Application Report
SPRA641 - March 2000

TMS320C5000 DMA Applications

Ramesh A. Iyer *DSP West Applications*

ABSTRACT

A Direct Memory Access (DMA) controller is available on select members of the TMS320C5000 family of digital signal processors (DSPs). The DMA controller is used to transfer data to/from internal memory, internal peripherals, or external devices, independent of the CPU. There are six independent DMA channels available, allowing six different contexts for DMA operation.

This application report explains and provides source code for the following common DMA applications:

- Data sorting
- Program paging
- Data transfer using a ping-pong scheme
- DMA interrupt service routines

Contents

List of Figures

1 Introduction

Configuration and operation of the direct memory access (DMA) is achieved using a set of memory-mapped control registers. The C5000 devices use a sub-addressing scheme to access the DMA registers, which allows a large number of registers to be mapped into a small memory space.

There are two types of sub-bank access registers. The first, Sub-bank Access Register with Autoincrement (DMSDI), allows automatic increment of the sub-address. The autoincrement feature is a convenient and efficient method to use when the entire set of DMA configuration registers (or even a subset of consecutive configuration registers) needs to be configured. The second, Sub-bank Access Register without Autoincrement (DMSDN), is used if only a single register has to be accessed. Using DMSDN does not modify the sub-address.

To access a specific DMA sub-addressed register, the desired sub-address is written to the DMA sub-bank address register (DMSA). Next, the desired value can be written to or read from one of the two sub-bank access registers, DMSDI or DMSDN.

The entire register set used to program the DMA is contained in the four memory-mapped registers: DMPREC (Channel Priority and Enable Control Register), DMSA, DMSDI, and DMSDN. For a full list of all the DMA registers, see reference [3].

Each DMA channel has the following set of registers that must be configured prior to a data transfer:

- Source Address Register – the memory location that the element is transferred from
- Destination Address Register – the memory location that the element is transferred to
- Element Count Register – specifies the number of DMA transfers to be performed
- Sync Select and Frame Count Register – specifies the word size of an element to be transferred, the number of frames to be transferred, and the synchronization events that can be used to trigger a DMA transfer
- Transfer Mode Control Register – controls the transfer mode of the DMA channel

The element count register is a 16-bit register and must be initialized with one less than the desired number of element transfers. The frame count register is an 8-bit field in the DMA Sync Event and Frame Count register (DMSFC) and must be initialized with one less than the desired number of frames. Therefore, every DMA channel can transfer a maximum of 64K elements $\times$ 256 frames = 16777216 elements or, 16M elements.

In addition to these registers, there are four global DMA registers that can be used to reload the DMA configuration registers at the end of the current transfer:

- Global Source Address Reload Register
- Global Destination Address Reload Register
- Global Element Count Reload Register
- Global Frame Count Reload Register

The DMA controller also has the ability to perform transfers to and from the extended program memory space (capability to transfer to/from program memory is available only on select C5000 devices – consult the data sheet for the appropriate device). The two sub-addressed registers used to provide this capability are:

- Source program page address register (DMSRCP)
- Destination program page address register (DMDSTP)

A DMA transfer can be programmed to be postincremented by 1 or postincremented by the offset value contained in one of the two element index registers (DMIDX0 or DMIDX1), and/or one of the two frame index registers (DMFRI0 or DMFRI1).

At the time this application report and its examples were created, only the 5410, 5409, and 5421 devices support DMA accesses to internal and external memory; all the other C5000 devices with DMA capability (5402, 5416, and 5420) support only internal DMA accesses. See reference [1] for the DMA memory map of the appropriate C5000 device. The DMA memory map is not affected by the state of the MP/MC, DROM, or OVLY bits.

2 System Structure

All the examples in this application report were tested on a 5410 simulator and a 5410 target system from DSP Research. To adapt these examples to your specific C5000 device, consult the relevant data sheets for the appropriate memory map, DMA memory map, and specific mapping of interrupt vectors.

The dma.h header file included in the main() program contains a definition of the structures used to program the DMA configuration registers. The structure, dma, is used to pass the following arguments to the child assembly program that handles initialization of the DMA registers:

- DMA channel number
- Source address
- Destination address
- Element count
- DMA sync event and frame count (DMSFC)
- DMA transfer mode control (DMMCR)
- Extended source program page
- Extended destination program page
- Global source address reload
- Global destination address reload
- Global element count reload
- Global frame count reload
- DMA channel priority and enable control (DMPREC)

DMA events can be used to trigger interrupts to the CPU upon completion of a transfer. Due to a limited number of interrupts in the C54x memory map, some of the DMA interrupts are multiplexed with other interrupts on the device. This is controlled by the INTOSEL field (bits 7 and 6) in the DMPREC register.

Since the bit fields in the DMPREC register selectively enable/disable DMA channels and assign priorities to them, it is important that this register be the last register to be written to, so that you are assured that the rest of the configuration registers have already been written to.

3 Data Sorting

Data sorting is useful in applications requiring data in multiple data arrays to be arranged such that the first elements of each array are contiguous, the second elements of each array are contiguous, and so on. This means that the data is sorted and stored by element number instead of frame number. Figure 1 demonstrates this concept. The input data could come from a peripheral or could be arranged in memory with each array arranged as contiguous frames. In these cases, the services of a DMA channel can be used to rearrange the data into the desired format.

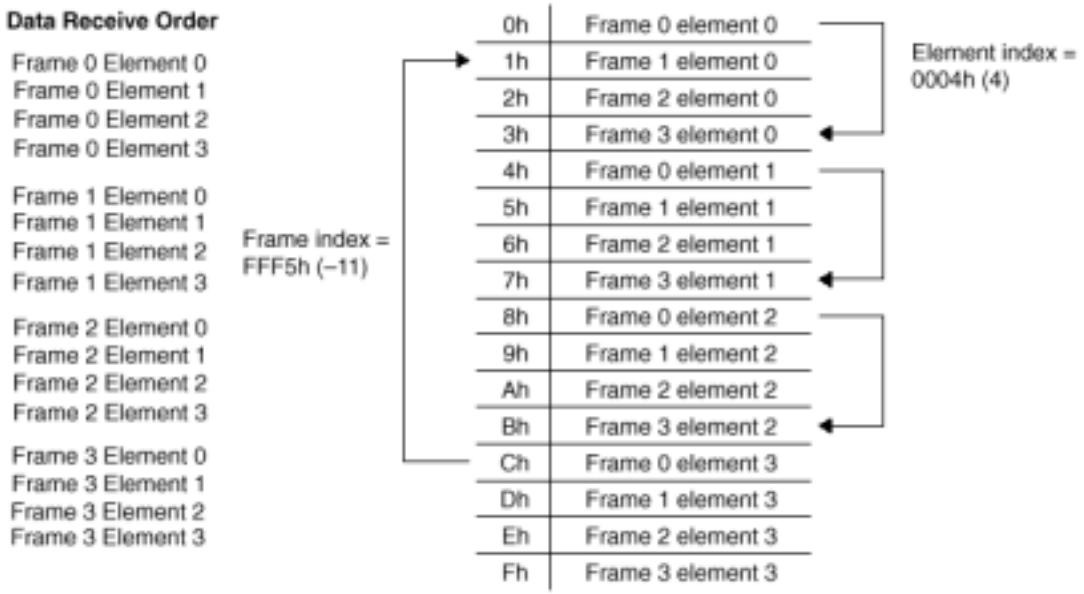

Figure 1. Demonstrating Data Sorting

Sorting is achieved by using the address modification capability of the DMA via the element index and frame index registers. The following formula can be used to determine the values to be loaded into the element and frame index registers:

Element Index = Number of frames programmed in Frame Count field + 1

Frame Index = –(Element Index $\times$ Number of elements programmed in Element Count Register – 1)

As an example, assume a data stream where the data is structured as four elements per frame and four frames per block. The source data arrives in the order shown on the left in Figure 1. For this example, the element count register and frame count field contain the following values:

Element count register = 3

Frame count field = 3

To sort this data, the element index register is programmed to *increment* the destination address by 4 (= Element count register + 1) after each transfer, and the frame index register is programmed to *decrement* the destination address by 11 (= 4 $\times$ 3 – 1) after each frame. When the entire block of 16 transfers has been completed, the data is sorted by element number as shown on the right in Figure 1.

