Technical Notes on using Analog Devices' DSP components and development tools
Phone: (800) ANALOG-D, FAX: (781) 461-3010, EMAIL: dsp.support@analog.com, FTP: ftp.analog.com, WEB: www.analog.com/dsp

SHARC SHORT WORD DMA

Last Modified: 8/1/00

Introduction

The DMA Controller of the Analog Devices SHARC® family processors does not automatically dma 16 bit data from an external memory to the SHARC's internal memory, however 16 bit data transfers can still easily be accomplished. The following Engineer to Engineer Note describes the method used to move 16 bit data into the SHARC's internal memory and includes a sample code.

The external port and serial port DMA channels have a packing capability allowing 16 bit data to be packed to 32 bits. The 32 bit packed words can be transferred to internal memory using standard dma. Once the data is safely in internal memory it can be accessed as short word memory. Figure 1 illustrates how short words are organized in normal word space. The least significant short word address is generated by arithmetically shifting the normal word address to the left 1 bit. Arithmetically shifting the normal word address to the left 1 bit and then setting bit 0 to 1 generates the most significant short word address. For example, if the normal word address for a memory location were 0xC000 the short word address would be 0x18000 for the least significant word and 0x18001 for the most significant word.

All of this information can be found in the *ADSP-2106x SHARC User's Manual* or the *ADSP-21065L SHARC User's Manual*. For better understanding please see the chapters on DMA and Memory in your User's Manual.

