
/*    intr_isn()           – Assign ISN to CPU interrupt                     */
/*    intr_get_cpu_intr()  – Return CPU interrupt assigned to ISN            */
/*    intr_map()           – Place ISN in interrupt multiplexor              */
/*  STATIC FUNCTIONS:                                                        */
/*    None.                                                                  */
/*                                                                           */
/*  GLOBAL VARIABLES DEFINED:                                                */
/*                                                                           */
/*    isr_jump_table       – Interrupt jump table                            */
/*                                                                           */
/*****************************************************************************/
/*---------------------------------------------------------------------------*/
/* INCLUDES AND LOCAL DEFINES                                                */
/*---------------------------------------------------------------------------*/
#include <stdio.h>
#include ”intr.h”
/*---------------------------------------------------------------------------*/
/* GLOBAL VARIABLES                                                          */
/*---------------------------------------------------------------------------*/
/* Interrupt Service Routine Jump Table                                      */
near unsigned int isr_jump_table[16] =
     { (unsigned int)c_int00,
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
      };
/*---------------------------------------------------------------------------*/
/* FILE LOCAL (STATIC) VARIABLES                                             */
/*---------------------------------------------------------------------------*/
/*---------------------------------------------------------------------------*/
/* FILE LOCAL (STATIC) PROTOTYPES                                            */
/*---------------------------------------------------------------------------*/
interrupt void unhooked_isr(void);
/*---------------------------------------------------------------------------*/
/* FUNCTIONS                                                                 */
/*---------------------------------------------------------------------------*/
/* INTR_RESET – reset interrupt registers to default values                  */
/*---------------------------------------------------------------------------*/
void intr_reset(void)
{
  unsigned int val;
  /* disable global interrupts                                              */
  LOAD_REG_FIELD(CSR,0,0,2);
```

```
  /* disable interrupts                                                    */
  SET_REG(IER,1);
  SET_REG(ICR,OxFFF0);
  /* default external interrupts to rising-edge triggered                  */
  REG_WRITE(EXTERNAL_INTR_POL_ADDR,0);
  /* reset interrupt multiplexers                                          */
  INTR_MAP_RESET();
  /* initialize interrupt service table pointer                            */
  val= (unsigned int)(&istb);
  SET_REG(ISTP,val);
}
/*-------------------------------------------------------------------------*/
/* INTR_INIT - initialize Interrupt Service Table Pointer                  */
/*                                                                         */
/*   This function initializes the ISTP based upon the global vec_table which */
/*   is resolved at link time. Refer to linker command file for this value    */
/*-------------------------------------------------------------------------*/
void intr_init(void)
{
  unsigned int val;
  val= (unsigned int)(&istb);
  SET_REG(ISTP,val);
}
/*-------------------------------------------------------------------------*/
/* INTR_HOOK - hooks an ISR to interrupt.                                  */
/*                                                                         */
/*-------------------------------------------------------------------------*/
void intr_hook(void (*fp)(void),int intr_num)
{
  if (intr_num < 16)
    *((unsigned int *)isr_jump_table + intr_num)= (unsigned int)fp;
}
/*-------------------------------------------------------------------------*/
/* unhooked_isr - generic ISR for testing                                  */
/*                                                                         */
/*   This function serves as the interrupt service routine for unhooked       */
/*   ISR locations within the IST                                          */
/*-------------------------------------------------------------------------*/
interrupt void unhooked_isr(void)
{
  printf("unhooked_isr() entered");
}
/*-------------------------------------------------------------------------*/
/* INTR_MAP() - Map interrupt source (isn) to cpu interrupt (cpu_intr).    */
/*                                                                         */
/*   This function loads the isn value into the INTSEL field of the           */
/*   appropriate Interrupt Multiplexer register indicated by cpu_intr      */
/*-------------------------------------------------------------------------*/
void intr_map(int cpu_intr,int isn)
{
  int intsel;
  int intr_field;
  int sel;
```

```
  sel= 0;
  intr_field= cpu_intr;
  if (cpu_intr > CPU_INT9)
  {
    sel=1;
    intr_field -= 6;
  }
  intsel = (intr_field - 4) * 5;
  if (intsel > 10)
    intsel++;
  INTR_SET_MAP(intsel,isn,sel);
}
/*-------------------------------------------------------------------------*/
/* INTR_GET_ISN() - return isn currently mapped to cpu interrupt           */
/*-------------------------------------------------------------------------*/
int intr_isn(int cpu_intr)
{
  int intsel;
  int intr_field;
  int sel;
  int isn;
  sel= 0;
  intr_field= cpu_intr;
  if (cpu_intr > CPU_INT9)
  {
    sel=1;
    intr_field -= 6;
  }
  intsel = (intr_field - 4) * 5;
  if (intsel > 10)
    intsel++;
  isn= INTR_GET_ISN(intsel,sel);
  return(isn);
}
/*-------------------------------------------------------------------------*/
/* intr_get_cpu_intr() - return cpu interrupt corresponding to isn in      */
/*                       interrupt selecter register.  If the isn is not   */
/*                       mapped, return -1                                 */
/*-------------------------------------------------------------------------*/
int intr_get_cpu_intr(int isn)
{
  int i;
  for (i= CPU_INT4;i<=CPU_INT15;i++)
  {
    if (intr_isn(i) == isn)
      return(i);
  }
  return(-1);
}
```

A.2.4 intr_.asm

```
;   intr_handler.asm
;
;     This file provides run time installable ISR capability through the use of
;     the intr_jump_table which is defined in intr.c.  This file provides the
;     ISFPs (Interrupt Service Fetch Packets) for the IST (Interrupt Service
;     Table).  If the address in the jump table index corresponding to the in
;     service CPU interrupt is 0, no branch is exucuted and control is returned
;     to the previous thread.
;
;

                    .ref          _c_int00     ;reset ISR
                    .ref          _isr_jump_table
                    .global       _istb        ; interrupt service table base
                    .text                      ;compile in code section
                    .sect         ".vec"
_istb:
                     mvk          _c_int00,b0
                     mvkh         _c_int00,b0
                     b            .s2 b0
                     nop          5
                     nop
                     nop
                     nop
                     nop
                    .asg          1, vec
                    .loop         15
                     stw     .d2   b0, *--b15
                     ldw     .d2   *+b14(_isr_jump_table + vec * 4), b0
                     nop           4
                [b0] b       .s2   b0
                     .if ( vec == 1 )
                [!b0] b      .s2   nrp          ;NMI ISR
             ||       ldw     .d2   *b15++, b0