For the application example, we assume that the data is already preinitialized in data memory beginning at location 1000h. This is achieved by using the load= option that loads the .data section at address 1000h in the linker command file. We specify DMA channel 0, and operate the DMA in the multiframe, unsynchronized mode with interrupts disabled. The source address is programmed for no modification and the destination address is programmed to postincrement with index offsets supplied by the DMIDX0 and DMFRI0 registers. Since the data to be sorted is structured as multiple elements in a frame and multiple frames in a block, the requisite DMA channel must be used in the Multiframe mode. The global reload registers contain 0s since we assume that this is a one-time operation.

The following values are passed by the structure, dma:

DMA Channel number = 0

Source address = 1000h

Destination address = 1F00h

Element count = 3

DMSFC0 (Figure 2) = 0000 0000 0000 0011b = 0003h

15–12	11	10–8	7–0
0000	0	000	0000 0011
DSYN[3:0]	DBLW	Reserved	FRAME COUNT

Figure 2. DMSFC Register for Data Sort Example

DMMCR0 (Figure 3) = 0000 0000 0101 0101b = 55h

15	14	13	12	11	10–8	7–6	5	4–2	1–0
0	0	0	0	0	000	01	0	101	01
AUTO INIT	DINM	IMOD	CTMOD	Rsvd	SIND	DMS	Rsvd	DIND	DMD

Figure 3. DMMCR Register for Data Sort Example

Extended source program space = 00

Extended destination program space = 00

Global source address reload = 0

Global destination address reload = 0

Global element count reload = 0

Global frame count reload = 0

DMPREC (Figure 4) = 0000 0001 0000 0001b = 0101h

15	14	13–8	7–6	5–0
0	0	00 0001	00	00 0001
FREE	Resvd	DPRC	INTOSEL	DE[5:0]

Figure 4. DMPREC Register for Data Sort Example

The child assembly program that handles data sorting loads the element and frame indices with necessary values based on the previous equation. Enabling the DMA channel field in the DMPREC register activates the DMA channel and starts the data sort process.

4 Program Paging

Program paging becomes necessary when all the code cannot be fitted into on-chip memory and it is not desirable to operate the code from external memory. A DMA channel can be programmed to bring in a block of code from external memory to internal program memory. For efficient utilization of the CPU, it is necessary to have at least two sections of on-chip program memory so that while the DMA channel is filling up one section, the CPU can execute from the other section. Program paging is usually a low-priority task, so the priority of the DMA channel can be programmed accordingly.

The blocks of code to be paged in are resident in external program memory. In this example, there are four distinct blocks of code, each resident in its own section in external program memory. The memory map for this example includes the following sections (as shown in the linker command file for this example):

- Interrupt service table linked to run from internal overlaid single-access RAM at location 2000h. This section contains the interrupt vector table.
- A main block of code linked to run from P_DARAM in internal program memory. This section contains the main() subroutine.
- The child assembly subroutine pgm_page.asm, linked to run from P_DARAM in internal program memory. This section programs the DMA configuration registers.
- The assembly routine pgm_page_isr.asm, linked to run from P_DARAM in internal program memory. This section contains the interrupt service routine, and determines the program address of the block of code that the CPU must start executing.

- The section, test1, linked to load to external program memory address 0x1_0100h, and run from internal program memory address 0x1_8000h.
- The section, test2, linked to load to external program memory address 0x1_0200h, and run from internal program memory address 0x1_8100h.
- The section, test3, linked to load to external program memory address 0x1_0300h, and run from internal program memory address 0x1_8000h.
- The section, test4, linked to load to external program memory address 0x1_0400h, and run from internal program memory address 0x1_8100h.

The length of each section (test1, test2, test3, and test4) is computed at run time by using the .label directive in each of these sections (for more details on using this directive, see reference [4]). The DMA channels are programmed to transfer each of these sections to their run-time program space, sequentially. At the end of each of these sections, the CPU is programmed to idle; the CPU wakes up from idle on receipt of a DMA interrupt that signals the successful completion of the next section to internal program memory. Alternatively, you may also design to have the program wait in an idle thread.

Particular attention must be paid to the location of the interrupt vector table, especially when blocks of code that are paged in from external memory are scheduled to run from *extended on-chip program memory*. In such a case, the interrupt vector table must be made visible to all extended program memory pages by turning overlay on and relocating the interrupt vector table. By the same token, since the program address of the section to branch to may reside in extended on-chip program memory, Far instructions are used in the vector table. For more information about creating and relocating interrupt vector tables and interrupt service routines, see reference [5].

The global reload registers are reloaded at the beginning of each section, test1, test2, test3, and test4, scheduled to be paged in. As an example, when the CPU is executing the section of code test1.asm, the DMA global reload registers are programmed to reload the section test3, while the DMA channel is already busy fetching the section test2. Similarly, when the CPU is executing the section of code test2.asm, the DMA global reload registers are programmed to reload the section test4, while the DMA channel is already busy fetching the section test3.

Since the blocks of code are loaded into one portion of memory and run from a different portion of memory, you must use the load=, run=, option in the linker command file for each of the blocks paged in. Since the sections test1 and test3 run from the same on-chip address 0x1_8000, and the sections test2 and test4 run from the same on-chip address 0x1_8100, you must also use the Union option in the linker command file.

Since program blocks are fetched from external memory, it is important that you initialize the software wait-state register (SWSR) with the appropriate number of wait states depending on the speed of the external memory used. The author initialized the SWSR in the initialization file, init.gel, of Code Composer. *For embedded applications, you must explicitly initialize the wait-state register(s) inside your software.*

The following values are passed via the structure, dma:

DMA Channel number = 0

Source address = 0100h

Destination address = 8000h

Element count = computed at run time using the .label directives

DMSFC0 (Figure 5) = 0000 0000 0000 0000b = 0000h

15–12	11	10–8	7–0
0000	0	000	0000 0000
DSYN[3:0]	DBLW	Reserved	FRAME COUNT

Figure 5. DMSFC Register for Program Paging Example

DMMCR0 (Figure 6) = 1100 0001 0000 0101b = C105h

15	14	13	12	11	10–8	7–6	5	4–2	1–0
1	1	0	0	0	001	00	0	001	01
AUTO INIT	DINM	IMOD	CTMOD	Rsvd	SIND	DMS	Rsvd	DIND	DMD

Figure 6. DMMCR Register for Program Paging Example

Extended source program space = 01

Extended destination program space = 01

Global source address reload = 200h

Global destination address reload = 8100h

Global element count reload = as computed at run time using the .label directives

Global frame count reload = 0

DMPREC (Figure 7) = 0000 0001 1000 0001b = 0181h

15	14	13–8	7–6	5–0
0	0	00 0001	10	00 0001
FREE	Resvd	DPRC	INTOSEL	DE[5:0]

Figure 7. DMPREC Register for Program Paging Example

This example only demonstrates the concept of using a DMA channel to perform program paging without taking into account other system-level considerations. An important consideration to keep in mind (that has not been covered in this example) is the handling of a DMA interrupt, signaling the end of transfer of a block of code to on-chip memory, before the DSP finishes execution of the current block of code. Another aspect is the initialization of the software wait-state register, and the overlay (OVLY) and MP/MC bits in the PMST register that control the memory map of the device.

5 Data Transfer Using a Ping-Pong Scheme

The standard circular buffer scheme that is often used to bring in data is inefficient because the DSP CPU must always wait for the DMA channel to finish fetching a block of data before it can start processing. You could work around this by letting the DSP and DMA access the same buffer simultaneously by ensuring that the DSP does not read data faster than the rate at which data is brought in by the DMA channel. A way of improving efficiency is to have a double-buffered scheme. In this scheme, the DMA moves data to and from one pair of input/output buffers, while the DSP CPU is operating on the second pair.

