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

ADSP-2192 INTERPROCESSOR COMMUNCATION

Modified 11/8/2000

Overview

Each DSP core on the ADSP-2192 has the ability to communicate with the other DSP core directly with DSP-to-DSP semaphores and interrupts or indirectly with shared memory. The purpose of this application note is to describe the functionality of both forms.

ADSP-2192 Shared Memory

The ADSP-2192 is a 16-bit DSP with dual 219x cores. The DSP memory map is shown in Figure 1.

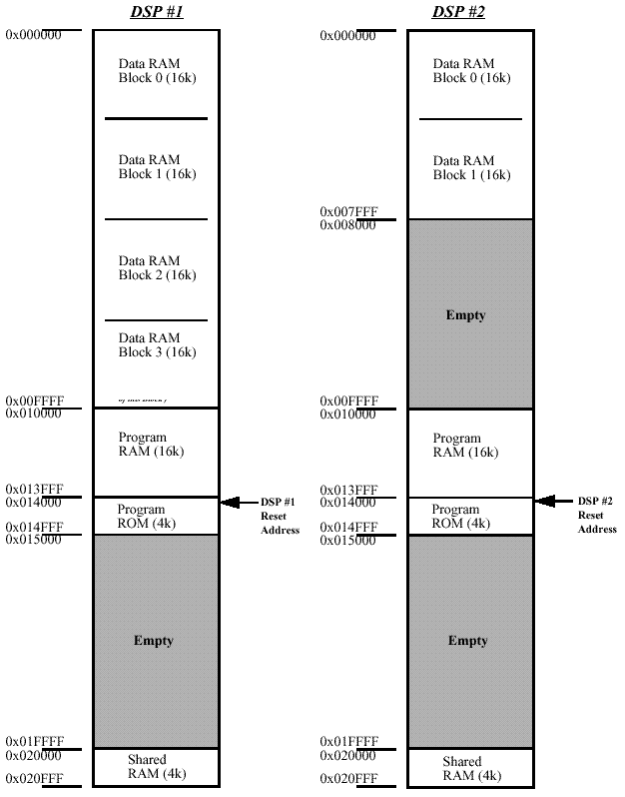

Figure 1. ADSP-2192 Memory Map

There are 140K words of internal memory on the ADSP-2192. Within this space, the P0 DSP core has 80K words of SRAM and 4K words of ROM. The P1 DSP core has 48K words of SRAM and 4K words of ROM. The P0 and P1 DSP cores also share a 4K x 16-bit memory space.

The ADSP-2192 has a modified Harvard architecture, which allows each DSP core to access a data word and an instruction simultaneously. This is accomplished by using two Data Address Generators (DAG). As shown in Figure 2, each DSP core has a DAG1 and a DAG2.

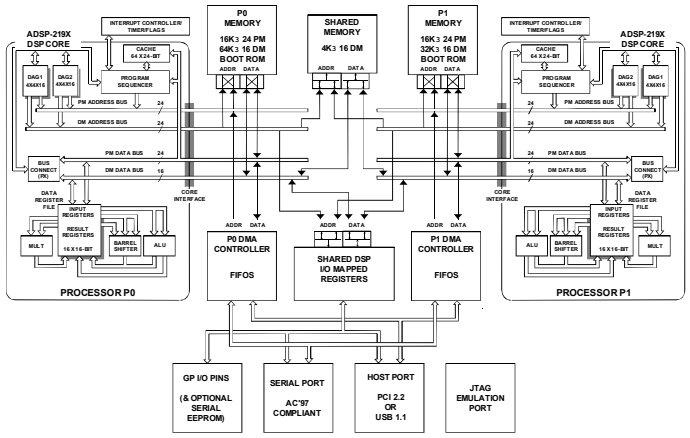

Figure 2. ADSP-2192 Dual-Core DSP Block Diagram

Each DAG is associated with a particular data bus and 16K word memory page, which allows it to access its own DM/PM memory in addition to the Shared Memory. From settings at reset, DAG1 supplies addresses over the DM bus for memory page 0 and DAG2 supplies addresses over the PM bus for memory page 1. These memory page selections may be changed using DMPGx registers. For more information on address generation, please refer to the Hardware Reference Manual.

