# **ANALOG** Engineer To Engineer Note EE-129

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



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:

Copyright 2001, Analog Devices, Inc. All rights reserved. Analog Devices assumes no responsibility for customer product design or the use or application of customers' products or for any infringements of patents or rights of others which may result from Analog Devices assistance. All trademarks and logos are property of their respective holders. Information furnished by Analog Devices Applications and Development Tools Engineers is believed to be accurate and reliable, however no responsibility is assumed by Analog Devices regarding the technical accuracy of the content provided in all Analog Devices' Engineer-to-Engineer Notes.

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	Direction	Function	DSP Core
Bit			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	4
	_	Bus Access Status	
13	Input	PDC Interface Busy	5
		Status (write from DSP	
		pending)	
14	Input	Reserved	
15	Input	Register Bus Lock	7
		Status	

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;
req(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.

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;
```

Page 3

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 #define Num Bins	40 (Num Samp/Sampa par Bin)	/* 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;	/* I	
VAR IN[Num_Samp] = "input_dec.dat";	/* Input buffer */ /* Delay line */	
	/ Delity life /	
/* PM data */		
.section/pm seg_pmda; VARCOEFEITaps1 = "coeff 1 dat":	/* Low pass filter coefficients	*/
true configues, coon_i.u.e.,	/ Low pass little 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;		
R11 (db); reg(0x24)=ex0:		
ax0=0x1000:	/* Assert DSP-DSP interrupt	to core 1 */
	· · · · · · · · · · · · · · · · · · ·	
/* Program memory code */		
.section/pm seg_pmco;		
I0=Delay Line:	/* Initialize delay line pointer	*/
M1 = 1;	· ····································	,
I1 = IN;	/* Initialize Input pointer */	
$I2 = SHARED_MEM;$	/* Initialize Shared Memory p	pointer */
L1=0; M3=-1	/* Initialize for modulo addre	ssing */
M4=1;		
SE = Shift;		
DMPG2=page(COEFF);	/* Initializa dalay lina airayla	r huffar */
ax0 = Delay Line:	/* Initialize delay life circula	i ouner 7
REG(b0) = ax0;	/* Initialize pointer to delay li	ne */
L2=LENGTH (SHARED_MEM);	/* Initialize shared memory ci	rcular buffer */
$axu = SHARED_MEM;$ REG(B2) = ax0;	/* Initialize pointer to shared	memory */
L4=length(COEFF):	/* Initialize politici to shared	buffer */
ax0 = COEFF;		
$\operatorname{reg}(b4) = ax0;$	/* Initialize pointer to coeficie	ent */
CNTR=Tans:	/* 'Tans' location delay line *	
DO zero UNTIL CE;	/ rups location dealy line /	
zero: dm(I0,M1)=L0;	/* Delay_line = 0 */	
A VOLTMA CIZ.		
A I U=IMASK; AY1=0x0100		
AR = AY0  or  AY1;	/* Enable DSP-DSP Interrupt	\s */
IMASK=AR;		
ENA INT; CNTP – Num Pinc:	/* Enable global interrupts */	Some */
do per bin until CE.	/ Num_bins to process Num	i_Samp '/
call fir_p0 (db);	/* Call FIR routine */	
I4=COEFF;	/* Load Coeff buffer pointer	*/
nop;	/* Accort 'Din Pagded flag */	
$reg(0x34)=ax0^{-1}$	/ Assert Dill Reauy flag */	
ay0=0x1000;		
bin_rqst:		

## EE-129

Page 4