                     .else
                [!b0] b      .s2   irp          ;Non-NMI ISR
             ||       ldw     .d2   *++b15, b0
                     .endif
                     nop           4
                     nop           1
                     .eval         vec + 1, vec
                     .endloop
```

A.2.5 mcbsp.c

```
/*****************************************************************************/
/*   MCBSP.C – TMS320C6x Peripheral Support Library McBSP Support            */
/*                                                                           */
/*    This file provides support for the TMS320C6x DSP's McBSPs.             */
/*                                                                           */
/*  FUNCTIONS:                                                               */
/*    mcbsp_init() – initializize McBSP registers                            */
/*                                                                           */
/*                                                                           */
/*  STATIC FUNCTIONS:                                                        */
/*    none.                                                                  */
/*                                                                           */
/*  GLOBAL VARIABLES DEFINED                                                 */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
/*---------------------------------------------------------------------------*/
/* INCLUDES AND LOCAL DEFINES                                                */
/*---------------------------------------------------------------------------*/
#include ”mcbsp.h”
/*---------------------------------------------------------------------------*/
/* GLOBAL VARIABLES                                                          */
/*---------------------------------------------------------------------------*/
/*---------------------------------------------------------------------------*/
/* FILE LOCAL (STATIC) VARIABLES                                             */
/*---------------------------------------------------------------------------*/
/*---------------------------------------------------------------------------*/
/* FILE LOCAL (STATIC) PROTOTYPES                                            */
/*---------------------------------------------------------------------------*/
/*---------------------------------------------------------------------------*/
/* FUNCTIONS                                                                 */
/*---------------------------------------------------------------------------*/
void mcsp_init(unsigned short port_no,
               unsigned int spcr_ctrl,
               unsigned int rcr_ctrl,
               unsigned int xcr_ctrl,
               unsigned int srgr_ctrl,
               unsigned int mcr_ctrl,
               unsigned int rcer_ctrl,
               unsigned int xcer_ctrl,
               unsigned int pcr_ctrl)
```

```
{
  unsigned int *port = (unsigned int *)MCBSP_ADDR(port_no);
  /*************************************************************/
  /* Place port in reset – setting XRST & RRST to 0            */
  /*************************************************************/
  *(port + 2)       &= ~(MASK_BIT(RRST) | MASK_BIT(XRST));
  /*************************************************************/
  /* Set values of all control reigsters                       */
  /*************************************************************/
  *(port + 3) = rcr_ctrl;
  *(port + 4) = xcr_ctrl;
  *(port + 5) = srgr_ctrl;
  *(port + 6) = mcr_ctrl;
  *(port + 7) = rcer_ctrl;
  *(port + 8) = xcer_ctrl;
  *(port + 9) = pcr_ctrl;
  *(port + 2) = ~(MASK_BIT(RRST) | MASK_BIT(XRST)) & (spcr_ctrl);
  *(port + 2) |= (MASK_BIT(RRST) | MASK_BIT(XRST)) & (spcr_ctrl);
}
```

A.2.6 timer.c

```
/*****************************************************************************/
/*  TIMER.C – TMS320C6x Peripheral Support Library Timer Support             */
/*                                                                           */
/*     This file provides support for the TMS320C6x DSP's timers.            */
/*                                                                           */
/*                                                                           */
/*  FUNCTIONS:                                                               */
/*    timer_delay() – delay specified number of timer periods                */
/*                                                                           */
/*  STATIC FUNCTIONS:                                                        */
/*    None.                                                                  */
/*                                                                           */
/*  GLOBAL VARIABLES DEFINED                                                 */
/*    None.                                                                  */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/
/*---------------------------------------------------------------------------*/
/* INCLUDES AND LOCAL DEFINES                                                */
/*---------------------------------------------------------------------------*/
#include "timer.h"
/*---------------------------------------------------------------------------*/
/* GLOBAL VARIABLES                                                          */
/*---------------------------------------------------------------------------*/
/*---------------------------------------------------------------------------*/
/* FILE LOCAL (STATIC) VARIABLES                                             */
/*---------------------------------------------------------------------------*/
/*---------------------------------------------------------------------------*/
/* FILE LOCAL (STATIC) PROTOTYPES                                            */
/*---------------------------------------------------------------------------*/
/*---------------------------------------------------------------------------*/
/* FUNCTIONS                                                                 */
/*---------------------------------------------------------------------------*/
/*****************************************************************************/
/* timer_delay – delay for specified number of timer periods                 */
/*                                                                           */
/*****************************************************************************/
int timer_delay(                          /* RET: OK (0) or ERROR (–1)       */
                short num_timer_periods)
{
  unsigned int period_reg;
  unsigned int ctrl_reg= 0;
  unsigned int chan    = 0;
  if (!TIMER_AVAILABLE(chan))
  {
    chan++;
    if (!TIMER_AVAILABLE(chan))
      return(–1);
  }
```

```
  /* define timer's period and control register values                    */
  period_reg= num_timer_periods;
  ctrl_reg= MASK_BIT(C_P) | MASK_BIT(CLKSRC);
  TIMER_INIT(chan,ctrl_reg,period_reg,0);
  TIMER_START(chan);
  /* poll for high to low transition */
  while (!(TIMER_GET_TSTAT(chan))){};
  while   (TIMER_GET_TSTAT(chan)) {};
  /* stop timer and return 0 to indicate success                          */
  TIMER_STOP(chan);
  return(0);
}
```

A.2.7 Makefile for Peripheral Support Library

```
makefile/
# Makefile for Peripheral Support Library

C6x_DIR     = C:\C6xTOOLS

C6x_BIN_DIR = $(C6x_DIR)\bin
C6x_LIB_DIR = $(C6x_DIR)\lib
C6x_INCLUDE_DIR = $(C6X_DIR)\include
H_DIR   = . . \h

# utilities
AR = $(C6x_BIN_DIR)\ar6x     # archiver
AS = $(C6x_BIN_DIR)\cl6x     # assembler
CC = $(C6x_BIN_DIR)\cl6x     # compiler
HX = $(C6x_BIN_DIR)\hex6x    # hex conversion utility
LK = $(C6x_BIN_DIR)\lnk6x    # linker