In the application example listed here, the DMA transfers 28 words from a table defined in the .data section into a buffer starting at address 2000h. After this set of data is transferred, the DMA starts transferring the same data buffer to address 4000h. The next pair of transfers will return to address 2000h. The CPU uses the input data following the first transfer, and stores the result to memory. Once completed, the CPU uses the second input buffer, and stores the result to memory. Note that this example uses the same stored values from initialized memory as a means to demonstrate the concept. In real–life applications, data would be brought in by way of the serial port or external memory or the host port interface (HPI), for processing by the DSP, before being written to another pair of output buffers serviced by another DMA channel in ping-pong mode.

The following values are passed via the structure, dma:

DMA Channel number = 0

Source address = 1000h

Destination address = 2000h

Element count = 31

DMSFC0 (Figure 8) = 0000 0000 0000 0000b = 0000h

15–12	11	10–8	7–0
0000	0	000	0000 0000
DSYN[3:0]	DBLW	Reserved	FRAME COUNT

Figure 8. DMSFC for Ping Pong Example

DMMCR0 (Figure 9) = 1100 0001 0100 0101b = C105h

15	14	13	12	11	10–8	7–6	5	4–2	1–0
1	1	0	0	0	001	01	0	001	01
AUTO INIT	DINM	IMOD	CTMOD	Rsvd	SIND	DMS	Rsvd	DIND	DMD

Figure 9. DMMCR for Ping Pong Example

Extended source program space = 00

Extended destination program space = 00

Global source address reload = 1000h

Global destination address reload = 4000h

Global element count reload = 31

Global frame count reload = 0

DMPREC (Figure 10) = 0000 0001 1000 0001b = 0181h

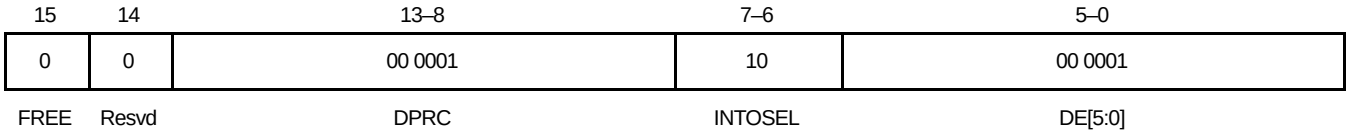

Figure 10. DMPREC for Ping Pong Example

6 DMA Interrupt Service Routines

The DMA interrupt service routines used in the examples in this application note are generic and make assumptions about the final system. In practice, be sure to perform context saves and restores of status registers and any other registers that are affected.

Particular attention must also be paid to the structure of the vector table, a generic version of which is listed. Since the DMA interrupts are multiplexed with other peripheral interrupts, consult references [1] and [3], or the data sheet for the vector table of the relevant C5000 DSP.

7 References

1. *TMS320C54x DSP Reference Set, Volume 1: CPU and Peripherals*, SPRU131
2. *TMS320C54x DSP Reference Set, Volume 2: Mnemonic Instruction Set*, SPRU172
3. *TMS320C54x DSP Reference Set, Volume 5: Enhanced Peripherals*, SPRU302
4. *TMS320C54x Assembly Language Tools User's Guide*, SPRU102
5. *Interrupt Handling Using Extended Addressing of the TMS320C54x Family*, SPRA492

Appendix A Source Code

This appendix contains source code for all the examples listed in this application note.

A.1 Data Sorting

```
/* FILENAME:        main.c                                          */
/*                                                                  */
/* DESCRIPTION:     This is the main program that passes            */
/*                  arguments to the underlying asm program         */
/*                  that handles data sorting.                      */
/*                                                                  */
/* ARGUMENTS:       Arguments to the asm program are passed         */
/*                  via Accumulator A.                              */
/*                  In this example, all arguments are              */
/*                  passed via the structure called ”dma”           */
/*                  Also, source and destination addresses          */
/*                  are hard–coded inside main().  This is          */
/*                  not necessarily good coding practice,           */
/*                  but I’ve used it nevertheless.                  */
/*                                                                  */
/* RETURN:          None                                            */
/*                                                                  */
/* AUTHOR:          Ramesh A. Iyer                                  */
/*                                                                  */
/* CREATED:         June 7, 1999                                    */
/*                                                                  */
/* VERSION:         1.0                                             */
/*                                                                  */
#include ”dma.h”
extern void datasort();
void main()
{
/* Assign a dma channel                                             */
dma.dma_chan = dma_channel_0;
/* Assign a starting address                                        */
dma.src_address = 0x1000;
/* Assign a destination address                                     */
dma.dst_address = 0x1f00;
/* Assign an element count                                          */
dma.element_count = 3;
/* Initialize DMSFC register                                        */
dma.DMSFC.frame_count = 3;
```

```
dma.DMSFC.double_word = 0;
dma.DMSFC.dmasyncevent = 0;
dma.DMSFC.rsvd = 0;
/* Initialize DMMCR register                                    */
dma.DMMCR.dma_dest_addr_space_select = 1;
dma.DMMCR.dma_dest_addr_index_mode = 5;