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: non JUMP looping; /\* Loop upon itself \*/ FIR SUBROUTINE fir\_p0: MR = 0, MX0 = DM(I1,M1), MY0 = PM(I4,m4);DM(I0,M3) = MX0; /\* Read Input sample and read coefficient \*/ /\* Put Input sample in delay line \*/ CNTR=Samps\_per\_Bin; DO mult\_ace 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(10,M1), MY0=PM(14,M4); SR=ASHIFT MR2 (HI), MX0=DM(11,M1); SR=SR OR LSHIFT MR1 (LO), DM(10,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); MX0 = DM(I1,M3); /\* Update delay pointer \*/

#### Fir\_core\_p1.asm

/********	*****	******	6*
CONSTAN ********	T & MACRO DEFINITIONS	**********	*/
#define #define #define #define #define	Num_Samp Samps_per_Bin Taps Shift Num Bins	512 40 31 2 (Num Samp/Samps per Bin)	/* NUmber of Input samples */ /* Number of Input samples in each bin */ /* Number of filter taps */ /* Number of bints */ /* Number of bins */
/******		*****	**
EXTERNA	L DECLARATIONS	*****	*/
.EXTERN SH	IARED_MEM;		7
/* DM data */ .section/data s .VAR OUT .VAR Delay	seg_dmda; [Num_Samp]; /_Line[Taps];	/* Output buffer */ /* Delay line */	
/* PM data */ .section/pm se .VAR COE	eg_pmda; FF[Taps] = "coeff_h.dat";	/* High pass filter coefficients */	
/* PM interrup .section/pm se JUMP st	pt vector code */ g_rth; art; NOP; NOP; NOP;		
/* PM DSP-D .section/pm IV ax0=0x10	SP interrupt vector code */ /dspdspint; 000; RTI; NOP; NOP;		
/* Program m .section/pm se start:	emory code */ eg_pmco;		
I0=Delay_I M1 = 1;	Line;	/* Initialize delay line pointer */	
I2 = OUT; L2=0; M3=-1; M4=1;		/* Initialize Output pointer */ /* Initialize for modulo addressing */	
SE = Shift; DMPG2=p L0=LENG ax0 = Delay	age(COEFF); ГН (Delay_Line); y_Line;	/* Initialize delay line circular buffer */	
REG(b0) =	ax0;	/* Initialize pointer to delay line */	
L1=LENG	TH (SHARED_MEM);	/* Initialize shared memory circular buffer */	/

#### EE-129

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

#### Page 5

and of MARED, MEME, REX(b) = 200, for the shared memory */ L-4-negRICOEFF; // Infinition conduction circular buffer */ and by = 10, // Infinition conduction circular buffer */ ** Zero the Duby Line **/ (CNRF=Tips: // Top Conduction duby line */ DO neor UNTL CE: 200, for the Duby Line **/ AVI-960000 AVI-960000 AVI-960000 AVI-960000 AVI-960000 AVI-960000 AVI-960000 AVI-960000 AVI-9600000 AVI-9600000 AVI-9600000 AVI-9600000 AVI-9600000 AVI-9600000 AVI-9600000 AVI-9600000 AVI-9600000 AVI-9600000 AVI-9600000 AVI-96000000 AVI-9600000 AVI-9600000 AVI-96000000 AVI-960000000 AVI-96000000 AVI-960000000 AVI-96000000000000000000000000000000000000	ax0 = SHARED_MEM;         /*           REG(b1) = ax0;         /*           L4=length(COEFF);         /*           ax0 = COEFF;         /*           reg(b4) = ax0;         /*           CNTR=Taps;         /*           DO zero UNTIL CE;         zero:           zero: dm(10,M1)=L0;         /*	<pre>* Initialize pointer to shared memory */ * Initialize coeficient circular buffer */ * Initialize pointer to coeficient */ * Taps' location delay line */ * Taps' location delay line */ * Delay_line = 0 */ * Enable DSP-DSP Interrupts */ * Enable global interrupts */</pre>
add         add           reglob = add, add         ** Zero the Delay Law **/           Add         ** Zero the Delay Law **/           Add         ** Tags location delay line * /           Add         ** Zero the Delay Law **/           are add Add or Add o	ax0 = COEFF;         /*           reg(b4) = ax0;         /*           Zero the Delay Line **/         /*           CNTR=Taps;         /*           DO zero UNTIL CE;         /*           zero: dm(10,M1)=L0;         /*	<pre>* Initialize pointer to coefficient */ * Taps' location delay line */ * Delay_line = 0 */ * Enable DSP-DSP Interrupts */ * Enable global interrupts */</pre>