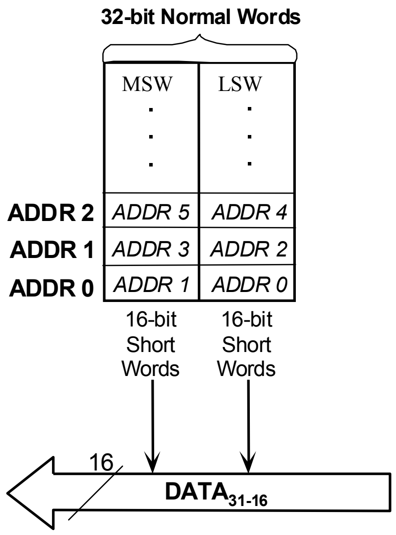

Figure 1. Short word addresses

Listing1: dma.asm

```
/*
DMA.ASM         ADSP-21065L EPORT 16/32 bit packing
GGL, Analog Devices, Inc.      7/19/00

This code is a simple example of getting 16 bit floating point data from SDRAM packing it to 32 bits
and using dma to get it into internal memory.  Once in internal memory, the data is accessed as short
word and operated on and sent back to internal memory.

*/
#define N 4
#include "def21065l.h"

.section/DM     seg_dmda;
.VAR dest[N];
/* This is your destination buffer in internal memory which resides in normal word space */


.section/DM seg_sdram;
.VAR source[2*N]=0x1111, 0x2222, 0x3333, 0x4444, 0xAAAA, 0xBBBB, 0xCCCC, 0xDDDD;
/* This is the sample 16 bit data from an external sdram */



.section/pm     seg_rth;
/* This is the interrupt vector table */
Reserved_1:   rti; nop; nop; nop;
Chip_Reset:   idle; jump start; nop; nop;
Reserved_2:   rti; nop; nop; nop;
stack_ov:     rti; nop; nop; nop;
timerhi:      rti; nop; nop; nop;
Vector:       rti; nop; nop; nop;
IRQ2:         rti; nop; nop; nop;
IRQ1:  rti; nop; nop; nop;
IRQ0:  rti; nop; nop; nop;
Reserved_3:   rti; nop; nop; nop;
sport0r:      rti; nop; nop; nop;
sport1r:      rti; nop; nop; nop;
sport0t:      rti; nop; nop; nop;
sport1t:      rti; nop; nop; nop;
Reserved_4:   rti; nop; nop; nop;
Reserved_5:   rti; nop; nop; nop;
EP0:          nop; nop; rti; nop;
```

```
EP1:            rti; nop; nop; nop;


.section/pm     seg_pmco;
/*______________________start of main routine___________________________*/
start:
                ustat1=937;                              /*refresh rate*/
                dm(SDRDIV)=ustat1;
                ustat1=dm(IOCTL);                 /*mask in SDRAM settings*/
                bit set ustat1 SDPSS|SDBN2|SDBS3|SDTRP3|SDTRAS5|SDCL2|SDPGS256|SDDCK1;
                dm(IOCTL)=ustat1;                 /*initialize sdram*/


                /* enable interrupts here */
                bit set MODE1 IRPTEN;             /* enable global interrupts here */
                bit set IMASK EP0I;  /* enable eport interrupts here */


/* must set up II IM and C register in memory bufffer ; then enable the channel with DEN in DMAC */
/* dma channel 8 is external port buffer 0 */

        r0=0; dm(DMAC0)=r0;                              /*clear DMA register*/
        r0 = 1; dm(IMEP0) = r0;        /* set DMA internal memory DMA modifier to 1*/
        dm(EMEP0)=r0;                                    /* set DMA external memory DMA
modifier to 1*/
        r0 = N; dm(CEP0) = r0;                    /*set internal  count to N */
        r0=2*N; dm(ECEP0)=r0;                     /*set external DMA count to 2N */
        r0=dest; dm(IIEP0) = r0;           /* Write internal index pointer for dest */
        r0=source; dm(EIEP0)=r0;           /*Write External DMA buffer Index pointer for source */
        r0=0x0241; dm(DMAC0)=r0;                  /* enable DMA channel, master mode, 16/32bit
packing */

wait1: idle;
        /* when this dma completes you will have the 16 bit words that were stored in SDRAM
        packed to 32 bit words in external memory */

        /* the ISR only has an RTI in it, so after the dma is complete
        the program flow will continue in the next instruction */



/* find the first address of the data buffer and convert it to short word */
r0=dest;
```

```
r1=ashift r0 by 1; /* arithmetically shifting a normal word address to the left produces the short word
address of the LSW */

/* initialize dags to the short word buffer */
b0=r1;
m0=1;
l0=2*N;

/* operate on the short word data and return to short word space */

r0=dm(i0,m0); f0=funpack r0; /* read the short word floating point data to a register and then unpack it
*/
r1=dm(i0,m0); f1=funpack r1;
f2=f0*f1; r2=fpack f2;          /* multiply f0 and f1 and then pack it back into a 16 bit floating point word
*/
dm(i0,m0)=r2;/* send the 16 bit result out to memory */

r3=dm(i0,m0);/* read 16 bit fixed point value */
r4=dm(i0,m0);

r5=r3+r4;                 /* fixed point add */
dm(i0,m0)=r5;/* send 16 bit result out to memory */

wait2:   jump wait2;
```