dma.DMMCR.dma_src_addr_space_sel = 1;
dma.DMMCR.dma_src_addr_index_mode = 1;
dma.DMMCR.dma_transfer_cntr_mode = 0;
dma.DMMCR.dma_int_gen_mode = 0;
dma.DMMCR.dma_int_gen_mask = 0;
dma.DMMCR.dma_autoinit = 0;
dma.DMMCR.rsvd = 0;
dma.DMMCR.reserved = 0;
/* Initialize DMPREC register                                   */
dma.DMPREC.dma_chan0_enable = 1;
dma.DMPREC.dma_chan1_enable = 0;
dma.DMPREC.dma_chan2_enable = 0;
dma.DMPREC.dma_chan3_enable = 0;
dma.DMPREC.dma_chan4_enable = 0;
dma.DMPREC.dma_chan5_enable = 0;
dma.DMPREC.int_mux_control = 0;
dma.DMPREC.dma_chan0_priority = 1;
dma.DMPREC.dma_chan1_priority = 0;
dma.DMPREC.dma_chan2_priority = 0;
dma.DMPREC.dma_chan3_priority = 0;
dma.DMPREC.dma_chan4_priority = 0;
dma.DMPREC.dma_chan5_priority = 0;
dma.DMPREC.free = 0;
dma.DMPREC.rsvd = 0;
/* Call the datasort function                                   */
datasort(dma);
/* Do something useful or wait forever                          */
for (;;);
}
/* End of main()                                                */
```

```
;  FUNCTION:      dma_datasort.asm
;
;  DESCRIPTION:   This function uses one of the DMA channels to sort data
;                 as it is transferred to data memory.
;                 The DMA sorts the incoming data by modifying the
;                 destination addresses with the element index register
;                 (DMIDX0) and the frame index register (DMFRI0).
;                 This routine calculates the necessary values of DMIDX0
;                 and DMFRI0 based on the number of elements and frames.
;
;                 The resulting data is sorted by element number instead of
;                 frame number.
;
;                 Note that the DMA channels are programmed with interrupts
;                 disabled
;
;  ARGUMENTS:     None
;
;  AUTHOR:        Ramesh A. Iyer
;
;  CREATED:       June 7, 1999
;
;  VERSION:       1.0
   .global _datasort
   .include ”dmaregs.h”
   .mmregs
   .bss dmidx, 1
   .bss dmfri, 1
   .text
_datasort:
   FRAME #–(frame)                ; Allocate frame to save local vars
   STLM  A, AR0                   ; AR0 contains the starting address of the
                                  ; structure that was passed
   nop
   nop
   mac   *ar0+, offset, B
   STLM  B, DMSA
   LD    *AR0+, B                 ; Get source address
   stlm  B, DMSDI
   ld    *AR0+, B                 ; Get destination address
```

```
stlm  B, DMSDI
ld    *AR0+, B                ; Get element count
stlm  B, DMSDI
stlm  B, T                    ; Save temporary copy in T register also !
ld    *AR0+, B                ; Get DMSFC
stlm  B, DMSDI
and   #0ffh, B                ; Extract frame count
add   #1, B                   ; Actual number of frames (num_frame) = frame
                              ; count + 1
stl   B, dmidx
mpyu  dmidx, B                ; B = num_frame * element count
sub   #1, B, B                ; B = (num_frame * element count) – 1
neg   B                       ; Compute signed equivalent i.e. –(x)
stl   B, dmfri

ld    *AR0+, B                ; Get DMMCR
stlm  B, DMSDI

stm   DMIDX0, DMSA            ; Write to DMIDX0
mvdk  dmidx, DMSDN
stm   DMFRI0, DMSA            ; Write to DMFRI0
mvdk  dmfri, DMSDN
LD    *AR0+, B                ; Get DMPREC
stlm  B, DMPREC
FRAME #3                      ; Deallocate frame
ret
```

```
;  FUNCTION:         meminit.asm
;
;  DESCRIPTION:      Contains only an initialization table that is used to
;                    test the code...
;
;  ARGUMENTS:        None
;
;  AUTHOR:           Ramesh A. Iyer
;
;  CREATED:          June 27, 1999
;
;  VERSION:          1.0
         .def _meminit
_meminit:
         .data
         .word    0d000h, 0d001h, 0d002h, 0d003h
         .word    0d004h, 0d005h, 0d006h, 0d007h
         .word    0d008h, 0d009h, 0d00ah, 0d00bh
         .word    0d00ch, 0d00dh, 0d00eh, 0d00fh
```

```
/* FILENAME:         dma.cmd                                          */
/*                                                                    */
/* DESCRIPTION:      This is the linker command file                  */
/*                   for the datasort example                         */
/*                                                                    */
/* AUTHOR:           Ramesh A. Iyer                                   */
/*                                                                    */
/* CREATED:          June 7, 1999                                     */
/*                                                                    */
/* VERSION:          1.0                                              */
/*                                                                    */
MEMORY
      {
         PAGE 0:
               P_DARAM: org = 80h              len = 1000h
               VECS:    org = 0ff80h           len = 80h
         PAGE 1:
               D_SPRAM: org = 0060h            len = 0020h
               D_DARAM: org = 80h              len = 3f7fh
      }
SECTIONS
      {
         .text    :>    P_DARAM                PAGE 0
         .bss     :>    D_DARAM                PAGE 1
         .stack   :>    D_DARAM                PAGE 1
         .data    :     load = 0x1000          PAGE 1
         vectors  :>    VECS                   PAGE 0
   }
```