As shown in Figure 2, unlike core memory, shared memory is internally single ported, so stalls will happen when being accessed by both cores. Each DSP core using the Data Memory (DM) bus can access shared memory. Shared memory is mapped to address 0x020000 and is thus on page 2. To access shared memory, DAG1 would be initialized as follows:

```
DMPG1 = 0x2;
```

Shared memory could then be accessed over the DM bus using either Direct or Indirect addressing. The following example would write 0xFF to location 0x020010, which resides in shared memory.

```
AX0 = 0xFF;
DMPG1 = 0x2;
DM(0x0010) = AX0;
```

Unlike the internal memory of each DSP core, every access to shared memory incurs at least one cycle of stall (to perform synchronization), therefore a minimum latency of 2 cycles is incurred.

When accessing shared memory, a DSP core 'locks out' the other DSP core for several cycles. A DSP core can completely lock out the other core from shared memory by performing back-to-back or nearly back-to-back accesses to shared memory. Once a particular DSP core "owns" the shared memory, the shared memory interface is relinquished only after 2 cycles of inactivity on the interface. In the case where both cores try to access shared memory in the same cycle, the ADSP-2192 has a bus arbitration scheme to handle the conflict. Arbitration is fixed at the following priority: DSP0, DSP1.

Performing burst accesses is an effective way to get good bandwidth from shared memory. Each access after the first will take 2 cycles, which is the maximum throughput.

Inter-processor Semaphores

The ADSP-2192 has a two internal DSP-to-DSP core flags and an internal DSP-to-DSP core interrupt controlled in the core semaphores register. This is shown below in Figure 3.

Flag Bit	Direction	Function	DSP Core Flag In
0	Output	DSP-DSP Semaphore 0	
1	Output	DSP-DSP Semaphore 1	
2	Output	DSP-DSP Interrupt	
3		*Reserved*	
4		*Reserved*	
5		*Reserved*	
6		*Reserved*	
7	Output	Register Bus Lock	
8	Input	DSP-DSP Semaphore 0	0
9	Input	DSP-DSP Semaphore 1	1
10	Input	DSP-DSP Interrupt	2
11	Input	*Reserved*	
12	Input	AC'97 Register – PDC Bus Access Status	4
13	Input	PDC Interface Busy Status (write from DSP pending)	5
14	Input	*Reserved*	
15	Input	Register Bus Lock Status	7

Figure 3. DSP-to-DSP Semaphore Register Table

The following example will illustrate the case where DSP core P0 asserts DSP-DSP semaphore 0 of DSP core P1.

```
/* assert DSP-to-DSP flag 0 */
ax0 = 0x0001;
reg(0x34) = ax0;
```

When a 1 is written to bit 0 of the core semaphores register of DSP core P0, the resulting core semaphore of DSP core P1, bit 8, is set. The DSP-to-DSP semaphore 1 function the same way.

The DSP-to-DSP interrupt allows DSP core P0 to interrupt DSP core P1 if the DSP-DSP interrupt is unmasked in the IMASK register of DSP core P1. The DSP-DSP interrupt routine can either be

nested with higher priority interrupts taking precedence or processed sequentially.

The following example will illustrate the initialization of DSP-DSP Interrupt.

```
/* Initialize DSP-to-DSP Interrupt */
      AY0=IMASK;
      AY1=0x0100;
      AR = AY0 or AY1;

/* Unmask DSP-DSP Interrupts */
      IMASK=AR;

/* Enable global interrupts */
      ENA INT;
```

The content of IMASK is *OR'd* with 0x0100 and written back to IMASK. This unmasks DSP-DSP interrupt for that particular DSP core. The last instruction globally enables interrupt servicing. If the above DSP-DSP interrupt initialization is executed on DSP core P1, DSP core P0 would initiate the DSP-DSP interrupt by asserting bit 2 of its core semaphores register.

