CESR BPM/BSM/FLM SYSTEM DIGITAL PROCESSOR BOARD

(For XILINX V5.4)
z:\crs\Cesr_BPM_BSM\Docs\DSP_Board_Programming_V5-4.doc
12/12/2006 10:04 AM

New in 5.4:

1. Address autoincrement for ColdFire readout has been fixed.

2. The ERL_BPM module type has been created. It uses a 25 MHz input clock (instead of 24 MHz in CESR) and it doubles that to 50 MHz for data acquisition (instead of tripling it to 72 MHz in CESR). The ERL_BPM only support 122 bunches (instead of 183 in CESR).

```
MODULE_TYPE
        = 1 BSM
        = 2 BPM
        = 3 FLM
        = 4 FLMA
        = 5 ERL_BPM
MAJOR_REV = 5
MINOR_REV = 4
```

3. Timing of all DSP operations have been verified. After it boots, the DSP can no longer access the FLASH memory. It requires too many wait states.

New in 5.3:

```
MODULE_TYPE
        = 1 BSM
        = 2 BPM
        = 3 FLM
        = 4 FLMA
MAJOR_REV = 5
MINOR_REV = 3
```

1. The BSM current monitor board is supported.

New in 5.2:

1. “Force Hi Hit Register” added to the accumulator board for testing.

2. Added detail about accumulator Geo Bunch Rate Register.

3. Scrambled accumulator board mapping to use ADC channels 1 thru 6 to generate the lookup table address.

New in 5.1:

1. Support for Ethernet access thru a ColdFire DIMM board from Arcturus.

2. Reset for the DIMM module is provided thru the new register “DIMM_RESET”.

3. The “MODULE_TYPE” register has been modified so each project has its own identifier.

MODULE_TYPE
= 1 BSM
= 2 BPM
= 3 FLM
= 4 FLMA
MAJOR_REV = 5
MINOR_REV = 1

4. The FLMA accumulator board is supported.

To Do:

Implement ‘ACQ_SKIP_CNT’ register.
Implement shadow readback for timing board
Make sign extension be programmable for unipolar/bipolar
Speed up ‘loc_dat_tri_drive’ with combinatorial switching

DATA FORMAT

When reading any memory that is less than 32-bits wide, the data will be considered to be signed 2’s-complement and will be sign extended. When writing any memory that is less than 32-bits wide, the high bits will be ignored.

When reading any register that is less than 32 bits wide, the data will be considered to be unsigned and will contain zeroes in the high bits. If a register needs signed data, it will be designed as a 32 bit register. When writing any register that is less than 32 bits wide, the high bits will be ignored.

ADDRESS MAPS

The local (on-board) address bus, LOC_ADR[31..0], addresses longword (4-byte) entities. The address map for all peripherals is driven by the addressing capabilities of the DSP. The DSP breaks down the 32-bit address range as follows:

DSP INTERNAL SPACE	0X00000000 – 0X003FFFFF
DSP UNUSED	0X00400000 – 0X07FFFFFF
DSP BANK 0 (/MS0)	0X08000000 – 0X0BFFFFFF
DSP BANK 1 (/MS1)	0X0C000000 – 0X0FFFFFFF
DSP HOST (/MSH)	0X10000000 – 0XFFFFFFFF

The region labeled “unused” is specific to this project. The DSP actually defines features in this region, such as multiprocessor memory space and SDRAM memory space, but this project does not use any of the features.

DSP (from the XBUS or Ethernet perspective)

One has to use 'multiprocessor space' to access things inside the DSP from the outside. For the ADSP-TS101S TigerSHARC processor with ID=0 (which is the ID for this project), the address range is from 0x02000000 to 0x023fffff. When you want to access a location inside of the DSP, you will need to add 0x02000000 to the actual internal DSP addresses so that the XBUS or Ethernet uses the correct address. The internal DSP address will be calculated by masking off the high byte (0x02).

DSP (from the DSP perspective)

The DSP memory map is unique to the ADSP-TS101S TigerSHARC chip. Other DSPs may use different mapping.