A.2 Program Paging

```
/* FILENAME:         main.c                                          */
/*                                                                   */
/* DESCRIPTION:      This is the main program that passes            */
/*                   arguments to the underlying asm program         */
/*                   that handles program paging.                    */
/*                                                                   */
/* ARGUMENTS:        Arguments to the asm program are passed         */
/*                   via Accumulator A.                              */
/*                   In this example, all arguments are              */
/*                   passed via the structure called ”dma”           */
/*                   Also, source and destination addresses          */
/*                   are hard–coded inside main(). This is           */
/*                   not necessarily good coding practice,           */
/*                   but I’ve used it nevertheless.                  */
/*                                                                   */
/* RETURN:           None                                            */
/*                                                                   */
/* AUTHOR:           Ramesh A. Iyer                                  */
/*                                                                   */
/* CREATED:          June 17, 1999                                   */
/*                                                                   */
/* VERSION:          1.0                                             */
/*                                                                   */
#include ”dma.h”
extern   void pgm_page();
extern   test2_start;
extern   test2_end;
extern   test1_start;
extern   test1_end;
void main()
{
/* Assign a dma channel                                              */
dma.dma_chan = dma_channel_0;
/* Assign a starting address                                         */
dma.src_address = 0x0100;
/* Assign an extended page to source address                         */
dma.ext_src_pgm_page = 1;
/* Assign an extended page to destination address                    */
dma.ext_dst_pgm_page = 1;
/* Assign a destination address                                      */
dma.dst_address = 0x8000;
/* Assign an element count                                           */
dma.element_count = (&test1_end – &test1_start – 1);
```

```
/* Initialize DMSFC register                                          */
dma.DMSFC.frame_count = 0;
dma.DMSFC.double_word = 0;
dma.DMSFC.dmasyncevent = 0;
dma.DMSFC.rsvd = 0;
/* Initialize DMMCR register                                          */
dma.DMMCR.dma_dest_addr_space_select = 1;
dma.DMMCR.dma_dest_addr_index_mode = 1;
dma.DMMCR.dma_src_addr_space_sel = 0;
dma.DMMCR.dma_src_addr_index_mode = 1;
dma.DMMCR.dma_transfer_cntr_mode = 0;
dma.DMMCR.dma_int_gen_mode = 0;
dma.DMMCR.dma_int_gen_mask = 1;
dma.DMMCR.dma_autoinit = 1;
dma.DMMCR.rsvd = 0;
dma.DMMCR.reserved = 0;
/* Initialize DMPREC register                                         */
dma.DMPREC.dma_chan0_enable = 1;
dma.DMPREC.dma_chan1_enable = 0;
dma.DMPREC.dma_chan2_enable = 0;
dma.DMPREC.dma_chan3_enable = 0;
dma.DMPREC.dma_chan4_enable = 0;
dma.DMPREC.dma_chan5_enable = 0;
dma.DMPREC.int_mux_control = 2;
dma.DMPREC.dma_chan0_priority = 1;
dma.DMPREC.dma_chan1_priority = 0;
dma.DMPREC.dma_chan2_priority = 0;
dma.DMPREC.dma_chan3_priority = 0;
dma.DMPREC.dma_chan4_priority = 0;
dma.DMPREC.dma_chan5_priority = 0;
dma.DMPREC.free = 0;
dma.DMPREC.rsvd = 0;
/* Initialize global reload register values                           */
dma.global_src_addr_reload = 0x200;
dma.global_dst_addr_reload = 0x8100;
dma.global_element_cnt_reload = (&test2_end – &test2_start –1);
dma.global_frame_cnt_reload = 0;
pgm_page(dma);
/* Do something                                                       */
for (;;);
}
/* End of main()                                                      */
```

```
;  FUNCTION:         pgm_page.asm
;
;  DESCRIPTION:      Handles the transfer of code from external program
;                    memory to on–chip program memory via a dedicated
;                    DMA channel, in the background.  This scheme is
;                    also called program paging.
;                    The calling program passes all the necessary
;                    arguments via the structure called ”dma”
;
;                    This program does a few additional things besides
;                    initializing the DMA specific registers:
;
;                    a) Turn on interrupts
;
;                    b)    Saving a copy of the destination address,
;                          destination address page, global reload
;                          destination address and global reload destination
;                          address page.
;
;
;  AUTHOR:           Ramesh A. Iyer
;
;  CREATED:          June 17, 1999
;
;  VERSION:          1.0
   .global _pgm_page
   .include ”dmaregs.h”
   .ref test1
   .bss dst_addr, 2, 1
   .bss global_dst_addr, 2, 1
   .global dst_addr
   .global global_dst_addr
   .ref vec_tbl_start, vec_tbl_end
   .mmregs
   .sect ”pgm_page”
_pgm_page:
; Enable interrupts before you start doing anything
   stm   #0040h, IMR             ; Turn on interrupt for dma channel 0
   rsbx  intm
; Initialize AR2 and AR3 which will be used to keep track of the
; destination addresses – this will help the program determine where to
; branch to as soon as the DMA signals an interrupt to the CPU
; indicating completion of transfer of the existing block of code.
   stm   #dst_addr, AR2
   stm   #global_dst_addr, AR3
```

```
; The actual usage of the arguments passed via the structure ”dma”
; starts from here...
   STLM  A, AR0                ; AR0 contains the starting address of
                               ; the structure that was passed
   nop
   nop
   mac   *ar0+, offset, B
   STLM  B, DMSA               ; Write effective offset i.e. starting
                               ; address into DMSA
   LD    *AR0+, B              ; Get source address
   stlm  B, DMSDI
   ld    *AR0+, B              ; Get destination address
   stlm  B, DMSDI
   stl   B, *AR2+              ; Save a copy of the destination address
   ld    *AR0+, B              ; Get element count
   stlm  B, DMSDI
   ld    *AR0+, B              ; Get DMSFC
   stlm  B, DMSDI
   ld    *AR0+, B              ; Get DMMCR
   stlm  B, DMSDI
   stm   DMSRCP, DMSA          ; Initialize to point to DMSRCP
   LD    *AR0+, B              ; Get extended src. pgm address
   STLM  B, DMSDI
   LD    *AR0+, B              ; Get extended dest. pgm address
   STLM  B, DMSDI
   stl   B, *AR2+              ; Save a copy of the extended destination
                               ; program address
   stl   B, *+AR3              ; Keep a copy of the extended pgm address
   stm   DMGSA, DMSA           ; Initialize to point to DMGSA
   LD    *AR0+, B              ; Get global src. address reload value
   STLM  B, DMSDI
   LD    *AR0+, B              ; Get global dest.address reload value
   STLM  B, DMSDI
   mar   *AR3–                 ; Save a copy of the global dest. Address
                               ; too...
   stl   B, *AR3
   LD    *AR0+, B              ; Get global element count reload value
   STLM  B, DMSDI
   LD    *AR0+, B              ; Get global frame count reload value
   STLM  B, DMSDI
   LD    *AR0+, B              ; Get DMPREC
   stlm  B, DMPREC
   idle  1                     ; The CPU idles until it is interrupted
                               ; by the DMA
```

```
;  FUNCTION(s):      test1.asm, test2.asm, test3.asm, test4.asm
;
;  DESCRIPTION:      These are test files that are used to
;                    demonstrate the program paging scheme.
;
;                    Remember to accomplish the following in each of the
;                    files:
;
;                    a) Make sure that interrupts are still on...
;
;                    b) Reload the DMA global reload registers
;
;                    c) Keep a copy of the global reload destination address
;                     in memory
;
;  AUTHOR:           Ramesh A. Iyer
;
;  CREATED:          June 18, 1999
;
;  VERSION:          1.0
;

;  FUNCTION:         test1.asm
   .sect ”test1”
   .ref test2
   .def test1
   .include ”global_reload.h”
   .include ”dmaregs.h”
   .global _test1_start, _test1_end
   .ref test3_start, test3_end
   .ref global_dst_addr
   .mmregs

   .label _test1_start
test1:
   rsbx  INTM                    ; Turn on the interrupts...
; Initialize global reload registers for source and destination addresses for
; test3.asm
   stm   DMGSA, DMSA             ; Initialize to point to DMGSA
   stm   #SRC_3, DMSDI
```

```
   stm   #DST_3, DMSDI