Inter-processor Communication Example

The following example will illustrate the use of shared memory and inter-processor communication. This is performed using two FIR routines, one executed on each core. The FIR on DSP core P0 uses a set of coefficients to implement a low pass filter (Fir_core_p0.asm) while the filter implemented on DSP core P1 uses a separate set of coefficients to implement a high pass filter (Fir_core_p1.asm). The example filters a buffer of input samples using the FIR filters in both cores, implementing a band pass filter.

1) DSP core P0 filters a block of samples located in a buffer in internal memory and stores the filtered results in shared memory. At this point, DSP core P0 flags DSP core P1 using the DSP-DSP semaphore 1.

```
/* DSP core P0 */

      ...
      ax0=0x0002;
      reg(0x34)=ax0;
      ...
```

When bit 9 of DSP core P1 core semaphore register is set, it indicates there is valid data in shared memory.

2) At this point, DSP core P1 can now begin processing that data. DSP core P1 reads the data from shared memory and filters the data again using a high pass FIR routine. These filtered samples are stored in the internal memory of DSP core P1.

3) When DSP core P1 has completed filtering the bin of data, DSP core P1 asserts bit 2 of the semaphore register, which initiates a DSP-to-DSP interrupt in DSP core P0.

```
/* DSP core P1 */

      ...
      ax0=0x0004;
      reg(0x34)=ax0;
      ...
```

When core P0 latches and services the DSP-DSP interrupt, it indicates that P1 is requesting another bin of data to be place in shared memory.

4) DSP core P0 then acknowledges the request by also generating a DSP-DSP interrupt in P1 and begins filtering another bin of input data storing the result in shared memory.

```
/* DSP core P0 */

/* DSP-DSP interrupt vector code */
   .section/pm IVdspdspint;
      ax0=0x0004;
      RTI (db);
      reg(0x34)=ax0;
      ax0=0x1000;
```

5) Steps 1 to 4 are repeated.

Appendix A

Listed below are the source codes used to illustrate the inter-processor communication and shared memory usage of a dual core band pass FIR for the ADSP-2192. *(Please note that these included code modules were built using the v7.0 VisualDSP development tools for the 219x processor family.)*