<pre>/** Zeri the Endoy Line **/ ** Tup' location delay ine */ DO zero UNTIL CE; ** Tup' location delay ine */ DO zero UNTIL CE; ** Tup' location delay ine */ DO zero UNTIL CE; ** Tup' location delay ine */ DO zero UNTIL CE; ** Tup' location delay ine */ DO zero UNTIL CE; ** Tup' location delay ine */ Note: ** *********************************</pre>	/* Zero the Delay Line **/ CNTR=Taps; /* DO zero UNTIL CE; zero: dm(I0,M1)=L0; /*	<pre>* Taps' location delay line */ * Delay_line = 0 */ * Enable DSP-DSP Interrupts */ * Enable global interrupts */</pre>
CNRe-Taps:       /* Tap' location delay line */         Do zeo UNIT. E;       /*         xno: dn()AUJ=L0;       /* Delay Line = 0 */         ANT=ANDO YO AY1;       /* Enable DSP-DSP Interrupts */         ANS-AY0 or AY1;       /* Enable DSP-DSP Interrupts */         MASS-AR,       /*         CNR = Num Ene;       /* Num Ene;         of point unit C;       /* Num Ene;         and Prog()       /* Initialize 'Bin Ready' line */         AND = 00.000;       /* Initialize 'Bin Ready' line */         and Prog()       /* Num Ene;         and Prog()       /* Wait for 'Bin Ready' line */         and Prog()       /* CAIT R routine */         and Prog()       /* Calt Calt Plane (Pointer */)         and Prog()       /* Calt Calt Plane (Pointer */)         and Prog()       /* De assert 'Bin Ready' line */         and Prog()       /* De assert 'Bin Ready' line */         and Prog()       /* De assert 'Bin Ready' line */         and Prog()       /* De assert 'Bin Ready' line */         and Prog()       /* De assert 'Bin Ready' line */         and Prog()       /* De assert 'Bin Ready' line */         and Prog()       /* De assert 'Bin Ready' line */         and Prog()       /* De assert 'Bin Ready' line */	CNTR=Taps; /* DO zero UNTIL CE; zero: dm(10,M1)=L0; /*	* Taps' location delay line */ * Delay_line = 0 */ * Enable DSP-DSP Interrupts */ * Enable global interrupts */
DO zero UNTIL CE; zero dn00M <sup>1</sup> D.6, / Delay_ine = 0 */ AY =-MASS; AY =-AVO or AYI; // Eable DSP-DSP Interrupts */ AYA =-AVO or AYI; // Eable DSP-DSP Interrupts */ AYA =-AVO or AYI; // Eable DSP-DSP Interrupts */ AYA =-0.0200; // Intailer Bin Ready check */ CNTR +-Num, Bins; // Num_Bins to process Num_Samp */ do cpr Jbu unit CE; vait Jn rdy: avd-reg(x5A); if cp jump wait, bm rdy; // W ant for Bin Ready flag */ if colump bm rdy if // Intailer Bin Ready flag */ if colump bm rdy if // Intailer Bin Ready flag */ if colump bm rdy if // Intailer Bin Ready flag */ if colump bm rdy if // Intailer Bin Ready flag */ if colump bm rdy if // Intailer Bin Ready flag */ if colump bm rdy if // Intailer Bin Ready flag */ if colump bm rdy if // Intailer Bin Ready flag */ if colump bm rdy if // Intailer Bin Ready flag */ if colump bm rdy if // Beasert Bin Request Interrupt */ avd-bo000; // Deasert Bin Request Interrupt */ prof.shr. avd =-0.0200; // Re-initailer Bin Ready check */ Bin gelp. Shr. avd =-0.0200; // Re-initailer Bin Ready check */ Bonging rop; I// Loop upon itself */ I// Loop upon itself */ I// Excent Interrupt */ prof.shr. avd =-0.0200; // P Loop upon itself */ I// Excent Interrupt */ prof.shr. avd =-0.0200; // P Loop upon itself */ I// Loop upon itself */ I// Loop upon itself */ I// Excent Interrupt */ prof.shr. avd =-0.0200; // P Loop upon itself */ DWG if page(HARED MEM); MR =-0. XXO = 0.011.010; MV0 ==M(H_M); MR =-0. XXO = 0.011.010; MV0 ==M(H_M); SH ==0. XXO = 0.011.010; MV0 ==M(H_M,H_M); SH ==0. XXO = 0.011.010; MV0 ==M(H_M,H_M); SH ==0. XXO = 0.011.010; MV0 ==M(H_M,H_M); SH ==0. XXO = 0.011.010; MV0 ==M(H_M,H_M);	DO zero UNTIL CE; zero: dm(I0,M1)=L0; /*	* Delay_line = 0 */ * Enable DSP-DSP Interrupts */ * Enable global interrupts */