DSP INTERNAL SPACE	0X00000000 – 0X03FFFFFF	
MEMORY BLOCK 0	0X00000000 – 0X0000FFFF	64 kW
MEMORY BLOCK 1	0X00080000 – 0X0008FFFF	64 kW
MEMORY BLOCK 2	0X00100000 – 0X0010FFFF	64 kW
INT REGISTERS (UREGS)	0X00180000 – 0X001807FF	2 kW

The LDF file defines how the memory is allocated for various functions. The current version of the file "BPM_ADSP-TS101_C.LDF" makes the following allocations:

```
// Start with full M0 block for code. We may use high addresses for some data structures.
// This gives 64k of code space.
   M0Code        { TYPE(RAM) START(0x00000000) END(0x0000FFFF) WIDTH(32) }

// M1 block will support data, heap, and stack. We expect no heap usage and very
// little stack usage. Start with 56k data, 2k heap, and 6k stack.
   M1Data        { TYPE(RAM) START(0x00080000) END(0x0008DFFF) WIDTH(32) }
   M1Heap        { TYPE(RAM) START(0x0008E000) END(0x0008E7FF) WIDTH(32) }
   M1Stack       { TYPE(RAM) START(0x0008E800) END(0x0008FFFF) WIDTH(32) }

// M2 block will support raw data from the ADCs. Start with one buffer using
// 56k. An "M2Stack" is required by the C/C++ runtime. Make it be 8k.
   M2Data        { TYPE(RAM) START(0x00100000) END(0x0010DFFF) WIDTH(32) }
   M2Stack       { TYPE(RAM) START(0x0010E000) END(0x0010FFFF) WIDTH(32) }

// This project does not use the SDRAM address range
   SDRAM         { TYPE(RAM) START(0x04000000) END(0x07FFFFFF) WIDTH(32) }

// MS0 bank will address the ADC boards.
// MS0mem will address memory on all 4 cards contiguously
   MS0mem        { TYPE(RAM) START(0x08000000) END(0x08FFFFFF) WIDTH(32) }
// MS0reg will address register space on all 4 cards contiguously
   MS0reg        { TYPE(RAM) START(0x09000000) END(0x09FFFFFF) WIDTH(32) }
// MS0unused is the remaining part of the MS0 bank
   MS0unused     { TYPE(RAM) START(0x0A000000) END(0x0BFFFFFF) WIDTH(32) }

// MS1 bank will address the FLASH and SRAM.
   MS1           { TYPE(RAM) START(0x0C000000) END(0x0FFFFFFF) WIDTH(32) }

// The HOST region will address the XILINX chip and the timing board
// Memory blocks need to be less than 2 Gig, and the total HOST space is almost 4 Gig.
// Arbitrarily, we create 7 segments of 1/4 Gig and 1 segment of 1/8 Gig.
// For this project, all of the hardware is in the first segment.
   HOST          { TYPE(RAM) START(0x10000000) END(0x2FFFFFFF) WIDTH(32) }
   HOST1         { TYPE(RAM) START(0x30000000) END(0x4FFFFFFF) WIDTH(32) }
   HOST2         { TYPE(RAM) START(0x50000000) END(0x6FFFFFFF) WIDTH(32) }
   HOST3         { TYPE(RAM) START(0x70000000) END(0x8FFFFFFF) WIDTH(32) }
   HOST4         { TYPE(RAM) START(0x90000000) END(0xAFFFFFFF) WIDTH(32) }
   HOST5         { TYPE(RAM) START(0xB0000000) END(0xCFFFFFFF) WIDTH(32) }
   HOST6         { TYPE(RAM) START(0xD0000000) END(0xEFFFFFFF) WIDTH(32) }
   HOST7         { TYPE(RAM) START(0xF0000000) END(0xFFFFFFFF) WIDTH(32) }
```

Hardware can be accessed by using pointers, as the following code snippet that accesses the timing card shows. An 'include' file should be created that symbolically defines all of the various addresses.

```
main() {
        int *tim_ptr = (int *)0x10020000;
        int i;

        for(;;) {
                for (i=0; i<1024; i++) {
                        *tim_ptr = i;
                }
        }
}
```

Analog Cards

For the BSM, FLM, and FLMA systems, each analog card has eight channels. Each channel has 512kW of memory space. There are no registers on the BSM/FLM analog cards. The FLMA cards have an accumulator module. Address line A[24] selects either memory space or accumulator space. Address lines A[23..22] select one of the four analog boards. Address lines A[21..19] select one of eight channels on a board. Address lines A[18..0] select a memory address.