   stm   #global_dst_addr, AR3
   st    #DST_3, *AR3            ; Copy to global_dst_addr placeholder
   stm   #(test3_end – test3_start –1 ), DMSDI
; Proceed with the rest of the code & do something…
   stm   #10h, BRC
   stm   #2000h, AR2
   STM   #1234h, T
   nop
   ld    #0DEADh, A
   idle  1                       ; Idle until DMA interrupts CPU
   .label _test1_end

;  FUNCTION:         test2.asm
;
   .sect ”test2”
   .def test2
   .ref test3
   .include ”global_reload.h”
   .include ”dmaregs.h”
   .global _test2_start, _test2_end
   .ref test4_start, test4_end
   .ref global_dst_addr
   .mmregs
   .label _test2_start
test2:
   rsbx  INTM                    ; Turn interrupt on..
; Initialize global reload registers for source and destination
; addresses for test4.asm
   stm   DMGSA, DMSA             ; Initialize to point to DMGSA
   stm   #SRC_4, DMSDI
   stm   #DST_4, DMSDI
   stm   #global_dst_addr, AR3
   st    #DST_4, *AR3            ; Copy to global_dst_addr placeholder
   stm   #(test4_end – test4_start – 1), DMSDI;
; Actual test code starts here...do something
   stm   #1400h, ar2
   stm   #0060h, ar1
   stm   #0CCCCh, *ar1+
   stm   #0AAAAh, *ar1+
```

```
   stm   #5555h, *ar1+
   stm   #0000h, *ar1
   stm   #3h, BRC
   rptb  error–1
   ld    *ar1, T
   st    T, *ar2
   ld    *ar2+, A
   sub   *ar1–, A
   bc    out, AEQ
   nop
error:
   ld    #0deadh, A
out:
   idle  1                      ; Idle until DMA interrupts CPU
   .label _test2_end

;  FUNCTION:         test3.asm
;
   .sect ”test3”
   .ref test4
   .def test3
   .include ”global_reload.h”
   .include ”dmaregs.h”
   .def test3_start, test3_end
   .ref _test1_start, _test1_end
   .ref global_dst_addr
   .mmregs
   .label test3_start
test3:
   rsbx  INTM                   ; Turn interrupt on..
; Initialize global reload registers for test1.asm
   stm   DMGSA, DMSA            ; Initialize to point to DMGSA
   stm   #SRC_1, DMSDI
   stm   #DST_1, DMSDI
   stm   #global_dst_addr, AR3
   st    #DST_1, *AR3           ; Copy to global_dst_addr placeholder
   stm   #(_test1_end – _test1_start – 1), DMSDI;
; Actual test code starts here...do something
   stm   #1400h, AR2
   stm   #0060h, AR1
```

```
   stm   1234h, *AR1
   idle  1                      ; Idle until DMA interrupts CPU...
   .label test3_end

;  FUNCTION:         test4.asm
;
   .sect ”test4”
   .ref test1
   .def test4
   .include ”global_reload.h”
   .include ”dmaregs.h”
   .def test4_start, test4_end
   .ref _test2_start, _test2_end
   .ref global_dst_addr
   .mmregs
   .label test4_start
test4:
   rsbx  INTM                   ; Turn interrupts on...
; Initialize global reload registers for source and destination addresses for
; test2.asm
   stm   DMGSA, DMSA            ; Initialize to point to DMGSA
   stm   #SRC_2, DMSDI
   stm   #DST_2, DMSDI
   stm   #global_dst_addr, AR3
   st    #DST_2, *AR3           ; Copy to global_dst_addr placeholder
   stm   #(_test2_end – _test2_start – 1), DMSDI
; Actual test code starts here...do something
   stm   #1400h, AR2
   STM   #0060h, AR1
   stm   #5555h, *AR1+
   stm   #0AAAAh, *AR1
   idle  1                      ; Idle until DMA interrupts CPU...
   .label test4_end
```

```
;  FUNCTION:         pgm_page_isr.asm
;
;  DESCRIPTION:      This is the interrupt service routine for the program
;                    paging scheme.  The procedure is as follows:
;
;                    a) Copies of the latest destination address (16 bits of
;                    extended destination and 16 bits of destination
;                    addresses) are used to determine the address to branch
;                    to
;
;                    b) Copy the global reload destination addresses (base
;                    and extended) to the memory   space previously occupied
;                    by the destination addresses.  These will be used to
;                    compute the effective address to branch to the next time
;                    an interrupt occurs...
;
;  ARGUMENTS:        None
;
;  AUTHOR:           Ramesh A. Iyer
;
;  CREATED:          July 12, 1999
;
;  VERSION:          1.0
;
   .def pgm_page_isr
   .include ”global_reload.h”
   .ref dst_addr, global_dst_addr
   .mmregs
   .sect ”dma_isr”
pgm_page_isr:
   stm   #dst_addr, AR2
   stm   #global_dst_addr, AR3
   ld    *AR2, B
   MVDD  *AR3+, *AR2+
   add   *AR2, 16, B            ; Create the final address that we branch to..
   FBACCD B
   MVDD  *AR3+, *AR2+           ; Global_dst_addr has been copied to dst_addr
   nop
```

```
/* FILENAME:         dma.cmd                                              */
/*                                                                        */
/* DESCRIPTION:      This is the linker command file                      */
/*                   for the program paging example                       */
/*                                                                        */
/* AUTHOR:           Ramesh A. Iyer                                       */
/*                                                                        */
/* CREATED:          June 17, 1999                                        */
/*                                                                        */
/* VERSION:          1.0                                                  */
/*                                                                        */
MEMORY
   {
      PAGE 0:
            P_DARAM: org = 800h        len = 017ffh
            PDARAM1: org = 010100h     len = 0100h
            PDARAM2: org = 010200h     len = 0100h
            PDARAM3: org = 010300h     len = 0100h
            PDARAM4: org = 010400h     len = 0100h
            PSARAM1: org = 18000h      len = 100h
            PSARAM2: org = 18100h      len = 100h
      VECS:          org = 2000h       len = 80h
      PAGE 1:
            D_SPRAM: org = 0060h       len = 0020h
            D_DARAM: org = 80h         len = 780h
            D_SARAM: org = 2000h,      len = 6000h
   }
SECTIONS
{
   .text    :>       P_DARAM             PAGE 0
   UNION:   run =    PSARAM1             PAGE 0
   {
      test1:   load = PDARAM1            PAGE 0
      test3:   load = PDARAM3            PAGE 0
   }
   UNION:   run =     PSARAM2            PAGE 0
   {
      test2:   load = PDARAM2            PAGE 0
      test4:   load = PDARAM4            PAGE 0
   }
   pgm_page :>       P_DARAM             PAGE 0
   dma_isr  :>       P_DARAM             PAGE 0
   .bss     :>       D_DARAM             PAGE 1
   .stack   :>       D_DARAM             PAGE 1
   vectors  :>       VECS                PAGE 0
}
```

```
;  FILENAME:         global_reload.h
;
;  DESCRIPTION:      This header file contains the source, extended source,
;                    destination and extended destination addresses.
;                    For simplicity, we shall assume that extended source
;                    and extended destination page numbers do not change
;                    throughout this example.  This is not necessarily true
;                    in a real system.
;
;                    It is also important to remember that the values
;                    assigned to each of the quantities in this file must
;                    match with the corresponding source and destination
;                    addresses specified in the command file (dma.cmd, in
;                    this case).  As an example, if the .cmd file specifies
;                    the module “test1.asm” is to be loaded in PDARAM1
;                    (origin = 0x1_0100h) but run from PSARAM1 (origin =
;                    0x1_8000h), the following values are assigned:
;
;                    EXT_DST     .set  0x1         ; Destination extended page 1
;                    DST_1       .set  8000h       ; Destination address
;                    EXT_SRC     .set  0x1         ; Source extended page 1
;                    SRC_1       .set  0100h       ; Source address
;
;  ARGUMENTS:        None
;
;  AUTHOR:           Ramesh A. Iyer
;
;  CREATED:          June 17, 1999
;
;  VERSION:          1.0
;
EXT_DST  .set  0x1                    ; Extended destination DMA address
DST_1    .set  8000h                  ; Destination DMA address for test1.asm
DST_2    .set  8100h                  ; Destination DMA address for test2.asm
DST_3    .set  8000h                  ; Destination DMA address for test3.asm
DST_4    .set  8100h                  ; Destination DMA address for test4.asm
EXT_SRC  .set  0x1                    ; Extended source DMA address
SRC_1    .set  0100h                  ; Source DMA address for test1.asm
SRC_2    .set  0200h                  ; Source DMA address for test2.asm
SRC_3    .set  0300h                  ; Source DMA address for test3.asm
SRC_4    .set  0400h                  ; Source DMA address for test4.asm
```