# includes and defines
DEFINEs  =
INCLUDES = –i$(H_DIR) –i$(C6X_INCLUDE_DIR)

# options
CC_OPTIONS = –c –q –ss –o0 –op0 –g
AS_OPTIONS = –c –q –al –as –gs
AR_OPTIONS = –rvq

# targets
ASMSRC = intr_.asm
CSRC = dma.c emif.c intr.c mcbsp.c timer.c
HSRC = $(H_DIR)\cache.h $(H_DIR)\dma.h $(H_DIR)\emif.h $(H_DIR)\hpi.h \
       $(H_DIR)\intr.h $(H_DIR)\mcbsp.h $(H_DIR)\regs.h $(H_DIR)\timer.h
ASMOBJ = $(ASMSRC:.asm=.obj)
COBJ = $(CSRC:.c=.obj)
OBJS = $(ASMOBJ) $(COBJ)
dev6x.lib : $(OBJS) makefile makefile.big
        $(AR) $(AR_OPTIONS) dev6x.lib $(ASMOBJ) $(COBJ)
dev6x.src : $(ASMSRC) $(CSRC) $(HSRC) makefile makefile.big
        $ (AR) $ (AR_OPTIONS) dev6x.src $(ASMSRC) $(HSRD) $(CSRC) makefile
        makefile.big
all : dev6x.lib dev6x.src
allclean : clean all
clean :
      del *.obj
      del *.lst
```

```
# object dependencies
dma.obj : dma.c
      $(CC) $(CC_OPTIONS) $(INCLUDES) $(DEFINES) dma.c
mcbsp.obj : mcbsp.c
      $(CC) $(CC_OPTIONS) $(INCLUDES) $(DEFINES) mcbsp.c
intr.obj : intr.c
      $(CC) $(CC_OPTIONS) $(INCLUDES) $(DEFINES) intr.c
timer.obj : timer.c
      $(CC) $(CC_OPTIONS) $(INCLUDES) $(DEFINES) timer.c
emif.obj : emif.c
      $(CC) $(CC_OPTIONS) $(INCLUDES) $(DEFINES) emif.c
intr_.obj : intr_.asm
    $(AS) $(AS_OPTIONS) $(INCLUDES) $(DEFINES) intr_.asm

# source dependencies
dma.c : $(H_DIR)\dma.h
emif.c : $(H_DIR)\emif.h
intr.c : $(H_DIR)\intr.h
mcbsp.c : $(H_DIR)\mcbsp.h
timer.c : $(H_DIR)\timer.h

# include dependencies
$(H_DIR)\dma.h : $(H_DIR)\regs.h
$(H_DIR)\mcbsp.h : $(H_DIR)\regs.h
$(H_DIR)\intr.h : $(H_DIR)\regs.h
$(H_DIR)\timer.h : $(H_DIR)\regs.h
$(H_DIR)\emif.h : $(H_DIR)\regs.h
```

A.2.8 Makefile for Peripheral Support Library (large memory model)

```
makefile/
# Makefile for Peripheral Support Library

C6x_DIR     = C:\C6xTOOLS

C6x_BIN_DIR = $(C6x_DIR)\bin
C6x_LIB_DIR = $(C6x_DIR)\lib
C6x_INCLUDE_DIR = $(C6X_DIR)\include
H_DIR   = . . \h

# utilities
AR = $(C6x_BIN_DIR)\ar6x     # archiver
AS = $(C6x_BIN_DIR)\cl6x     # assembler
CC = $(C6x_BIN_DIR)\cl6x     # compiler
HX = $(C6x_BIN_DIR)\hex6x    # hex conversion utility
LK = $(C6x_BIN_DIR)\lnk6x    # linker

# includes and defines
DEFINEs  =
INCLUDES = –i$(H_DIR) –i$(C6X_INCLUDE_DIR)

# options
CC_OPTIONS = –c –q –ss –o0 –op0 –g –me
AS_OPTIONS = –c –q –al –as –gs –me
AR_OPTIONS = –rvq

# targets
ASMSRC = intr_.asm
CSRC = dma.c emif.c intr.c mcbsp.c timer.c
HSRC = $(H_DIR)\cache.h $(H_DIR)\dma.h $(H_DIR)\emif.h $(H_DIR)\hpi.h \
       $(H_DIR)\intr.h $(H_DIR)\mcbsp.h $(H_DIR)\regs.h $(H_DIR)\timer.h
ASMOBJ = $(ASMSRC:.asm=.obj)
COBJ = $(CSRC:.c=.obj)
OBJS = $(ASMOBJ) $(COBJ)
dev6xe.lib : $(OBJS) makefile makefile.big
        $(AR) $(AR_OPTIONS) dev6xe.lib $(ASMOBJ) $(COBJ)
dev6x.src : $(ASMSRC) $(CSRC) $(HSRC) makefile makefile.big
        $ (AR) $ (AR_OPTIONS) dev6x.src $(ASMSRC) $(HSRD) $(CSRC) makefile
        makefile.big
all : dev6xe.lib dev6x.src
allclean : clean all
clean :
      del *.obj
      del *.lst
```

```
# object dependencies
dma.obj : dma.c
      $(CC) $(CC_OPTIONS) $(INCLUDES) $(DEFINES) dma.c
mcbsp.obj : mcbsp.c
      $(CC) $(CC_OPTIONS) $(INCLUDES) $(DEFINES) mcbsp.c
intr.obj : intr.c
      $(CC) $(CC_OPTIONS) $(INCLUDES) $(DEFINES) intr.c
timer.obj : timer.c
      $(CC) $(CC_OPTIONS) $(INCLUDES) $(DEFINES) timer.c
emif.obj : emif.c
      $(CC) $(CC_OPTIONS) $(INCLUDES) $(DEFINES) emif.c
intr_.obj : intr_.asm
    $(AS) $(AS_OPTIONS) $(INCLUDES) $(DEFINES) intr_.asm

# source dependencies
dma.c : $(H_DIR)\dma.h
emif.c : $(H_DIR)\emif.h
intr.c : $(H_DIR)\intr.h
mcbsp.c : $(H_DIR)\mcbsp.h
timer.c : $(H_DIR)\timer.h