AY - Mori Dic, AY - Mori Dic, AY - Mori Dic, AY - Mori Dic, ENA NY, * Enable global interrupts */ INASKAK, ENA NY, * Enable global interrupts */ INASKAK, ENA NY, * Enable global interrupts */ AY - 0.0200; * Initiative Tim Ready Check */ CVTR = Num, Bins; * Num, Bins to process Num, Samp */ do per glo multi CE; ** and AND ay0; ** and - moti CC; ** and AND ay0; ** and ND ay0; ** Last Differ Differ Pointer */ ** and AND ay0; ** De assert Bin Request Received interrupt */ ** and AND ay0; ** Last Differ Pointer */ ** And AND ay0; ** And AND ay0; ** Last Differ Pointer */ ** Last Differ Pointer */ ** Last Differ Pointer */ ** Last Differ Pointer */ ** And AND AND AND AND AND AND AND ** And	A MO-IN ( A CIZ.	* Enable DSP-DSP Interrupts */
AV1-bol010; AR=AY0 or AY1; /* Enable DSP-DSP Interrupts */ MASK~AK2; /* Enable global interrupts */ AY0 = 0x0200; /* Initialize Bn Ready check */ CVTR = Num Bine; /* Num_Bine process Num_Samp */ AY0 = 0x0200; /* Initialize Bn Ready flag */ call fig. 1 (db); /* Call TR routine */ 1 = SHARED_MEM; /* Initialize Shared Memory pointe */ reg00:34j=a.0; /* Assert Bin Request Interrupt */ reg00:34j=a.0; /* Assert Bin Request Interrupt */ reg00:34j=a.0; /* Assert Bin Request Interrupt */ reg00:34j=a.0; /* Assert Bin Request Recieved interrupt */ reg00:34j=a.0; /* Re-initialize 'Bin Request Recieved interrupt */ point /* Recived Point /* Point Point */ Point /* Recieved in	AY0=IMASK;	* Enable DSP-DSP Interrupts */
ARE - AVO or AV1;       /* Enable Bob DSP-DSP Interrupts */         MASK-AK;       /* Enable global interrupts */         ENA INT:       /* Enable global interrupts */         AV0 - 0x0200;       /* Initialize Bin Ready Check */         CNTR - Num Bins:       /* Num_Bins to process Num_Samp */         do per_Jon mult CE;       /*         um CD-rupt;       /*         I = SHARED SMC;       /*         Land Coeff buffer pointer */         ax0=0x0000;       /* De-assert 'Bin Request interrupt */         ax0=0x0000;       /* De-assert 'Bin Request interrupt */         ay0=0x0100;       /* De-assert 'Bin Request interrupt */         ay0=0x0200;       /* Re-initialize 'Bin Ready check */         boping:       /* Loop upon itself */         /* Tet page(SHARED_MEN);	AY1=0x0100;	* Enable DSP-DSP Interrupts */
IDCONSTANC,       /* Enable global interrupts */         ATU- 0x020;       /* Initialize Toin Ready check */         ATU- 0x020;       /* Num June;         ATU- 0x020;       /* Num June;         add prof.       /* ORTF         (all fin p) (db);       /* Call FiR routine */         11 - STARED MEM;       /* Initialize Shared Memory pointe */         add-bc000b;       /* Assert Tisn Request Interrupt */         add-bc000b;       /* De-assert Tisn Request Interrupt */         add-bc000b;       /* De-assert Tisn Request Interrupt */         add-bc000b;       /* Lose pupon itself */         add-bc000b;       /* Re-initialize Tisn Request Interrupt */         add-bc000b;       /* Re-initialize Tisn Request Recieved interrupt */         add-bc000b;       /* Lose pupon itself */         add-bc000b;       /* Re-initialize Tisn Ready check */         looping;       /* Loop upon itself */         aff end MAD add;       /* Read Input sample from shared memory and read coefficient */         D// Tisne padd(SHARED_MEM);       /* Read Input sample fr	AR= AY0 or AY1; /*	* Enable global interrupts */