A.3 Data Transfer Using a Ping-Pong Scheme

```
/* FILENAME:         main.c                                          */
/*                                                                   */
/* DESCRIPTION:      This is the main program that passes            */
/*                   arguments to the underlying asm program         */
/*                   that handles dma initialization &               */
/*                   the ping–pong scheme.                           */
/*                                                                   */
/* ARGUMENTS:        Arguments passed to the asm program             */
/*                   follow C–to–asm conventions.                    */
/*                   All arguments are passed via the                */
/*                   structure called ”dma”                          */
/*                                                                   */
/* RETURN:           None                                            */
/*                                                                   */
/* AUTHOR:           Ramesh A. Iyer                                  */
/*                                                                   */
/* CREATED:          July 27, 1999                                   */
/*                                                                   */
/* VERSION:          1.0                                             */
/*                                                                   */
#include ”dma.h”
extern void pingpong();
void main()
{
/* Assign a dma channel                                              */
dma.dma_chan = dma_channel_0;
/* Assign a starting address                                         */
dma.src_address = 0x1000;
/* Assign an extended page to source address                         */
dma.ext_src_pgm_page = 0;
/* Assign an extended page to destination address                    */
dma.ext_dst_pgm_page = 0;
/* Assign a destination address                                      */
dma.dst_address = 0x2000;
/* Assign an element count                                           */
dma.element_count = 31;
/* Initialize DMSFC register                                         */
dma.DMSFC.frame_count = 0;
dma.DMSFC.double_word = 0;
dma.DMSFC.dmasyncevent = 0;
dma.DMSFC.rsvd = 0;
```

```
/* Initialize DMMCR register                                    */
dma.DMMCR.dma_dest_addr_space_select = 1;
dma.DMMCR.dma_dest_addr_index_mode = 1;
dma.DMMCR.dma_src_addr_space_sel = 1;
dma.DMMCR.dma_src_addr_index_mode = 1;
dma.DMMCR.dma_transfer_cntr_mode = 0;
dma.DMMCR.dma_int_gen_mode = 0;
dma.DMMCR.dma_int_gen_mask = 1;
dma.DMMCR.dma_autoinit = 1;
dma.DMMCR.rsvd = 0;
dma.DMMCR.reserved = 0;
/* Initialize DMPREC register                                   */
dma.DMPREC.dma_chan0_enable = 1;
dma.DMPREC.dma_chan1_enable = 0;
dma.DMPREC.dma_chan2_enable = 0;
dma.DMPREC.dma_chan3_enable = 0;
dma.DMPREC.dma_chan4_enable = 0;
dma.DMPREC.dma_chan5_enable = 0;
dma.DMPREC.int_mux_control = 2;
dma.DMPREC.dma_chan0_priority = 1;
dma.DMPREC.dma_chan1_priority = 0;
dma.DMPREC.dma_chan2_priority = 0;
dma.DMPREC.dma_chan3_priority = 0;
dma.DMPREC.dma_chan4_priority = 0;
dma.DMPREC.dma_chan5_priority = 0;
dma.DMPREC.free = 0;
dma.DMPREC.rsvd = 0;
/* Initialize global reload register values                     */
dma.global_src_addr_reload = 0x1000;
dma.global_dst_addr_reload = 0x4000;
dma.global_element_cnt_reload = 31;
dma.global_frame_cnt_reload = 0;
/* Call the application                                         */
pingpong(dma);
/* Do something                                                 */
for (;;);
/* End of main()                                                */
}
```

```
;  FUNCTION:         pingpong.asm
;
;  DESCRIPTION:      Handles the ping pong scheme...
;
;  AUTHOR:           Ramesh A. Iyer
;
;  CREATED:          June 27, 1999
;
;  VERSION:          1.0
;
   .global _pingpong
   .include ”dmaregs.h”
   .bss addr, 1, 1
   .bss global_addr, 1, 1
   .global addr
   .global global_addr
   .mmregs
   .sect ”circ_buffer”
_pingpong:
; Enable interrupts before you start doing anything
   stm   #0040h, IMR                  ; Turn on interrupt for dma channel 0
   rsbx  intm
; Initialize AR2 and AR3 which will be used to keep a track of the destination
; addresses
   stm   #addr, AR2
   stm   #global_addr, AR3
; The actual usage of the arguments passed via the structure ”dma” starts from
; here...
   STLM  A, AR0                       ; AR0 contains the starting address of the
                                      ; structure that was passed
   nop
   nop
   mac   *ar0+, offset, B
   STLM  B, DMSA                      ; Write effective offset i.e. starting
                                      ; address into DMSA
   LD    *AR0+, B                     ; Get source address
   stlm  B, DMSDI
   ld    *AR0+, B                     ; Get destination address
   stlm  B, DMSDI
   stl   B, *AR2                      ; Save a copy of the destination address
```

```
ld    *AR0+, B                  ; Get element count
stlm  B, DMSDI
ld    *AR0+, B                  ; Get DMSFC
stlm  B, DMSDI
ld    *AR0+, B                  ; Get DMMCR
stlm  B, DMSDI
stm   DMSRCP, DMSA              ; Initialize to point to DMSRCP
LD    *AR0+, B                  ; Get extended src. pgm address
STLM  B, DMSDI
LD    *AR0+, B                  ; Get extended dest. pgm address
STLM  B, DMSDI
stm   DMGSA, DMSA               ; Initialize to point to DMGSA
LD    *AR0+, B                  ; Get global src. address reload value
STLM  B, DMSDI
LD    *AR0+, B                  ; Get global dest.address reload value
STLM  B, DMSDI
stl   B, *AR3                   ; Save a copy of the global dest. address
                                ; too...
LD    *AR0+, B                  ; Get global element count reload value
STLM  B, DMSDI
LD    *AR0+, B                  ; Get global frame count reload value
STLM  B, DMSDI
LD    *AR0+, B                  ; Get DMPREC
stlm  B, DMPREC
ret
```

```
;  FUNCTION:         vector.asm
;
;  DESCRIPTION:      This is the interrupt vector table specific to the
;                    VC5410 device.  If you plan on using this table for
;                    other C54x/C54xx devices, please refer to the data sheet
;                    for information on the vector table structure of the
;                    specific device.
;
;  ARGUMENTS:        None
;
;  AUTHOR:           Ramesh A. Iyer
;
;  CREATED:          June 20, 1999
;
;  VERSION:          1.0
;
   .ref _c_int00
   .ref pingpong_isr
   .mmregs
   .sect ”vectors”
   b  _c_int00                    ; Reset vector
   nop
   nop
   .space 84*16
   b  pingpong_isr                ; DMA channel 0 multiplexed with McBSP2 Rx
   nop
   nop
```

```
;  FUNCTION:         pingpong_isr.asm
;
;  DESCRIPTION:      This is the interrupt service routine for the ping–pong
;                    buffer scheme, and accomplishes the following:
;
;                    a) Use the data in the input buffer for computation
;                    b) Switch the destination address and the global
;                     destination address.  Also write to the DMA global
;                     reload destination register before exiting.
;
;  ARGUMENTS:        None
;
;  AUTHOR:           Ramesh A. Iyer
;
;  CREATED:          July 27, 1999
;
;  VERSION:          1.0
;
   .def pingpong_isr
   .ref addr, global_addr
   .bss temp, 1, 1
   .include ”dmaregs.h”
   .mmregs
   .sect ”dma_isr”
pingpong_isr:
   mvdm  #addr, AR5
; The destination address needs to be swapped with the corresponding
; global reload values, so that:
;  a) The location ”addr” has the latest destination address the next
;     time the ISR is executed
;  b) The DMA global destination address can be reloaded
   stm   #addr, AR2
   stm   DMGDA, DMSA
   ld    *AR2, B
   stlm  B, DMSDN                              ; Reload global dest. address
   ; Swap destination and global reload destination
   stm   #global_addr, AR3
   stm   #temp, AR4
   mvdd  *AR2, *AR4                            ; copy dst_addr to temp
   mvdd  *AR3, *AR2                            ; copy global_dst_addr to dst_addr
```

```
mvdd  *AR4, *AR3                          ; copy temp to global_dst_addr
; Do something useful with the data from one of the input buffers
stm   #3000h, AR1
ld    #0, A
rpt   #31
add   *AR5+, A
stl   A, *AR1                             ; write result
rete
```

```
;  FUNCTION:         meminit.asm
;
;  DESCRIPTION:      Contains only an initialization table that is used to
;                    test the code...
;
;  ARGUMENTS:        None
;
;  AUTHOR:           Ramesh A. Iyer
;
;  CREATED:          June 27, 1999
;
;  VERSION:          1.0
;
   .def _meminit
_meminit:
   .data
   .word    0000h, 0001h, 0002h, 0003h
   .word    0004h, 0005h, 0006h, 0007h
   .word    0008h, 0009h, 000ah, 000bh
   .word    000ch, 000dh, 000eh, 000fh
   .word    0010h, 0011h, 0012h, 0013h
   .word    0014h, 0015h, 0016h, 0017h
   .word    0018h, 0019h, 001ah, 001bh
   .word    001ch, 001dh, 001eh, 001fh
```

```
/* FILENAME:         dma.cmd                                          */
/*                                                                    */
/* DESCRIPTION:      This is the linker command file                  */
/*                   for the ping pong example                        */
/*                                                                    */
/* AUTHOR:           Ramesh A. Iyer                                   */
/*                                                                    */
/* CREATED:          July 27, 1999                                    */
/*                                                                    */
/* VERSION:          1.0                                              */
/*                                                                    */
MEMORY
   {
      PAGE 0:
            P_DARAM: org = 800h        len = 017ffh
            PSARAM1: org = 2000h       len = 1000h
            PSARAM2: org = 3000h       len = 1000h
            VECS:    org = 0ff80h      len = 80h
      PAGE 1:
            D_SPRAM: org = 0060h       len = 0020h
            D_DARAM1:org = 80h         len = 780h
            D_DARAM2:org = 800h        len = 1800h
            D_SARAM: org = 2000h       len = 6000h
   }
SECTIONS
   {
   .text    :>       P_DARAM             PAGE 0
   pingpong :>       P_DARAM             PAGE 0
   dma_isr  :>       P_DARAM             PAGE 0
   .bss     :>       D_DARAM1            PAGE 1
   .stack   :>       D_DARAM1            PAGE 1
   vectors  :>       VECS                PAGE 0
   .data    :        load = 0x1000       PAGE 1
   }
```