# include dependencies
$(H_DIR)\dma.h : $(H_DIR)\regs.h
$(H_DIR)\mcbsp.h : $(H_DIR)\regs.h
$(H_DIR)\intr.h : $(H_DIR)\regs.h
$(H_DIR)\timer.h : $(H_DIR)\regs.h
$(H_DIR)\emif.h : $(H_DIR)\regs.h
```

Appendix B

Glossary

A

address: The location of program code or data stored; an individually accessible memory location.

A-law companding: See *compress and expand (compand)*.

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

assert: To make a digital logic device pin active. If the pin is active low, then a low voltage on the pin asserts it. If the pin is active high, then a high voltage asserts it.

B

bit: A binary digit, either a 0 or 1.

big endian: An addressing protocol in which bytes are numbered from left to right within a word. More significant bytes in a word have lower numbered addresses. Endian ordering is specific to hardware and is determined at reset. See also *little endian*.

block: The three least significant bits of the program address. These correspond to the address within a fetch packet of the first instruction being addressed.

boot: The process of loading a program into program memory.

boot mode: The method of loading a program into program memory. The ’C6x DSP supports booting from external ROM or the host port interface (HPI).

byte: A sequence of eight adjacent bits operated upon as a unit.

C

cache: A fast storage buffer in the central processing unit of a computer.

cache controller: System component that coordinates program accesses between CPU program fetch mechanism, cache, and external memory.

central processing unit (CPU): The portion of the processor involved in arithmetic, shifting, and Boolean logic operations, as well as the generation of data- and program-memory addresses. The CPU includes the central arithmetic logic unit (CALU), the multiplier, and the auxiliary register arithmetic unit (ARAU).

clock cycle: A periodic or sequence of events based on the input from the external clock.

clock modes: Options used by the clock generator to change the internal CPU clock frequency to a fraction or multiple of the frequency of the input clock signal.

code: A set of instructions written to perform a task; a computer program or part of a program.

coder-decoder or compression/decompression (codec): A device that codes in one direction of transmission and decodes in another direction of transmission.

compiler: A computer program that translates programs in a high-level language into their assembly-language equivalents.

compress and expand (compand): A quantization scheme for audio signals in which the input signal is compressed and then, after processing, is reconstructed at the output by expansion. There are two distinct companding schemes: A-law (used in Europe) and μ-law (used in the United States).

control register: A register that contains bit fields that define the way a device operates.

control register file: A set of control registers.

D

device ID: Configuration register that identifies each peripheral component interconnect (PCI).

digital signal processor (DSP): A semiconductor that turns analog signals—such as sound or light—into digital signals, which are discrete or discontinuous electrical impulses, so that they can be manipulated.

direct memory access (DMA): A mechanism whereby a device other than the host processor contends for and receives mastery of the memory bus so that data transfers can take place independent of the host.

DMA operation: A series of DMA data transfers.

DMA source: The module where the DMA data originates. DMA data is read from the DMA source.

DMA transfer: The process of transferring data from one part of memory to another. Each DMA transfer consists of a read bus cycle (source to DMA holding register) and a write bus cycle (DMA holding register to destination).

E

evaluation module (EVM): A board and software tools that allow the user to evaluate a specific device.

external interrupt: A hardware interrupt triggered by a specific value on a pin.

external memory interface (EMIF): Microprocessor hardware that is used to read to and write from off-chip memory.

F

fetch packet: A contiguous 8-word series of instructions fetched by the CPU and aligned on an 8-word boundary.

flag: A binary status indicator whose state indicates whether a particular condition has occurred or is in effect.

frame: An 8-word space in the cache RAMs. Each fetch packet in the cache resides in only one frame. A cache update loads a frame with the requested fetch packet. The cache contains 512 frames.

G

global interrupt enable bit (GIE): A bit in the control status register (CSR) that is used to enable or disable maskable interrupts.

H

host: A device to which other devices (peripherals) are connected and that generally controls those devices.

host port interface (HPI): A parallel interface that the CPU uses to communicate with a host processor.

I

index: A relative offset in the program address that specifies which of the 512 frames in the cache into which the current access is mapped.

indirect addressing: An addressing mode in which an address points to another pointer rather than to the actual data; this mode is prohibited in RISC architecture.

instruction fetch packet: A group of up to eight instructions held in memory for execution by the CPU.

internal interrupt: A hardware interrupt caused by an on-chip peripheral.

interrupt: A signal sent by hardware or software to a processor requesting attention. An interrupt tells the processor to suspend its current operation, save the current task status, and perform a particular set of instructions. Interrupts communicate with the operating system and prioritize tasks to be performed.

interrupt service fetch packet (ISFP): A fetch packet used to service interrupts. If eight instructions are insufficient, the user must branch out of this block for additional interrupt service. If the delay slots of the branch do not reside within the ISFP, execution continues from execute packets in the next fetch packet (the next ISFP).

interrupt service routine (ISR): A module of code that is executed in response to a hardware or software interrupt.

Internal peripherals: Devices connected to and controlled by a host device. The ’C6x internal peripherals include the direct memory access (DMA) controller, multichannel buffered serial ports (McBSPs), host port interface (HPI), external memory-interface (EMIF), and runtime support timers.

L

least significant bit (LSB): The lowest-order bit in a word.

linker: A software tool that combines object files to form an object module, which can be loaded into memory and executed.

little endian: An addressing protocol in which bytes are numbered from right to left within a word. More significant bytes in a word have higher-numbered addresses. Endian ordering is specific to hardware and is determined at reset. See also *big endian*.

M

μ-law companding: See *compress and expand (compand)*.

maskable interrupt: A hardware interrupt that can be enabled or disabled through software.

memory map: A graphical representation of a computer system's memory, showing the locations of program space, data space, reserved space, and other memory-resident elements.

memory-mapped register: An on-chip register mapped to an address in memory. Some memory-mapped registers are mapped to data memory, and some are mapped to input/output memory.

most significant bit (MSB): The highest order bit in a word.

multichannel buffered serial port (McBSP): An on-chip full-duplex circuit that provides direct serial communication through several channels to external serial devices.

multiplexer: A device for selecting one of several available signals.

N

nonmaskable interrupt (NMI): An interrupt that can be neither masked nor disabled.

O

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

off chip: A state of being external to a device.

on chip: A state of being internal to a device.

P

peripheral: A device connected to and usually controlled by a host device.-

program cache: A fast memory cache for storing program instructions allowing for quick execution.

program memory: Memory accessed through the ‘C6x’s program fetch interface.

R

random-access memory (RAM): A type of memory device in which the individual locations can be accessed in any order.

register: A small area of high speed memory located within a processor or electronic device that is used for temporarily storing data or instructions. Each register is given a name, contains a few bytes of information, and is referenced by programs.

reduced-instruction-set computer (RISC): A computer whose instruction set and related decode mechanism are much simpler than those of microprogrammed complex instruction set computers. The result is a higher instruction throughput and a faster real-time interrupt service response from a smaller, cost-effective chip.

reset: A means of bringing the CPU to a known state by setting the registers and control bits to predetermined values and signaling execution to start at a specified address.

S

synchronous-burst static random-access memory (SBSRAM): RAM whose contents does not have to be refreshed periodically. Transfer of data is at a fixed rate relative to the clock speed of the device, but the speed is increased.

synchronous dynamic random-access memory (SDRAM): RAM whose contents is refreshed periodically so the data is not lost. Transfer of data is at a fixed rate relative to the clock speed of the device.

syntax: The grammatical and structural rules of a language. All higher-level programming languages possess a formal syntax.

T

tag: The 18 most significant bits of the program address. This value corresponds to the physical address of the fetch packet that is in that frame.

timer: A programmable peripheral used to generate pulses or to time events.

W

word: A multiple of eight bits that is operated upon as a unit. For the ‘C6x, a word is 32 bits in length.

Appendix C

Summary of Updates in this Document

This appendix provides a summary of the updates in this version of the document. Updates within paragraphs appear in a **bold typeface**.

Page:	Change or Add:
1–3	Change the last line of code on the page. `unsigned init *dma_ptr = (unsigned int *)DMA_PRIMARY_CTRL_ADDR(1);`
2–3	Change the second sentence of the first paragraph. **Memory-mapped register bit-manipulation macros** are used to control bits and bit fields within the specified register. These macros use four arguments: addr, **val, bit, and length**. Change the LOAD_FIELD bullet. ❑ LOAD_FIELD(addr,**val,bit,length**)

2–8	Change the code in Example 1 of Section 2.2.2, *Cache Support (cache.h)*.
	`CACHE_ENABLE( );`
	Change the code in Example 2 of Section 2.2.2, *Cache Support (cache.h)*.
	`CACHE_DISABLE( );`

Page: **Change or Add:**

2–10 Change the occurrences of XFR in Table 2–8 to XFER and deleted the DMA_GCR_A_ADDR row.

Table 2–8. Timer Mode Values

Register Mnemonic	Register Address Mnemonic
DMA0_PRIMARY_CTRL	DMA0_PRIMARY_CTRL_ADDR
DMA0_SECONDARY_CTRL	DMA0_SECONDARY_CTRL_ADDR
DMA0_SRC_ADDR	DMA0_SRC_ADDR_ADDR
DMA0_DEST_ADDR	DMA0_DEST_ADDR_ADDR
DMA0_**XFER**_COUNTER	DMA0_**XFER**_COUNTER_ADDR
DMA1_PRIMARY_CTRL	DMA1_PRIMARY_CTRL_ADDR
DMA1_SECONDARY_CTRL	DMA1_SECONDARY_CTRL_ADDR
DMA1_SRC_ADDR	DMA1_SRC_ADDR_ADDR
DMA1_DEST_ADDR	DMA1_DEST_ADDR_ADDR
DMA1_**XFER**_COUNTER	DMA1_**XFER**_COUNTER_ADDR
DMA2_PRIMARY_CTRL	DMA2_PRIMARY_CTRL_ADDR
DMA2_SECONDARY_CTRL	DMA2_SECONDARY_CTRL_ADDR
DMA2_SRC_ADDR	DMA2_SRC_ADDR_ADDR
DMA2_DEST_ADDR	DMA2_DEST_ADDR_ADDR
DMA2_**XEFR**_COUNTER	DMA2_**XFER**_COUNTER_ADDR
DMA3_PRIMARY_CTRL	DMA3_PRIMARY_CTRL_ADDR
DMA3_SECONDARY_CTRL	DMA3_SECONDARY_CTRL_ADDR
DMA3_SRC_ADDR	DMA3_SRC_ADDR_ADDR
DMA3_DEST_ADDR	DMA3_DEST_ADDR_ADDR
DMA3_**XFER**_COUNTER	DMA3_**XFER**_COUNTER_ADDR
DMA_GCR_A	DMA_GCR_A
DMA_GCR_B	DMA_GCR_B_ADDR
DMA_GNDX_A	DMA_GNDX_A_ADDR
DMA_GNDX_B	DMA_GNDX_B_ADDR
DMA_GADDR_A	DMA_GADDR_A_ADDR
DMA_GADDR_B	DMA_GADDR_B_ADDR
DMA_GADDR_C	DMA_GADDR_C_ADDR
DMA_GADDR_D	DMA_GADDR_D_ADDR
DMA_GCTRL	DMA_GCTRL_ADDR

Page: **Change or Add:**

2–18 Change the bullet list in Section 2.2.4, *External Memory Interface Support (emif.h, emif.c)*.

- ❏ EMIF_GET_MAP_MODE**( )**
- ❏ SDRAM_INIT**( )**
- ❏ SDRAM_REFRESH_DISABLE**( )**
- ❏ SDRAM_REFRESH_ENABLE**( )**
- ❏ SDRAM_REFRESH_PERIOD(val)

2–21 In Example 1, remove the information found after the hex value that is associated with the code that begins /* SDRAM.