Listing 2: ADSP21065L.ldf

```
ARCHITECTURE(ADSP-21065L)

//
// ADSP-21065L Memory Map:
//   ------------------------------------------------
//   Internal memory  0x0000 0000 to 0x0007 ffff
//   ------------------------------------------------
//                  0x0000 0000 to 0x0000 00ff  IOP Regs
//                  0x0000 0100 to 0x0000 01ff  IOP Regs of Processor ID 001
//                  0x0000 0200 to 0x0000 02ff  IOP Regs of Processor ID 002
//                  0x0000 0300 to 0x0000 07ff  Reserved
//          Block 0  0x0000 8000 to 0x0000 9fff  Normal Word (32/48) Addresses
//                (0x0000 8000 to 0x0000 97ff) 48-bit words
//                (0x0000 8000 to 0x0000 9fff) 32-bit words
//                                           0x0000 A000 to 0x0000 Bfff  Reserved
//          Block 1  0x0000 C000 to 0x0000 Dfff  Normal Word (32/48) Addresses
//                (0x0000 C000 to 0x0000 Cfff) 48-bit words
//                (0x0000 C000 to 0x0000 Dfff) 32-bit words
//                                          0x0000 E000 to 0x0000 ffff  Reserved
//          Block 0  0x0001 0000 to 0x0001 3fff  Short Word (16) Addresses
//                                          0x0001 4000 to 0x0001 7fff  Reserved
//          Block 1  0x0001 8000 to 0x0001 Bfff  Short Word (16) Addresses
//                                          0x0001 C000 to 0x0001 ffff  Reserved
//   ----------------------------------------------
//   External memory  0x0002 0000 to 0x03ff ffff
//   ----------------------------------------------
//   External Bank 0  0x0002 0000 to 0x00ff ffff
//   External Bank 1  0x0100 0000 to 0x01ff ffff
//   External Bank 2  0x0200 0000 to 0x02ff ffff
//   External Bank 3  0x0300 0000 to 0x03ff ffff
//
// This architecture file allocates:
//        Internal 256 words of run-time header in memory block 0
//                16 words of initialization code in memory block 0
//             3.73K words of C code space in memory block 0
//               2K words of C PM data space in memory block 0
//               4K words of C DM data space in memory block 1
//             1.25K words of C heap space in memory block 1
//             2.75K words of C stack space in memory block 1

SEARCH_DIR( $ADI_DSP\21k\lib )

// The lib060.dlb must come before libc.dlb because libc.dlb has some 21020
// specific code and data
$LIBRARIES = lib060.dlb, libc.dlb, libio32.dlb;

// Libraries from the command line are included in COMMAND_LINE_OBJECTS.
$OBJECTS =  $COMMAND_LINE_OBJECTS;

MEMORY
{
        seg_rth  { TYPE(PM RAM) START(0x00008000) END(0x000080ff) WIDTH(48) }
        seg_init { TYPE(PM RAM) START(0x00008100) END(0x0000810f) WIDTH(48) }
        seg_pmco { TYPE(PM RAM) START(0x00008110) END(0x00008fff) WIDTH(48) }
        seg_pmda { TYPE(PM RAM) START(0x00009800) END(0x00009fff) WIDTH(32) }

        seg_dmda { TYPE(DM RAM) START(0x0000C000) END(0x0000Cfff) WIDTH(32) }
        seg_heap { TYPE(DM RAM) START(0x0000D000) END(0x0000D4ff) WIDTH(32) }
        seg_stak { TYPE(DM RAM) START(0x0000D500) END(0x0000Dfff) WIDTH(32) }
        seg_sdram { TYPE(DM RAM) START(0x3000000) END(0x030ffeff) WIDTH(32) }
}



PROCESSOR p0
{
   LINK_AGAINST( $COMMAND_LINE_LINK_AGAINST)
   OUTPUT( $COMMAND_LINE_OUTPUT_FILE )

   SECTIONS
   {

                  // .text output section
                  seg_rth
                  {
                          INPUT_SECTIONS( $OBJECTS(seg_rth) $LIBRARIES(seg_rth))
                  } >seg_rth

                  seg_init
                  {
                          INPUT_SECTIONS( $OBJECTS(seg_init) $LIBRARIES(seg_init))
                  } >seg_init

                  seg_pmco
                  {
                          INPUT_SECTIONS( $OBJECTS(seg_pmco) $LIBRARIES(seg_pmco))
                  } >seg_pmco

                  seg_pmda
                  {
                          INPUT_SECTIONS( $OBJECTS(seg_pmda) $LIBRARIES(seg_pmda))
                  } >seg_pmda

                  seg_dmda
                  {
                          INPUT_SECTIONS( $OBJECTS(seg_dmda) $LIBRARIES(seg_dmda))
                  } > seg_dmda

                  seg_sdram /*SHT_NOBITS if you do not want to initialize SRAM area in executable*/
                  {
                          INPUT_SECTIONS( $OBJECTS(seg_sdram))
                  } > seg_sdram
```

```
                stackseg
                {

                        // allocate a stack for the application
                        ldf_stack_space = .;
                        ldf_stack_length = MEMORY_SIZEOF(seg_stak);
                } > seg_stak

                heap
                {
                        // allocate a heap for the application
                        ldf_heap_space = .;
                        ldf_heap_length = MEMORY_SIZEOF(seg_heap);
                        ldf_heap_end = ldf_heap_space + ldf_heap_length - 1;
                } > seg_heap

        }
}
```