Appendix B Header Files

This appendix contains a listing of the header files used in the application examples and a sample vector table file.

```
/* FILENAME:         dma.h                                          */
/*                                                                  */
/* DESCRIPTION:      Contains a definition of the                   */
/*                   structures used to access the                  */
/*                   DMA registers in the datasort example          */
/*                                                                  */
/* ARGUMENTS:        None                                           */
/*                                                                  */
/* AUTHOR:           Ramesh A. Iyer                                 */
/*                                                                  */
/* CREATED:          June 17, 1999                                  */
/*                                                                  */
/* VERSION:          1.0                                            */
/*                                                                  */
/* Structure definition for DMSFC                                   */
struct dmsfc_reg
{
   unsigned int   dmasyncevent                :  4;
   unsigned int   double_word                 :  1;
   unsigned int   rsvd                        :  3;
   unsigned int   frame_count                 :  8;
};
/* Structure definition for DMMCR                                   */
struct dmmcr_reg
{
   unsigned int   dma_autoinit                :  1;
   unsigned int   dma_int_gen_mask            :  1;
   unsigned int   dma_int_gen_mode            :  1;
   unsigned int   dma_transfer_cntr_mode      :  1;
   unsigned int   rsvd                        :  1;
   unsigned int   dma_src_addr_index_mode     :  3;
   unsigned int   dma_src_addr_space_sel      :  2;
   unsigned int   reserved                    :  1;
   unsigned int   dma_dest_addr_index_mode    :  3;
   unsigned int   dma_dest_addr_space_select  :  2;
};
```

```
/* Structure definition for DMPREC                                    */
struct dmprec_reg
{
   unsigned int   free                      :  1;
   unsigned int   rsvd                      :  1;
   unsigned int   dma_chan5_priority        :  1;
   unsigned int   dma_chan4_priority        :  1;
   unsigned int   dma_chan3_priority        :  1;
   unsigned int   dma_chan2_priority        :  1;
   unsigned int   dma_chan1_priority        :  1;
   unsigned int   dma_chan0_priority        :  1;
   unsigned int   int_mux_control           :  2;
   unsigned int   dma_chan5_enable          :  1;
   unsigned int   dma_chan4_enable          :  1;
   unsigned int   dma_chan3_enable          :  1;
   unsigned int   dma_chan2_enable          :  1;
   unsigned int   dma_chan1_enable          :  1;
   unsigned int   dma_chan0_enable          :  1;
};
/* This structure is used by the calling program to pass arguments to  */
/* the child assembly program                                          */
struct dma_param
{
   unsigned int         dma_chan;
   unsigned int         src_address;
   unsigned int         dst_address;
   unsigned int         element_count;
   struct dmsfc_reg     DMSFC;
   struct dmmcr_reg     DMMCR;
   struct dmprec_reg    DMPREC;
}dma;
/* DMA channels...                                                    */
#define  dma_channel_0       0
#define  dma_channel_1       1
#define  dma_channel_2       2
#define  dma_channel_3       3
#define  dma_channel_4       4
#define  dma_channel_5       5
```

```
/* FILENAME:         dma.h                                                */
/*                                                                        */
/* DESCRIPTION:      Contains a definition of the                         */
/*                   structures used to access the                        */
/*                   DMA registers   in the program paging and            */
/*                   ping–pong examples.                                  */
/*                                                                        */
/* ARGUMENTS:        None                                                 */
/*                                                                        */
/* AUTHOR:           Ramesh A. Iyer                                       */
/*                                                                        */
/* CREATED:          June 17, 1999                                        */
/*                                                                        */
/* VERSION:          1.0                                                  */
/*                                                                        */
/* Structure definition for DMSFC                                         */
struct dmsfc_reg
{
   unsigned int   dmasyncevent                :  4;
   unsigned int   double_word                 :  1;
   unsigned int   rsvd                        :  3;
   unsigned int   frame_count                 :  8;
};
/* Structure definition for DMMCR                                         */
struct dmmcr_reg
{
   unsigned int   dma_autoinit                :  1;
   unsigned int   dma_int_gen_mask            :  1;
   unsigned int   dma_int_gen_mode            :  1;
   unsigned int   dma_transfer_cntr_mode      :  1;
   unsigned int   rsvd                        :  1;
   unsigned int   dma_src_addr_index_mode     :  3;
   unsigned int   dma_src_addr_space_sel      :  2;
   unsigned int   reserved                    :  1;
   unsigned int   dma_dest_addr_index_mode    :  3;
   unsigned int   dma_dest_addr_space_select  :  2;
};
```

```
/* Structure definition for DMPREC                                          */
struct dmprec_reg
{
   unsigned int   free                        :  1;
   unsigned int   rsvd                        :  1;
   unsigned int   dma_chan5_priority          :  1;
   unsigned int   dma_chan4_priority          :  1;
   unsigned int   dma_chan3_priority          :  1;
   unsigned int   dma_chan2_priority          :  1;
   unsigned int   dma_chan1_priority          :  1;
   unsigned int   dma_chan0_priority          :  1;
   unsigned int   int_mux_control             :  2;
   unsigned int   dma_chan5_enable            :  1;
   unsigned int   dma_chan4_enable            :  1;
   unsigned int   dma_chan3_enable            :  1;
   unsigned int   dma_chan2_enable            :  1;
   unsigned int   dma_chan1_enable            :  1;
   unsigned int   dma_chan0_enable            :  1;
};
/* This structure is used by the calling program to pass arguments to       */
/* the child assembly program                                               */
struct dma_param
{
   unsigned int         dma_chan;
   unsigned int         src_address;
   unsigned int         dst_address;
   unsigned int         element_count;
   struct dmsfc_reg     DMSFC;
   struct dmmcr_reg     DMMCR;
   unsigned int         ext_src_pgm_page;
   unsigned int         ext_dst_pgm_page;
   unsigned int         global_src_addr_reload;
   unsigned int         global_dst_addr_reload;
   unsigned int         global_element_cnt_reload;
   unsigned int         global_frame_cnt_reload;
   struct dmprec_reg    DMPREC;
}dma;
/* DMA channels...                                                          */
#define  dma_channel_0       0
#define  dma_channel_1       1
#define  dma_channel_2       2
#define  dma_channel_3       3
#define  dma_channel_4       4
#define  dma_channel_5       5
```

```
;  FILENAME:         dmaregs.h
;
;  DESCRIPTION:      Contains DMA register definitions
;                    for C54xx and other global values used
;                    in the examples.
;
;  ARGUMENTS:        None
;
;  AUTHOR:           Ramesh A. Iyer
;
;  CREATED:          June 17, 1999
;
;  VERSION:          1.0
;
DMPREC   .set  0054h ;Channel Priority and Enable Control Register
DMSA     .set  0055h ;Sub–bank Address Register
DMSDI    .set  0056h ;Sub–bank Data Register with autoincrement
DMSDN    .set  0057h ;Sub–bank Data Register without modification
DMSRC0   .set  00h   ;Channel 0 Source Address Register
DMDST0   .set  01h   ;Channel 0 Destination Address Register
DMCTR0   .set  02h   ;Channel 0 Element Count Register
DMSFC0   .set  03h   ;Channel 0 Sync Select and Frame Count Register
DMMCR0   .set  04h   ;Channel 0 Transfer Mode Control Register
DMSRC1   .set  05h   ;Channel 1 Source Address Register
DMDST1   .set  06h   ;Channel 1 Destination Address Register
DMCTR1   .set  07h   ;Channel 1 Element Count Register
DMSFC1   .set  08h   ;Channel 1 Sync Select and Frame Count Register
DMMCR1   .set  09h   ;Channel 1 Transfer Mode Control Register
DMSRC2   .set  0Ah   ;Channel 2 Source Address Register
DMDST2   .set  0Bh   ;Channel 2 Destination Address Register
DMCTR2   .set  0Ch   ;Channel 2 Element Count Register
DMSFC2   .set  0Dh   ;Channel 2 Sync Select and Frame Count Register
DMMCR2   .set  0Eh   ;Channel 2 Transfer Mode Control Register
DMSRC3   .set  0Fh   ;Channel 3 Source Address Register
DMDST3   .set  10h   ;Channel 3 Destination Address Register
DMCTR3   .set  11h   ;Channel 3 Element Count Register
DMSFC3   .set  12h   ;Channel 3 Sync Select and Frame Count Register
DMMCR3   .set  13h   ;Channel 3 Transfer Mode Control Register
DMSRC4   .set  14h   ;Channel 4 Source Address Register
DMDST4   .set  15h   ;Channel 4 Destination Address Register
```

```
DMCTR4   .set  16h   ;Channel 4 Element Count Register
DMSFC4   .set  17h   ;Channel 4 Sync Select and Frame Count Register
DMMCR4   .set  18h   ;Channel 4 Transfer Mode Control Register
DMSRC5   .set  19h   ;Channel 5 Source Address Register
DMDST5   .set  1Ah   ;Channel 5 Destination Address Register
DMCTR5   .set  1Bh   ;Channel 5 Element Count Register
DMSFC5   .set  1Ch   ;Channel 5 Sync Select and Frame Count Register
DMMCR5   .set  1Dh   ;Channel 5 Transfer Mode Control Register
DMSRCP   .set  1Eh   ;Source Program Page Address
DMDSTP   .set  1Fh   ;Destination Program Page Address
DMIDX0   .set  20h   ;Element Address Index Register 0
DMIDX1   .set  21h   ;Element Address Index Register 1
DMFRI0   .set  22h   ;Frame Address Index Register 0
DMFRI1   .set  23h   ;Frame Address Index Register 1
DMGSA    .set  24h   ;Global Source Address Reload Register
DMGDA    .set  25h   ;Global Destination Address Reload Register
DMGCR    .set  26h   ;Global Element Count Reload Register
DMGFR    .set  27h   ;Global Frame Count Reload Register
; Also contains other global initialized variables for some of the
; examples
offset   .set  5     ; Used in all the examples to determine which
                     ; DMA channel registers need to be programmed
frame    .set  3     ; Used in the datasort example as an offset to
                     ; the Stack Pointer (SP) to save local variables.
```

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third–party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Mailing Address:

Texas Instruments
Post Office Box 655303
Dallas, Texas 75265