AY0 = ont200;       * limitalize 'lim Ready' check */         CNTR = Num, Bins;       * Num_Bins to process Num_Samp */         do per Jin unil CE;       *         axd=reglo24);       ar = aod ND ay0;         if eq jump vait, bin ry0;       * Vait for 'Bin Ready' flag */         call fr_p1 (db);       * Call FR routine */         14 = COEFF;       * Load Coeff buffer pointer */         11 = SHARED_MEX;       * load Coeff buffer pointer */         14 = Stock0000;       /* Assert 'Bin Request' Interrupt */         axd=0x0000;       /* De-assert 'Bin Request' Interrupt */         axd=0x0000;       /* De-assert 'Bin Request' Interrupt */         axd=0x1000;       /* Nation 'Bin Request' Interrupt */         axd=0x0000;       /* De-assert 'Bin Request' Interrupt */         axd=0x1000;       /* Nation 'Bin Request Interrupt */         ayd=0x1000;       /* Re-initialize 'Bin Ready' check */         looping rop;       JUMP looping;       /* Loop upon itself */         fr_g1:       DMROI = page(SHARED_MEM);       /* Read Input sample from shared memory and read coefficient */         DMROI = page(SHARED_MEM);       /* P tu Input sample in delay line */         MR = 0, MX0 = DM(II,MI), MY0 = PM(I4,M4);       /* P tu Input sample in delay line */         CNTR = Samps, pre Bin;       DO mait act UNTIL CE;	ENA INT· /*	
CNTR = Num, Bins;       /* Num_Bins to process Num_Samp */         do per_Join unil CC;;       /*         wait, bin_rdy;       ar = ax0 AND ay0;         if eq_Jung vait, bin_rdy;       /* Wait for 'Bin Ready' flag */         call fn.p1 (db);       /* Call FR rotine */         14-COEFF;       /* Lad Coeff buffer pointer */         11 = SHARED_MEM;       /* Initialize Shared Memory pointer */         ax0-cx0000;       /* De-assert 'Bin Request' Interrupt */         ax0-cx0000;       /* De-assert 'Bin Request' /         reg(0:34)=ax0;       /* Assert 'Bin Request' Interrupt */         ay0-ck1000;       /* De-assert 'Bin Request' Interrupt */         ay0-ck1000;       /* De-assert 'Bin Request Recieved' interrupt */         ay0-ck1000;       /* De-assert 'Bin Request Recieved' interrupt */         ay0-dx1000;       /* Recinitalize 'Bin Request Recieved' interrupt */         ay0-dx1000;       /* Recinitalize 'Bin Request Recieved' interrupt */         ar = av0 AND ay0;       /* Recinitalize 'Bin Request Recieved' interrupt */         ar = av0 AND ay0;       /* Recinitalize 'Bin Request Recieved' interrupt */         ar = av0 AND ay0;       /* Recinitalize 'Bin Request Recieved' interrupt */         ar = av0 AND ay0;       /* Recinitalize 'Bin Request Recieved' interrupt */         per_f bin:       -	AY0 = 0x0200; /*	* Initialize 'Bin Ready' check */
do per, joi until CE; ax0-reg10543; ar = ax0 AND ay0; if eq jump wai, bin.rdy; /* Wait for Bin Ready'flag */ call fr.p1 (db); /* Call FIR routine */ Lad Coef buffer pointer */ Lad Coef buffer pointer */ Ax0-0x0000; /* Lessert Bin Request // reg10543/ax0; /* Assert Bin Request */ reg10543/ax0; /* De-assert Bin Request */ reg10543/ax0; /* De-assert Bin Request */ reg10543/ax0; /* De-assert Bin Request */ reg10543/ax0; /* Assert Bin Request */ reg10543/ax0; /* Wait for Bin Request Recieved interrupt */ ay0 = 0x100; /* Re-initialize Bin Ready' check */ looping.no; JUMP looping; /* Loop upon itself */ ***********************************	CNTR = Num_Bins; /*	* Num_Bins to process Num_Samp */