```
/* SDRAM, default TRC TRP TRCD, init SDRAM, refresh enable, 16 bit devices */
#define DEFAULT_EMIF_SDRAM_CTRL    0x07229000   /*                          */
```

Change the hex value associated with the code, found in Example 1, that begins /* SDRAM.

```
/* SDRAM default refresh period                                          */
#define DEFAULT_EMIF_SDRAM_REF     0x00000619
```

Change the first line of code in Example 2.

```
if (EMIF_GET_MAP_MODE( ))
```

2–22 Change the code in Example 1.

```
HPI_RESET_DSPINT( );
```

Change the code in Example 2.

```
HPI_SET_HINT( );
```

2–24 Change the seventh bullet.

- ❏ INTR_GLOBAL_DISABLE

Change the eighth bullet.

- ❏ INTR_GLOBAL_ENABLE

Change the ninth bullet.

- ❏ INTR_MAP_RESET**( )**

2–27 Change the first line of code in Example 2.

```
interrupt void exampleISR(void)
```

2–36 Change the title of Table 2–41(b).

*(b). Transmit and receive frame length (XFRLEN1, **XFRLEN2**, RFRLEN1, **RFRLEN2**)*

Page:	Change or Add:

2–36 Change the Possible Value entry in Table 2–41(b).

Mnemonic	Possible Value
MAX_FRAME_LENGTH	0x7**F**

Change the title of Table 2–41(c).

*(c) Transmit and receive word length (XWDLEN1, **XWDLEN2**, RWDLEN1, **RWDLEN2**)*

Add the following row to the bottom of Table 2–41(c).

MAX_WORD_LENGTH	0x05

2–37 Delete the MAX_WORD_LENGTH row from Table 2–41(d).

Change the Possible Value entries in Table 2–42(a), (b), and (c).

(a) SRGR clock rate divide (CLKGDV)

Mnemonic	Possible Value
MAX_SRG_CLK_DIV	0x**FF**

(b) SRGR frame width (FWID)

Mnemonic	Possible Value
MAX_FRAME_WIDTH	0x**FF**

(c) SRGR frame period (FPER)

Mnemonic	Possible Value
MAX_FRAME_PERIOD	0x0**FFF**

2–38 Change the title of Table 2–42(d).

*(d) SRGR frame sync mode (FSG**M**)*

2–40 Change the sixteenth subbullet of the first bullet and added two other bullets.

- TINP_GET(chan)
- **TOUT_ASSERT(chan)**
- **TOUT_NEGATE(chan)**

Page: **Change or Add:**

2–41 Add the Timer Mode Values table after Table 2–43.

Table 2–44. Timer Mode Values

Mode Mnemonic	Value
TIMER_PULSE_MODE	0
TIMER_CLOCK_MODE	1

2–42 Change the code in the Example.