ANALOG CARD 0 CHAN 0	0X08000000 - 0X0807FFFF	(dec=134217728)
ANALOG CARD 0 CHAN 1	0X08080000 - 0X080FFFFF	(dec=134742016)
ANALOG CARD 0 CHAN 2	0X08100000 - 0X0817FFFF	(dec=135266304)
ANALOG CARD 0 CHAN 3	0X08180000 - 0X081FFFFF	(dec=135790592)
ANALOG CARD 0 CHAN 4	0X08200000 - 0X0827FFFF	(dec=136314880)
ANALOG CARD 0 CHAN 5	0X08280000 - 0X082FFFFF	(dec=136839168)
ANALOG CARD 0 CHAN 6	0X08300000 - 0X0837FFFF	(dec=137363456)
ANALOG CARD 0 CHAN 7	0X08380000 - 0X083FFFFF	(dec=137887744)
ANALOG CARD 1 CHAN 0	0X08400000 - 0X0847FFFF	(dec=138412032)
ANALOG CARD 1 CHAN 1	0X08480000 - 0X084FFFFF	(dec=138936320)
ANALOG CARD 1 CHAN 2	0X08500000 - 0X0857FFFF	(dec=139460608)
ANALOG CARD 1 CHAN 3	0X08580000 - 0X085FFFFF	(dec=139984896)
ANALOG CARD 1 CHAN 4	0X08600000 - 0X0867FFFF	(dec=140509184)
ANALOG CARD 1 CHAN 5	0X08680000 - 0X086FFFFF	(dec=141033472)
ANALOG CARD 1 CHAN 6	0X08700000 - 0X0877FFFF	(dec=141557760)
ANALOG CARD 1 CHAN 7	0X08780000 - 0X087FFFFF	(dec=142082048)
ANALOG CARD 2 CHAN 0	0X08800000 - 0X0887FFFF	(dec=142606336)
ANALOG CARD 2 CHAN 1	0X08880000 - 0X088FFFFF	(dec=143130624)
ANALOG CARD 2 CHAN 2	0X08900000 - 0X0897FFFF	(dec=143654912)
ANALOG CARD 2 CHAN 3	0X08980000 - 0X089FFFFF	(dec=144179200)
ANALOG CARD 2 CHAN 4	0X08A00000 - 0X08A7FFFF	(dec=144703488)
ANALOG CARD 2 CHAN 5	0X08A80000 - 0X08AFFFFF	(dec=145227776)
ANALOG CARD 2 CHAN 6	0X08B00000 - 0X08B7FFFF	(dec=145752064)
ANALOG CARD 2 CHAN 7	0X08B80000 - 0X08BFFFFF	(dec=146276352)
ANALOG CARD 3 CHAN 0	0X08C00000 - 0X08C7FFFF	(dec=146800640)
ANALOG CARD 3 CHAN 1	0X08C80000 - 0X08CFFFFF	(dec=147324928)
ANALOG CARD 3 CHAN 2	0X08D00000 - 0X08D7FFFF	(dec=147849216)
ANALOG CARD 3 CHAN 3	0X08D80000 - 0X08DFFFFF	(dec=148373504)
ANALOG CARD 3 CHAN 4	0X08E00000 - 0X08E7FFFF	(dec=148897792)
ANALOG CARD 3 CHAN 5	0X08E80000 - 0X08EFFFFF	(dec=149422080)

ANALOG CARD 3 CHAN 6	0X08F00000 - 0X08F7FFFF	(dec=149946368)
ANALOG CARD 3 CHAN 7	0X08F80000 - 0X08FFFFFF	(dec=150470656)

ACCUMULATOR		
ANALOG CARD 0	0x0A000000	(dec=167772160)
ANALOG CARD 1	0x0A400000	(dec=171966464)
ANALOG CARD 2	0x0A800000	(dec=176160768)
ANALOG CARD 3	0x0AC00000	(dec=180355072)

Refer to the section on ACCUMULATOR PROGRAMMING for details.