<pre>wait.bm_rdy: ar = x0 AND ay0; if eq jump wait.bm_rdy:</pre>	do per_bin until CE;	
axy=regions 41; ar = axo AND ay0; if eq jump bat bin _db; /* Vait for 'Bin Ready flag */ call fr _p1 (db); /* Call FIR routine */ H=COEFFF: /* Load Coeff buffer pointe */ H=COEFFF: /* Load Coeff buffer pointe */ ta = StARED_MEM; /* Initialize Shard Memory pointer */ ax0=0x0000; /* Assert 'Bin Request Interrupt */ ax0=0x0000; /* assert 'Bin Request 'Interrupt */ ax0=0x0000; /* assert 'Bin Request 'Interrupt */ ax0=0x0000; /* assert 'Bin Request 'Interrupt */ ax0=0x0000; /* assert 'Bin Request Recieved 'interrupt */ ax0=0x000; /* k=-initialize 'Bin Ready check */ looping_nop; JUMP looping; /* Loop upon itself */ /**********************************	wait_bin_rdy:	
a = AV AVD a by, if or jurny wait obm_rdy; /* Wait for 'Bin Ready' flag */ if or jurny wait obm_rdy; /* Call PIR rotatine */ H = SHARED_MEM; /* Call PIR rotatine */ H = SHARED_MEM; /* Initialize Shared Memory pointer */ avd=0x000d; reg(0x34)=ax0; /* Assert 'Bin Request Interrupt */ avd=0x0000; /* De-assert 'Bin Request Interrupt */ avd=0x0000; /* De-assert 'Bin Request Interrupt */ avd=0x0000; /* De-assert 'Bin Request Interrupt */ avd=0x0000; /* Wait for 'Bin Request Interrupt */ avd=0x0000; /* Wait for 'Bin Request Recieved' interrupt */ if eq jurny bin_rqst_ack; /* Wait for 'Bin Request Recieved' interrupt */ per bin: avg = 0x0200; /* Re-initialize 'Bin Ready' check */ looping:pp: JUMP looping; /* Loop upon itself */ FIR SUBBOUTINE FIR SUBBOUTINE FIR SUBBOUTINE FIR SUBBOUTINE M = 0, MX0 = DM(11,M1), MY0 = PM(14,m4); /* Read Input sample from shared memory and read coefficient */ DM(0,M3) = MX0; /* Put Input sample in delay line */ CNTR=Sampa_per_Bin; DM4(0,M3) = MX0; /* Put Input sample in delay line */ CNTR=Sampa_per_Bin; DM4(1 = page(SHARED_MEM; MR=MR=MX0*MY0(SS), MX0=DM(10,M1), MY0=PM(14,M4); end (repeat; MR=MR=MX0*MY0(SS), MX0=DM(10,M1), MY0=PM(14,M4); DMFG1 = page(SHARED_MEM; SR=ASSILPI LSHIFT INR (LO, DM(10,M3) = MX0; DMFG1 = 0; SR=SR CR LSHIFT INR (LO, DM(10,M3) = MX0;	ax0=reg(0x34);	
<pre>call fir p1 (db); /* Call FIR routine */ H=COEFF; /* Load Coef Buffer pointer */ ax0=0x0000; /* Lessert Bin Request Interrupt */ ax0=0x0000; /* De-assert Bin Request Interrupt */ ax0=0x0000; /* De-assert Bin Request */ reg(0x34)=ax0; /* Assert Bin Request Interrupt */ ax0=0x0000; /* De-assert Bin Request */ reg(0x34)=ax0; /* Wait for 'Bin Request excieved' interrupt */ per_g(0x34)=ax0; /* Wait for 'Bin Request Recieved' interrupt */ ay0=0x0200; /* Re-initialize 'Bin Ready' check */ looping:nop; JUMP looping; /* Loop upon itself */ /* FIR SUBROUTINE ************************************</pre>	if eq jump wait bin rdv: /*	* Wait for 'Bin Ready' flag */