```
int delay_usec(short numUsec)
{
  unsigned int period_reg;
  unsigned int ctrl_reg = 0;
  int               chan = 0;
  int               cpu_freqInMhz;

  if (!TIMER_AVAILABLE(chan))
  {
    chan++;
    if (!TIMER_AVAILABLE(chan))
      return(-1);
  }


  cpuFreqInMhz = cpu_freq( )     /* returns board CPU freq in Mhz  */

  if (cpu_freqInMhz == ERROR)
  {
    DEBUG("ERROR reading CPU frequency\n\n");
    return(ERROR);
  }

  period_reg = ((cpu_freqInMhz >> 3) * numUsec);
  ctrl_reg = MASK_BIG(C_P) | MASK_BIG(CLKSRC);

  TIMER_INIT(chan,ctrl_reg,period_reg,0);
  TIMER_START(chan);

  /* poll for high-low-high transition                          */
  while (!(TIMER_GET_TSTAT(chan))){NOPS;}           /* TSTAT = 1  */
  while   (TIMER_GET_TSTAT(chan)  {NOPS;}           /* TSTAT = 0  */
  while (!(TIMER_GET_TSTAT(chan))){NOPS;}           /* TSTAT = 1  */

  TIMER_STOP(chan);
  return(0);
}
```

Page: **Change or Add:**

3–2 Change the LOAD_FIELD row in Table 3–1(a).

LOAD_FIELD(addr,**val,bit,length**)	Assigns bits in register at address to value	4-31

3–3 Change four Function listings in Table 3–3.

Table 3–3. Macros and Functions Defined in dma.h and dma.c

Function	Description	Page
DMA_RSYNC_CLR**( )**	Clears the read sync bit in the DMA secondary control register, selecting no synchronization	4-10
DMA_RSYNC_SET**( )**	Sets the read sync bit in the DMA secondary control register, selecting synchronization	4-11
DMA_WSYNC_CLR**( )**	Clears the write sync bit in the DMA secondary control register, selecting no synchronization	4-13
DMA_WSYNC_SET**( )**	Sets the write sync bit in the DMA secondary control register, selecting syncnronization	4-14

3–4 Change four Function listings in Table 3–4.

Table 3–4. Macros and Functions Defined in emif.c and emif.h

Function	Description	Page
EMIF_GET_MAP_MODE**( )**	Returns value of MAP bit in EMIF global control register	4-50
emif_init(g_ctrl,ce0_ctrl,ce1_ctrl,ce2_ctrl, ce3_ctrl,sdram_ctrl,sdram_refresh)	Sets registers in parameter list to values passed in arguments	4-16
SDRAM_INIT**( )**	Initializes SDRAM in each CE space configured for SDRAM	4-49
SDRAM_REFRESH_DISABLE**( )**	Disables SDRAM refresh	4-50
SDRAM_REFRESH_ENABLE**( )**	Enables SDRAM refresh	4-50
SDRAM_REFRESH_PERIOD(val)	Sets SDRAM refresh period	4-51

Page: **Change or Add:**

3–4 Change all the Function listings in Table 3–5.

Table 3–5. Macros and Functions Defined in hpi.h

Function	Description	Page
HPI_GET_DSPINT**( )**	Returns value of DSP interrupt	4-20
HPI_GET_HINT**( )**	Returns value of host interrupt	4-20
HPI_RESET_DSPINT**( )**	Resets DSP interrupt flag generated by host	4-21
HPI_SET_HINT**( )**	Generates HPI interrupt to host	4-21

3–5 Change three Function listings in Table 3–6.

Table 3–6. Macros and Functions Defined in intr.h and intr.c (Continued)

Function	Description	Page
INTR_GLOBAL_DISABLE**( )**	Globally disables all masked interrupts by clearing the GIE bit	4-26
INTR_GLOBAL_ENABLE**( )**	Globally enables all masked interrupts by setting the GIE bit	4-26
INTR_MAP_RESET**( )**	Resets the interrupt multiplexer maps to their default values	4-29

3–8 Change two Function listings and add two Function listings to Table 3–8.

Table 3–8. Macros and Functions Defined in timer.h and timer.c

Function	Description	Page
TIMER_SET_COUNT(chan**,val**)	Sets the value of the timer counter register for the specified channel	4-60
TIMER_SET_PERIOD(chan**,val**)	Sets the value of the timer period register for the specified channel	4-61
TOUT_ASSERT(chan)	**Asserts TOUT pin (high).**	**4-63**
TOUT_NEGATE(chan)	**Negates TOUT pin (low).**	**4-64**

Page:	Change or Add:
4–2	Change the Syntax for CACHE_BYPASS.

Syntax

#include <cache.h>
#define **CACHE_BYPASS( )**

Change the Example for CACHE_BYPASS.

Example

```
#include <cache.h>
CACHE_BYPASS( );
```

4–3 Change the Syntax for CACHE_DISABLE.

Syntax

#include <cache.h>
#define **CACHE_DISABLE( )**

Change the Example for CACHE_DISABLE.

Example

```
#include <cache.h>
CACHE_DISABLE( );
```

Change the Syntax for CACHE_ENABLE.

Syntax

#include <cache.h>
#define **CACHE_ENABLE( )**

Change the Example for CACHE_ENABLE.

Example

```
#include <cache.h>
CACHE_ENABLE( );
```

4–4 Change the Syntax for CACHE_FLUSH.

Syntax

#include <cache.h>
#define **CACHE_FLUSH( )**

Change the Example for CACHE_FLUSH.

Example

```
#include <cache.h>
CACHE_FLUSH( );
```

Change the Syntax for CACHE_FREEZE.

Syntax

#include <cache.h>
#define **CACHE_FREEZE( )**

Change the Example for CACHE_FREEZE.

Example

```
#include <cache.h>
CACHE_FREEZE( );
```

Page: **Change or Add:**

4–5 Change the Example for DMA_DEST_ADDR_ADDR.

Example
```
#include <dma.h>
/* ------------------------------------------------------ */
/*Set destination address for DMA channel register 2*/
/* ------------------------------------------------------ */
* (unsigned int *) DMA_DEST_ADDR_ADDR(2) = (unsigned
int *)0X00400000u;
```

4–6 Change the fourth line in the Example.