Fir_core_p0.asm

```
/*****************************************************************************
  CONSTANT & MACRO DEFINITIONS
*****************************************************************************/
#define         Num_Samp                512                             /* Number of Input samples */
#define         Taps                    31                              /* Number of filter taps */
#define         Shift                   2                               /* Number of shifts */
#define         Samps_per_Bin           40                              /* Number of Input samples in each bin */
#define         Num_Bins                (Num_Samp/Samps_per_Bin)        /* Number of bins */

/*****************************************************************************
   EXTERNAL DECLARATIONS
*****************************************************************************/
.EXTERN SHARED_MEM;


/* DM data */
.section/data seg_dmda;
.VAR    IN[Num_Samp] = "input_dec.dat";         /* Input buffer */
.VAR    Delay_Line[Taps];                       /* Delay line */


/* PM data */
.section/pm seg_pmda;
.VAR    COEFF[Taps] = "coeff_l.dat";            /* Low pass filter coefficients */

/* PM interrupt vector code */
.section/pm seg_rth;
      JUMP start; NOP; NOP; NOP;

/* PM DSP-DSP interrupt vector code */
.section/pm IVdspdspint;
      ax0=0x0004;
      RTI (db);
      reg(0x34)=ax0;
      ax0=0x1000;                               /* Assert DSP-DSP interrupt to core 1 */

/* Program memory code */
.section/pm seg_pmco;
start:
   I0=Delay_Line;                               /* Initialize delay line pointer */
   M1 = 1;
   I1 = IN;                                     /* Initialize Input pointer */
   I2 = SHARED_MEM;                             /* Initialize Shared Memory pointer */
   L1=0;                                        /* Initialize for modulo addressing */
   M3=-1;
   M4=1;
   SE = Shift;
   DMPG2=page(COEFF);
   L0=LENGTH (Delay_Line);                       /* Initialize delay line circular buffer */
   ax0 = Delay_Line;
   REG(b0) = ax0;                               /* Initialize pointer to delay line */
   L2=LENGTH (SHARED_MEM);                      /* Initialize shared memory circular buffer */
   ax0 = SHARED_MEM;
   REG(B2) = ax0;                               /* Initialize pointer to shared memory */
   L4=length(COEFF);                            /* Initialize coeficient circular buffer */
   ax0 = COEFF;
   reg(b4) = ax0;                               /* Initialize pointer to coeficient */
/**  Zero the Delay Line **/
   CNTR=Taps;                                   /* 'Taps' location delay line */
   DO zero UNTIL CE;
zero:  dm(I0,M1)=L0;                            /* Delay_line = 0 */

   AY0=IMASK;
   AY1=0x0100;
   AR = AY0 or AY1;                             /* Enable DSP-DSP Interrupts */
   IMASK=AR;
   ENA INT;                                     /* Enable global interrupts */
   CNTR = Num_Bins;                             /* Num_Bins to process Num_Samp  */
   do per_bin until CE;
        call fir_p0 (db);                       /* Call FIR routine */
        I4=COEFF;                               /* Load Coeff buffer pointer */
        nop;
        ax0=0x0002;                             /* Assert 'Bin Ready' flag */
        reg(0x34)=ax0;
        ay0=0x1000;
bin_rqst:
```

```
        ar = ax0 AND ay0;
        if eq jump bin_rqst;                    /* Wait for DSP-DSP interrupt 'Bin request' */
        ax0=0x0000;                             /* Deassert DSP-DSP interrupt 'Bin request acknowledge' */
per_bin:
        reg(0x34)=ax0;

looping:
        nop;
        JUMP looping;                           /* Loop upon itself */

/************************************************************************
  FIR SUBROUTINE
************************************************************************/
fir_p0:
   MR = 0, MX0 = DM(I1,M1), MY0 = PM(I4,m4);    /* Read Input sample and read coefficient */
   DM(I0,M3) = MX0;                             /* Put Input sample in delay line */
   CNTR=Samps_per_Bin;
   DO mult_acc UNTIL CE;
   .repeat (Taps-1);
       MR=MR+MX0*MY0(SS), MX0=DM(I0,M3), MY0=PM(I4,M4);
   .end_repeat;
   MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M4);
   SR=ASHIFT MR2 (HI), MX0=DM(I1,M1);
   SR=SR OR LSHIFT MR1 (LO), DM(I0,M3) = MX0;
   DMPG1 = page(SHARED_MEM);
  MR = 0, DM(I2,M1)=SR0;                        /* Write to shared memory */
mult_acc:
   DMPG1 = 0;
   RTS (db);
   MX0 = DM(I0,M1);                             /* Update delay pointer */
   MX0 = DM(I1,M3);
```