I 1= STARED_MEN; /* Load Coeff buffer pointer */ and=0x0000; /* Initialize Shared Memory pointer */ and=0x0000; /* Assert 'Bin Request Interrupt */ arg(0x3)=ax0; arg=ax0 AND ay0; /* De-assert 'Bin Request '/ reg(0x3)=ax0; arg=ax0 AND ay0; if eq jump bin_regt_ack; /* Wait for 'Bin Request Recieved' interrupt */ if eq jump bin_regt_ack; /* Wait for 'Bin Request Recieved' interrupt */ per_bin: ay0 = 0x0200; /* Re-initialize 'Bin Request Recieved' interrupt */ if eq jump bin_regt_ack; /* Loop upon itself */ // FIR SUBROUTINE FIR SUBROUTINE FIR SUBROUTINE MR = 0, MX0 = DM(I1,MI), MY0 = PM(I4,M4); /* Read Input sample from shared memory and read coefficient */ DMFG[i = o, DMFG[i = o, DMFG[i = o, MR=M=MX0*MY0(SS), MX0=DM(I0,MI), MY0=PM(I4,M4); e_requert (Taps-1); MR=M=MX0*MY0(SS), MX0=DM(I0,MI), MY0=PM(I4,M4); e_requert (Taps-1); MR=MR=MX0*MY0(SS), MX0=DM(I0,MI), MY0=PM(I4,M4); e_requert (Taps-1); MR=MR=MX0*MY0(SS), MX0=DM(I0,MI), MY0=PM(I4,M4); e_requert (Taps-1); MR=MR=MX0*MY0(SS), MX0=DM(I0,MI), MY0=PM(I4,M4); e_requert (Taps-1); MR=MR=MX0*MY0(SS), MX0=DM(I0,MI), MY0=PM(I4,M4); DMFGI = 0, SR=SSR OR LISHIFT MR2 (10), DMX0=DMX0: EXENCE ON EXENCE ON E	call fir_p1 (db); /*	* Call FIR routine */
I I = SHARED_MEM; /* Initialize Shared Memory pointer */ axd=0x004; reg(0x34)=ax0; /* Assert Bin Request Interrupt */ axd=0x0000; /* De-assert Bin Request */ reg(0x34)=ax0; /* De-assert Bin Request */ reg(0x34)=ax0; /* De-assert Bin Request */ reg(0x34)=ax0; /* Wait for 'Bin Request Recieved' interrupt */ ay0=0x100; bin_rqut_ack: ar = ax0 AND ay0; if eq jump bin_rqut_ack; /* Wait for 'Bin Request Recieved' interrupt */ er	I4=COEFF; /*	* Load Coeff buffer pointer */
ax0-0x000; 'A Asert Bin Request Interrupt */ ax0-0x0000; /B De-asert Bin Request */ reg(0x34)=ax0; /B De-asert Bin Request Recieved interrupt */ ar = ax0 AND ay0; // ar = ax0 AN	I1 = SHARED_MEM; /*	* Initialize Shared Memory pointer */
<pre>reg(0x-y-ax0,</pre>	ax0=0x0004; rag(0x24)=ax0; /*	* Accord 'Din Doquant' Internut */
reg(0x34)=x0; ay0=0x100; bin_rqst_ack; /* Wait for 'Bin Request Recieved' interrupt */ per_bin: ay0 = 0x0200; /* Re-initialize 'Bin Ready' check */ kooping.nop; JUMP looping; /* Loop upon itself */ /**********************************	ax0=0x0000.	Asset: Bin Request interrupt -/ **
ay0=ox1000; 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 */ kooping:nop; JUMP looping; /* Loop upon itself */ /**********************************	reg(0x34)=ax0;	
bin_rgst_ack: if eq jump bin_rgst_ack; /* Wait for 'Bin Request Recieved' interrupt */ per_bin: ay0 = 0x0200; /* Re-initialize 'Bin Ready' check */ looping:nop; JUMP looping; /* Loop upon itself */ /**********************************	ay0=0x1000;	
ar = av0 AND av0; if eq jump bin_rqst_ack; /* Wait for 'Bin Request Recieved' interrupt */ per_bin: av0 = 0x0200; /* Re-initialize 'Bin Ready' check */ looping:nop; JUMP looping; /* Loop upon itself */ /**********************************	bin_rqst_ack:	
<pre>n equipmp bin_get_akx, // win ton bin Request Received interrupt */ per_bin:     ay0 = 0x0200; /* Re-initialize 'Bin Ready' check */ looping:nop;     JUMP looping; /* Loop upon itself */ /**********************************</pre>	ar = ax0 AND ay0;	* Writh for This Desured Desired interment */
ay0 = 0x0200; /* Re-initialize 'Bin Ready' check */ looping:nop: JUMP looping; /* Loop upon itself */ /**********************************	ner bin:	wan to bin kequest kecleved interrupt '/
looping:nop; JUMP looping;       /* Loop upon itself */         /************************************	ay0 = 0x0200; /*	* Re-initialize 'Bin Ready' check */
JUMP looping;       /* Loop upon itself */         /************************************	looping:nop;	
/*************************************	JUMP looping; /*	* Loop upon itself */
FIR SUBROUTINE ************************************	/*******	*****