```
#include <dma.h>
/* ------------------------------------------------------------------------------ */
/* Initialize DMA global control registers                                        */
/* gcr = 0x1 set aux ctrl register chan pri to give chan 2 priority               */
/* gcra = 0x256 set element count in global count register A                      */
```

4–9 Change the last line of the Example for DMA_PRIMARY_CTRL_ADDR.

```
*(unsigned int *) DMA_PRIMARY_CTRL_ADDR(2) = (unsigned int *)0X2A000A10U;
```

4–11 Change the last two lines of the Example for DMA_SECONDARY_CTRL_ADDR.

```
*(unsigned int *) DMA_SECONDARY_CTRL_ADDR (2) = (unsigned int *)MASK_BIT
(FRAME_IE) | MASK_BIT(SX_IE);
```

4–13 Add the following line to the bottom of the DMA_WSYNC_CLR Example.

```
DMA_WSYNC_CLR(0);
```

4–14 Change the Example for DMA_XFER_COUNTER_ADDR.

Example
```
#include <dma.h>
/* Get address of transfer counter for DMA channel 2*/
unsigned int *xfer_counter = (unsigned int *)
DMA_XFER_COUNTER_ADDR(2);
```

4–15 Change the Syntax for EMIF_GET_MAP_MODE.

Syntax

#include <emif.h>
#define **EMIF_GET_MAP_MODE( )**

Change the Example for EMIF_GET_MAP_MODE.

Example
```
#include <emif.h>
unsigned short map = EMIF_GET_MAP_MODE( );
```

Page: **Change or Add:**

4–20 Change the Syntax for HPI_GET_DSPINT.

Syntax

#include <hpi.h>
#define **HPI_GET_DSPINT( )**

Change the Example for HPI_GET_DSPINT.

Example

```
/*-------------------------------------------------------------------------*/
/* Including hpi.h to gain access to host port interface                   */
/* registers, macros, and functions. *                                     */
/*-------------------------------------------------------------------------*/
while (!(HPI_GET_DSPINT( )) write_to_hpi_mem( );
/* Write to HPI mem until HOST interrupt */
```

Change the Syntax for HPI_GET_HINT.

Syntax

#include <hpi.h>
#define **HPI_GET_HINT( )**

Change the Example for HPI_GET_HINT.

Example

```
/*-------------------------------------------------------------------------*/
/* Include hpi.h to gain access to host port interface                     */
/* registers, macros, and functions                                        */
/*-------------------------------------------------------------------------*/
while (! (HPI_GET_HINT( )));
/* pause until HOST has cleared DSP to HOST interrupt                      */
```

4–21 Change the Syntax for HPI_RESET_DSPINT.

Syntax

#include <hpi.h>
#define **HPI_RESET_DSPINT( )**

Change the Example for HPI_RESET_DSPINT.

Example

```
/*-------------------------------------------------------------------------*/
/* Include hpi.h to gain access to host port interface                     */
/* registers, macros, and functions                                        */
/*-------------------------------------------------------------------------*/
HPI_RESET_DSPINT( );
```

Change the Syntax for HPI_SET_HINT.

Syntax

#include <hpi.h>
#define **HPI_SET_HINT( )**

Page: **Change or Add:**

4–21 Change the Example for HPI_SET_HINT.

Example

```
/*-------------------------------------------------------------------------*/
/* Include hpi.h to gain access to host port interface                     */
/* registers, macros, and functions.                                       */
/*-------------------------------------------------------------------------*/
#include <hpi.h>
HPI_SET_HINT( );
/* Send interrupt to host processor                                        */
```

4–22 Change the Syntax for IDLE.

Syntax #include <cache.h>
#define **IDLE( )**

Change the Example for IDLE.

Example

```
/*-------------------------------------------------------*/
/* Idle CPU                                              */
/*-------------------------------------------------------*/
#include <cache.h>
IDLE( );
```

4–24 Change the Example for INTR_ENABLE.

Example

```
#include <intr.h>
INTR_ENABLE (CPU_INT14);
/* Enable CPU 14 interrupt */
```

4–26 Change the Syntax for INTR_GLOBAL_DISABLE.

Syntax #include <intr.h>
#define **INTR_GLOBAL_DISABLE( )**

Change the Example for INTR_GLOBAL_DISABLE.

Example

```
#include <intr.h>
INTR_GLOBAL_DISABLE( );
      /* Globally disable all interrupts*/
```

Change the Syntax for INTR_GLOBAL_ENABLE.

Syntax #include <intr.h>
#define **INTR_GLOBAL_ENABLE( )**

Change the Example for INTR_GLOBAL_ENABLE.

Example

```
#include <intr.h>
INTR_GLOBAL_ENABLE( );
      /* Globally enable all interrupts                    */
```

Page: **Change or Add:**

4–29 Change the Syntax for INTR_MAP_RESET.

Syntax

#include <intr.h>
#define **INTR_MAP_RESET( )**

Change the Example for INTR_MAP_RESET.

Example

```
#include <intr.h>
INTR_MAP_RESET( );
/* Set low and high interrupt multiplexer registers to default values    */
```

4–30 Change the Example for INTR_SET_FLAG.

Example

```
#include <intr.h>
INTR_SET_FLAG(CPU_INT4);
```

4–31 Change the Syntax for LOAD_FIELD.

Syntax

#include <regs.h>
#define **LOAD_FIELD(**addr,**val,bit,length)**

4–32 Change the Example for MCBSP_ADDR.

Example

```
#include <mcbsp.h>
unsigned int *port0_base_addr = (unsigne-
d int *) MCBSP_ADDR(0);
```

4–33 Change the Example for MCBSP_DRR_ADDR.

Example

```
/*-------------------------------------------------------*/
/* Get address of multi channel buffered                 */
/* serial port 1, receive address reg                    */
/*-------------------------------------------------------*/
#include <mcbsp.h>
unsigned int *ptr = (unsigned int *) MCBSP_DRR_ADDR
(1);
```

4–34 Change the Example for MCBSP_DXR_ADDR.

Example