For the BPM system, each analog card has two channels. Each channel has 512kW of memory and 1 gain register. Address line A[24] selects either memory space or register space. Address lines A[21..20] select one of the four analog boards. Address line A[19] selects one of two channels on a board. Address lines A[18..0] select either a memory address or a register address.

MEMORY		
ANALOG CARD 0 CHAN 0	0X08000000 - 0X0807FFFF	(dec=134217728)
ANALOG CARD 0 CHAN 1	0X08080000 - 0X080FFFFF	(dec=134742016)
ANALOG CARD 1 CHAN 0	0X08100000 - 0X0817FFFF	(dec=135266304)
ANALOG CARD 1 CHAN 1	0X08180000 - 0X081FFFFF	(dec=135790592)
ANALOG CARD 2 CHAN 0	0X08200000 - 0X0827FFFF	(dec=136314880)
ANALOG CARD 2 CHAN 1	0X08280000 - 0X082FFFFF	(dec=136839168)
ANALOG CARD 3 CHAN 0	0X08300000 - 0X0837FFFF	(dec=137363456)
ANALOG CARD 3 CHAN 1	0X08380000 - 0X083FFFFF	(dec=137887744)

GAIN REGISTERS		
ANALOG CARD 0 GAIN 0	0X09000000	(dec=150994944)
ANALOG CARD 0 GAIN 1	0X09080000	(dec=151519232)
ANALOG CARD 1 GAIN 0	0X09100000	(dec=152043520)
ANALOG CARD 1 GAIN 1	0X09180000	(dec=152567808)
ANALOG CARD 2 GAIN 0	0X09200000	(dec=153092096)
ANALOG CARD 2 GAIN 1	0X09280000	(dec=153616384)
ANALOG CARD 3 GAIN 0	0X09300000	(dec=154140672)
ANALOG CARD 3 GAIN 1	0X09380000	(dec=154664960)

FLASH Memory

FLASH	0X0C000000 - 0X0C07FFFF	(dec=201326592)

The FLASH memory is 512k by 8-bits, using an Atmel AT49LV040 chip. Its primary use is to store the DSP code.

Unlike the FLASH in other BPM projects, this chip does not have multiple sectors that can be individually erased. If the FLASH is to be used for non-volatile storage of anything other than the DSP code, the user's program will need to read and save the permanent information before reprogramming the memory. The saved information will then need to be written back to the FLASH after it has been erased and is ready for new DSP code.

Static RAM

STATIC RAM	0X0C080000 - 0X0C0FFFFF	(dec=201850880)

The static RAM is 512k by 32-bits.

Vector and Packet Support

VECTOR ADDRESS TABLE	0x10000000 - 0x100001FF	(dec=268435456)

The vector address table holds 512 addresses, each 32-bits wide. Xbus vector commands (‘vxgetn’ and ‘vxputn’) specify the first vector and the number of vectors to access. The vector address table maps Xbus vectors to hardware addresses.

The following vectors are currently defined:

0x078: FLASH (0x0C005555)
0x079: FLASH (0x0C002AAA)
0x07A: FLASH (0x0C005555)
0x07B: FLASH (0x0C005555)
0x07C: FLASH (0x0C002AAA)
0x07D: FLASH (0x0C005555)
0x07E: direct address register (0x10040000)
0x07F: This is a special vector number. The actual address comes from the 'direct_adr' register.

The standard MPM database address nodes (like “CBPM ADR TST”) are initialized with vector number 0x7E. They access the ‘DIRECT_ADR’ register.

The standard MPM database data nodes (like “CBPM DAT TST”) are initialized with vector number 0x7F. Operations to these data nodes use the address that was programmed thru the address node.

As an example, to write ‘some_data” to “some_address” in the module associated with element 2 of the nodes “CBPM ADR TST” and “CBPM DAT TST”, the control system program makes the following calls:

```
call vxputn(‘CBPM ADR TST’, 2, 2, some_address)
call vxputn(‘CBPM DAT TST’, 2, 2, some_data)
```

Vector 0X078 thru 0x07D (120 thru 125) are used for flash programming. To erase the FLASH chip, write the following data to the corresponding vector:

```
0x078=aa  0x079=55  0x07a=80  0x07b=aa  0x07c=55  0x07d=10
```

To program the FLASH chip with a vector operation, write the following data to the corresponding vector (PA is the address to program, PD is the data to program):

```
0x07b=aa  0x07c=55  0x07d=a0  0x07e=PA  0x07f=PD
```

PACKET START ADDRESS TABLE	0x10001000 - 0x100011FF	(dec=268439552)
PACKET MORE ADDRESS TABLE	0x10001800 - 0x100019FF	(dec=268441600)
PACKET SIZE TABLE	0x10010000 - 0x100101FF	(dec=268500992)

The ‘packet start address table’ and ‘packet more address table’ each hold 512 addresses, each 32-bits wide. The packet size table holds 512 values, each 12-bits wide.