<pre>************************************</pre>	FIR SUBROUTINE	
<pre>ml_pi: DMPG1 = page(SHARED_MEM); MR = 0, MX0 = DM(I1,MI), 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); .gnd_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:</pre>	***************************************	***************************************
<pre>DM OF = page(DFACLE) = page(DFACLE) = PM(14,m4); /* Read Input sample from shared memory and read coefficient */ DMPG1 = 0; DM(10,M3) = MX0; /* Put Input sample in delay line */ CNTR=Samps_per_Bin; DO mult_ace UNTLL CE; .repeat (Taps-1); MR=MR+MX0*MY0(SS), MX0=DM(10,M3), MY0=PM(14,M4); .end_repeat; MR=MR+MX0*MY0(SS), MX0=DM(10,M1), MY0=PM(14,M4); SR=ASHIFT MR2 (HI), MX0=DM(10,M1), MY0=PM(14,M4); DMPG1 = page(SHARED_MEM); SR=ASHIFT MR2 (HI), MX0=DM(11,M1); DMPG1 = 0; SR=SR OR LSHIFT MR1 (LO). DM(10,M3) = MX0:</pre>	$III_pI:$ DMPG1 = page(SHARED_MEM):	
DMPG1 = 0; /* 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;	MR = 0, $MX0 = DM(11.M1)$ , $MY0 = PM(14)$	
DM(10,M3) = MX0; /* Put Input sample in delay line */ CNTR=Samps_per_Bit; DO mult_acc UNTIL CE; .repeat (Taps-1); MR=MR+MX0*MY0(SS), MX0=DM(10,M3), MY0=PM(14,M4); .end_repeat; MR=MR+MX0*MY0(SS), MX0=DM(10,M1), MY0=PM(14,M4); DMPG1 = page(SHARED_MEM); SR=ASHIFT MR2 (HI), MX0=DM(11,M1); DMPG1 = 0; SR=SR OR LSHIFT MR1 (LO). DM(10,M3) = MX0:	DMPG1 = 0;	
CNTR=Samps_per_lsi; DO mult_acc UNTIL CE; .repeat (Taps-1); MR=MR+MX0*MY0(SS), MX0=DM(I0,M3), MY0=PM(I4,M4); .end_repeat; MR=MK+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M4); DMPGI = page(SHARED_MEM); SR=ASHIFT MR2 (HI), MX0=DM(I1,M1); DMPGI = 0; SR=SR OR LSHIFT DM(I_0M3) = MX0:	DM(I0,M3) = MX0;	/* Put Input sample in delay line */
DO MULace UNTL 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:	CNTR=Samps_per_Bin;	
Reptar (14p3 1), x         MR=MR+MX0*MY0(SS), MX0=DM(10,M3), MY0=PM(14,M4);         .end_repeat;         MR=MR+MX0*MY0(SS), MX0=DM(10,M1), MY0=PM(14,M4);         DMPG1 = page(SHARED_MEM);         SR=SR OR LSHIFT MR1 (L0), DM(10,M3) = MX0;	repeat (Tans-1):	
.end_repeat; MR=MR+MX0*MY0(SS), MX0=DM(10,M1), MY0=PM(14,M4); DMPG1 = page(SHARED_MEM); SR=ASHIFT MR2 (HI), MX0=DM(11,M1); DMPG1 = 0; SR=SR OR LSHIFT MR1 (LO). DM(10,M3) = MX0;	MR=MR+MX0*MY0(SS), MX	X0=DM(I0.M3), MY0=PM(I4.M4);
MR=MR+MX0*MY0(SS), MX0=DM(I0,M1), MY0=PM(I4,M4); DMPG1 = page(SHARED_MEM); SR=ASHIFT MR2 (H), MX0=DM(I1,M1); DMPG1 = 0; SR=SR OR LSHIFT MR1 (LO. DM(I0,M3) = MX0;	.end_repeat;	
$DMPG1 = page(SHARED_MEM);$ $SR=ASHIFT MR2 (HI), MX0=DM(11,M1);$ $DMPG1 = 0;$ $SR=SR OR LSHIFT MR1 (LO), DM(10,M3) = MX0;$	MR=MR+MX0*MY0(SS), MX0=DM(I0,	,M1), MY0=PM(14,M4);
SK-ASHFT WIX2 (III), WAX-DW(II,WI), DMPG1 = 0; SR=SR OR LSHIFT MR1 (LO), $DM(10,M3) = MX0;$	DMPG1 = page(SHARED_MEM);	
SR=SR OR LSHIFT MR1 (LO), $DM(0,M3) = MX0$ :	DMPG1 = 0	1,
	SR=SR OR LSHIFT MR1 (LO), DM(I0,M	M3) = MX0;
mul_acc: MR = 0, DM(I2,M1)=SR0; /* Write to output */	mult_acc: $MR = 0$ , $DM(I2,M1)=SR0$ ;	/* Write to output */
RTS (db):	RTS (db);	
NAU = DM(10,M1); /* Update delay pointer */	MAU = DM(10, M1); NOP;	/* Update delay pointer */

## Fir\_shared.asm

/*************************************
#define Samps per Bin 40 /* Number of samples in each bin */
#define Samps per_Bin 40 /* Number of samples in each bin */
/*************************************
.global SHARED_MEM;
/* Shared DM data */ .section/data Dm_Shared1;
.var SHARED_MEM [Samps_per_Bin]; /* Initialize buffer in shared memory */

EE-129

Page 6