Fir_core_p1.asm

```
/************************************************************************
  CONSTANT & MACRO DEFINITIONS
************************************************************************/
#define     Num_Samp            512                     /* NUmber of Input samples */
#define     Samps_per_Bin       40                      /* Number of Input samples in each bin */
#define     Taps                31                      /* Number of filter taps */
#define     Shift               2                       /* Number of shifts */
#define     Num_Bins            (Num_Samp/Samps_per_Bin)/* Number of bins */

/************************************************************************
  EXTERNAL DECLARATIONS
************************************************************************/
.EXTERN SHARED_MEM;

/* DM data */
.section/data seg_dmda;
.VAR    OUT[Num_Samp];              /* Output buffer */
.VAR    Delay_Line[Taps];           /* Delay line */


/* PM data */
.section/pm seg_pmda;
.VAR    COEFF[Taps] = "coeff_h.dat";    /* High pass filter coefficients */


/* PM interrupt vector code */
.section/pm seg_rth;
      JUMP start; NOP; NOP; NOP;

/* PM DSP-DSP interrupt vector code */
.section/pm IVdspdspint;
     ax0=0x1000; RTI; NOP; NOP;

/* Program memory code */
.section/pm seg_pmco;
start:
   I0=Delay_Line;                   /* Initialize delay line pointer */
   M1 = 1;
   I2 = OUT;                        /* Initialize Output pointer */
   L2=0;                            /* Initialize for modulo addressing */
   M3=-1;
   M4=1;
   SE = Shift;
   DMPG2=page(COEFF);
   L0=LENGTH (Delay_Line);          /* Initialize delay line circular buffer */
   ax0 = Delay_Line;
   REG(b0) = ax0;                   /* Initialize pointer to delay line */
   L1=LENGTH (SHARED_MEM);          /* Initialize shared memory circular buffer */
```

```
   ax0 = SHARED_MEM;
   REG(b1) = ax0;                       /* Initialize pointer to shared memory */
   L4=length(COEFF);                    /* Initialize coeficient circular buffer */
   ax0 = COEFF;
   reg(b4) = ax0;                       /* Initialize pointer to coeficient */
/**  Zero the Delay Line **/
   CNTR=Taps;                           /* 'Taps' location delay line */
   DO zero UNTIL CE;
zero:  dm(I0,M1)=L0;                    /* Delay_line = 0 */

   AY0=IMASK;
   AY1=0x0100;
   AR= AY0 or AY1;                      /* Enable DSP-DSP Interrupts */
   IMASK=AR;
   ENA INT;                             /* Enable global interrupts */
   AY0 = 0x0200;                        /* Initialize 'Bin Ready' check */
   CNTR = Num_Bins;                     /* Num_Bins to process Num_Samp  */
   do per_bin until CE;
wait_bin_rdy:
   ax0=reg(0x34);
   ar = ax0 AND ay0;
   if eq jump wait_bin_rdy;             /* Wait for 'Bin Ready' flag */
   call fir_p1 (db);                    /* Call FIR routine */
   I4=COEFF;                            /* Load Coeff buffer pointer */
   I1 = SHARED_MEM;                     /* Initialize Shared Memory pointer */
   ax0=0x0004;
   reg(0x34)=ax0;                       /* Assert 'Bin Request' Interrupt */
   ax0=0x0000;                          /* De-assert 'Bin Request' */
   reg(0x34)=ax0;
   ay0=0x1000;
bin_rqst_ack:
   ar = ax0 AND ay0;
   if eq jump bin_rqst_ack;             /* Wait for 'Bin Request Recieved' interrupt */
per_bin:
   ay0 = 0x0200;                        /* Re-initialize 'Bin Ready' check */

looping:nop;
    JUMP looping;                       /* Loop upon itself */

/************************************************************************
  FIR SUBROUTINE
************************************************************************/
fir_p1:
  DMPG1 = page(SHARED_MEM);
  MR = 0, MX0 = DM(I1,M1), MY0 = PM(I4,m4);         /* Read Input sample from shared memory and read coefficient */
  DMPG1 = 0;
  DM(I0,M3) = MX0;                                  /* Put Input sample in delay line */
  CNTR=Samps_per_Bin;
  DO mult_acc UNTIL CE;
     .repeat (Taps-1);
              MR=MR+MX0*MY0(SS), MX0=DM(I0,M3), MY0=PM(I4,M4);
     .end_repeat;
     MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M4);
     DMPG1 = page(SHARED_MEM);
     SR=ASHIFT MR2 (HI),  MX0=DM(I1,M1);
                          DMPG1 = 0;
     SR=SR OR LSHIFT MR1 (LO), DM(I0,M3) = MX0;
mult_acc: MR = 0, DM(I2,M1)=SR0;                    /* Write to output */
  RTS (db);
  MX0 = DM(I0,M1);                                  /* Update delay pointer */
  NOP;
```

Fir_shared.asm

```
/************************************************************************
  CONSTANT & MACRO DEFINITIONS
************************************************************************/
#define    Samps_per_Bin      40              /* Number of samples in each bin */

/************************************************************************
  GLOBAL DECLARATIONS
************************************************************************/
.global SHARED_MEM;

/* Shared DM data */
.section/data Dm_Shared1;
.var              SHARED_MEM   [Samps_per_Bin];          /* Initialize buffer in shared memory */
```