```
/*-------------------------------------------------------*/
/* Get address of data transmit register                 */
/* for MCBSP port 0.                                     */
/*-------------------------------------------------------*/
#include <mcbsp.h>
unsigned int *ptr=(unsigned int *)MCBSP_DXR_ADDR (0);
```

Page: **Change or Add:**

4–39 Change the Example for MCBSP_MCR_ADDR.

Example
```
/* ------------------------------------------------------ */
/* Get address of MCBSP 1 multi–channel                   */
/* control register                                       */
/* ------------------------------------------------------ */
#include <mcbsp.h>
unsigned int *ptr = (unsigned int *)MCBSP_MCR_ADDR(1);
```

Change the Example for MCBSP_PCR_ADDR.

Example
```
/* ------------------------------------------------------ */
/* Get address of MCBSP 1 pin control reg                 */
/* ------------------------------------------------------ */
#include <mcbsp.h>
unsigned int *ptr =(unsigned int *)MCBSP_PCR_ADDR(1);
```

4–40 Change the Example for MCBSP_RCER_ADDR.

Example
```
/* ------------------------------------------------------ */
/*Get address of MCBSP 1 receive channel enable reg*/
/* ------------------------------------------------------ */
#include <mcbsp.h>
unsigned int *ptr=(unsigned int *)MCBSP_RCER_ADDR(1);
```

Change the Example for MCBSP_RCR_ADDR.

Example
```
/* ------------------------------------------------------ */
/* Get address of MCBSP 1 receive control register */
/* ------------------------------------------------------ */
#include <mcbsp.h>
unsigned int *ptr = (unsigned int *)MCBSP_RCR_ADDR(1);
```

4–43 Change the Example for MCBSP_SPCR_ADDR.

Example
```
/* ------------------------------------------------------ */
/* Get address of MCBSP 0 sample rate generator reg*/
/* ------------------------------------------------------ */
#include <mcbsp.h>
unsigned int *ptr = (unsigned int *)MCBSP_SPCR_ADDR
(0);
```

Page:	Change or Add:

4–44 Change the Example for MCBSP_SRGR_ADDR.

Example
```
/*----------------------------------------------------*/
/* Get address of MCBSP 0 sample rate generator reg*/
/*----------------------------------------------------*/
#include <mcbsp.h>
unsigned int *ptr = (unsigned int *)MCBSP_SRGR_ADDR
(0);
```

4–46 Change the Example for MCBSP_XCR_ADDR.

Example
```
/*----------------------------------------------------*/
/* Get address of MCBSP 1 transmit control register*/
/*----------------------------------------------------*/
#include <mcbsp.h>
unsigned int *ptr=(unsigned int *)MCBSP_XCR_ADDR (1);
```

4–49 Change the Syntax for SDRAM_INIT.

Syntax
```
#include <emif.h>
#define SDRAM_INIT( )
```

Change the Example for SDRAM_INIT.

Example
```
#include <emif.h>
SDRAM_INIT( );
```

4–50 Change the Syntax for SDRAM_REFRESH_DISABLE.

Syntax
```
#include <emif.h>
#define SDRAM_REFRESH_DISABLE( )
```

Change the Example for SDRAM_REFRESH_DISABLE.

Example
```
#include <emif.h>
SDRAM_REFRESH_DISABLE( );
```

Change the Syntax for SDRAM_REFRESH_ENABLE.

Syntax
```
#include <emif.h>
#define SDRAM_REFRESH_ENABLE( )
```

Change the Example for SDRAM_REFRESH_ENABLE.

Example
```
#include <emif.h>
SDRAM_REFRESH_ENABLE( );
```

Page: **Change or Add:**

4–54 Change the Example for TIMER_COUNTER_ADDR.

Example

```
/*----------------------------------------------------------------------*/
/* Get address of timer 1 counter register                              */
/*----------------------------------------------------------------------*/
#include <timer.h>
unsigned int *ptr = (unsigned int *)TIMER_COUNTER_ADDR (1);
```

4–55 Change the Example for TIMER_CTRL_ADDR.

Example

```
/*-------------------------------------------------------- */
/* get address of timer 0 control register                 */
/*-------------------------------------------------------- */
#include <timer.h>
unsigned int *timer_ctrl = (unsigned int*)
TIMER_CTRL_ADDR(0);
```

4–57 Change the Syntax of TIMER_INIT.

Syntax

#include <timer.h>
#define **TIMER_INIT(**chan,ctrl,per,**cnt)**

Add a fourth bullet to the Description of TIMER_INIT.

- cnt: value to set timer counter register

Change the Example of TIMER_INIT.

Example

```
#include <timer.h>
/*----------------------------------------------------------------------*/
/* Configure timer 0 as timer pin                                       */
/*----------------------------------------------------------------------*/
      TIMER_INIT (0, 0x1, 0xFFFF, 0);
/* Configure timer 0 as timer pin and set timer period to 0xFFFF      */
```

4–58 Change the Example for TIMER_PERIOD_ADDR.

Example

```
/*-------------------------------------------------------- */
/* get address of timer 1 period register                  */
/*-------------------------------------------------------- */
#include  <timer.h>
unsigned int *ptr = (unsigned int *)TIMER_PERIOD_ADDR
(1);
```

4–60 Change the Example for TIMER_SET_COUNT.

Example

```
#include <timer.h>
TIMER_SET_COUNT(0,256);
```

Page:	Change or Add:
4–63	Add the TOUT_ASSERT instruction.

TOUT_ASSERT *Asserts 1 on TOUT pin*

Syntax

#include <timer.h>
#define **TOUT_ASSERT(**chan**)**

Defined in

timer.h as a macro

Description

TOUT_ASSERT writes a 1 to the TOUT pin of the specified channel. It uses the following parameter:

- ❑ chan: channel selector (0,1)

Example

```
#include <timer.h>
TOUT_ASSERT(0);     /*Write a 1 to the TOUT
pin of timer 0 */
```

4–64 Add the TOUT_NEGATE instruction.

TOUT_NEGATE *Asserts 0 on TOUT pin*

Syntax

#include <timer.h>
#define **TOUT_NEGATE(**chan**)**

Defined in

timer.h as a macro

Description

TOUT_NEGATE writes a 0 to the TOUT pin of the specified channel. It uses the following parameter:

- ❑ chan: channel selector (0,1)

Example

```
#include <timer.h>
TOUT_NEGATE (0);      /* Write a 0 to the
TOUT pin of timer 0 */
```

Page:	Change or Add:
A–1	Change the paragraph. This appendix provides the entire code listing for each source file contained in the 'C6x peripheral support library. **However, the code provided to you may be a more recent version than the one that is listed here.** For more information on the source files and the macros and functions that comprise them, see Chapter 2.
A–2	Replace sections A.1 and A.2.
B–4	Change the index definition. **index**: A **relative offset** in the program address that specifies which of the 512 frames in the cache into which the current access is mapped.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

W