The ‘packet start address table’ holds the first address of each data structure or block that is accessed for a given packet tag. The address is loaded into a counter. After each access, the counter is incremented and

the result is written into the ‘packet more address table’. After the amount of data specified in the ‘packet size table’ has been transferred, the address stored in the ‘packet more address table’ will be the address of the next piece of data in the data structure. If another packet operation is performed and the tag is offset by 2048, the first address will be retrieved from the ‘packet more address table’. This scheme allows for access to blocks of memory that are larger than the maximum size of a single packet by simply using the regular tag for the first block and the offset tag for all of the remaining blocks.

A constraint imposed by this scheme is that the size of any data structure must be a multiple of the size written in the ‘packet size table’. If a 260 word structure needs to be transferred, one can either use a 256 word packet size and pad the structure out to 512 words, or use a 130 word packet size. Additionally, the ‘packet more address table’ is read-only.

Xbus packet commands (‘vugetn’ and ‘vuputn’) specify the packet tag to use for a data transfer. The packet address tables map Xbus packet numbers to the first hardware address involved in the transfer. Packet tags from 1 thru 2047 will find the first address in the ‘packet start address table’. Packet tags from 2049 thru 4095 will find the first address in the ‘packet more address table. The packet size table specifies how many 32-bit words to transfer for a ‘read’ (vugetn) operation. For ‘write’ (vuputn) operations, the number of words written determines the transfer size.

Generally, the DSP will initialize the address and size tables with the addresses and sizes of internal data structures that the control system needs to access. An exception is for packet 0x1ff (511).

Packet 0x1FF (511) is reserved for FLASH programming. Any data written to this packet will cause the FLASH programming sequence in the XILINX chip to be invoked. The procedure to program FLASH with packet operations is:

```
! write the address of PACKET ADDRESS TABLE entry #511 to the address node
call vxputn(‘CBPM ADR TST, 2, 2, ‘100011ff’x)

!write the next FLASH address to program to the data node
call vxputn(‘CBPM DAT TST’, 2, 2, flash_adr)

! write the address of PACKET SIZE TABLE entry #511 to the address node
call vxputn(‘CBPM ADR TST, 2, 2, ‘100101ff’x)

!write the size of the packet to the data node
call vxputn(‘CBPM DAT TST’, 2, 2, size)

!send the packet of data with packet tag #511
call vuputn(‘CBPM PKT TST’, 2, 2, data_array, 511, size)
!note: the ‘data_array’ is a longword (32-bit) array with one byte per longword
```

Timing Board

TIMING BOARD 0X10020000 - 0X100200FF (dec=268566528)

These registers control delay settings on the timing board. All registers are 10-bits, and are write-only. The timing board has two timing blocks: A and B. Each block has four clock outputs, one for each analog card. The four clock outputs of a block consist of a global delay that is the sum of two global delays settings (common to all four outputs), plus a specific delay setting for each channel. The global delay setting can span the 14 nsec bunch spacing. The channel delays are intended to compensate for cable length variations and should generally be within +/- 1.5 nsec of each other. The delay for each chip can vary from 3.2ns to 14.8ns in 10ps increments.

BSM/FLM (Block A Only; Block B not used for BSM/FLM)

Offset	Contents
0	Block A, Global Delay 1
1	Block A, Global Delay 0
2	Block A, Card 3 Delay (chan 24 - 31)
3	Block A, Card 2 Delay (chan 16 - 23)
4	Block A, Card 1 Delay (chan 8 - 15)
5	Block A, Card 0 Delay (chan 0 – 7)

BPM (Block A for one species; Block B for the other)

Offset	Contents
0	Block A, Global Delay 1
1	Block A, Global Delay 0
2	Block A, Card 3 Delay
3	Block A, Card 2 Delay
4	Block A, Card 1 Delay
5	Block A, Card 0 Delay
8	Block B, Global Delay 1
9	Block B, Global Delay 0
A	Block B, Card 3 Delay
B	Block B, Card 2 Delay
C	Block B, Card 1 Delay
D	Block B, Card 0 Delay

A typical calibration scheme is to set the channel delays to mid-scale and adjust the global delay for optimal results looking at the sum of all four channels. Then increment or decrement the individual channel delays to optimize the results for each channel.

Auxiliary Board

The auxiliary board is not accessible from the DSP board. Instead, it contains a ColdFire ‘dimm’ CPU module that can access all of the registers and memory described in this document. Refer to the section on COLDFIRE PROGRAMMING for details.

Registers

DIRECT_ADR 0X10040000 (dec=268697600)

This register holds the 32-bit address used for XBUS vector operations when the vector equals #127. This register is programmed from the XBUS by specifying vector #126. Refer to the discussion of VECTOR ADDRESS TABLE. It is not generally accessed by the DSP.

DSP_RESET 0X10040001 (dec=268697601)

This register controls the /DSP_RESET pin on the DSP_CHIP. If data bit D0 is zero, the reset signal is asserted. This stops the DSP. When data bit D0 changes to a one, the DSP booting and configuration process begins.

When the board is initialized (power-up or front panel pushbutton), data bit D0 will be set to zero. This differs from the DIMM_RESET, which is set to one.

DATA_ACQ 0X10040002 (dec=268697602)

This register controls acquisition of data by the analog cards.

bit 0: ACQ_MODE (READ/WRITE)

Once the acquisition parameters have been initialized, setting the ACQ_MODE bit to ‘1’ will switch control of the analog card signals to the acquisition controller and start the collection of data. This bit must be cleared to ‘0’ before programmed readout of the analog boards can occur. It can be cleared at any time. If acquisition is in progress, it will be halted.

bit 1: ACQ_ACTIVE (READ ONLY)

This bit shows the status of data collection. A ‘1’ after setting ACQ_MODE indicates that a turn marker has been received and data is being collected. This bit will revert back to ‘0’ when acquisition is complete or when ACQ_MODE is cleared to ‘0’.

bit 2: ACQ_DONE (READ ONLY)

This bit shows the status of data collection. A ‘1’ after setting ACQ_MODE indicates that data collection is finished. This bit will revert back to ‘0’ when ACQ_MODE is cleared to ‘0’.

bit 3: ACQ_CONT (READ/WRITE)

This bit control single-shot vs. continuous mode data acquisition. Once the acquisition parameters have been initialized, setting the ACQ_CONT bit to ‘1’ at the same time that the ACQ_MODE bit is set to ‘1’ will cause the system to acquire data continuously. If the ACQ_CONT’ bit is later set to ‘0’ while ACQ_MODE is left at ‘1’, acquisition will continue until ACQ_TURN_REQ additional turns have been acquired (post-trigger mode).

If the ACQ_CONT bit is ‘0’ when the ACQ_MODE bit is set to ‘1’, then the system will acquire a single shot of data as controlled by the contents of ACQ_TURN_REQ.

For single-shot mode, write 0x01 to the DATA_ACQ register and monitor bit 2 for completion. For continuous data acquisition, write 0x09 to the DATA_ACQ register. Then if you want acquisition to stop immediately, write 0x00 to the DATA_ACQ register. If you want is to continue for ACQ_TURN_REQ additional turns, write 0x01 to the DATA_ACQ register.

bit 4: DCM_LOCKED (READ ONLY)

The LOCKED signal activates after the DCM has achieved lock. To achieve lock, the DCM may need to sample several thousand clock cycles. After the DCM achieves lock, the LOCKED signal goes high. To guarantee that the system clock is established prior to the device waking up, the DCM can delay the completion of the device configuration process until after the DCM locks. Until the LOCKED signal activates, the DCM output clocks are not valid and can exhibit glitches, spikes, or other spurious movement.

bit 7:5: DCM_STATUS (READ ONLY)

The 3 signals connect to the status register in the DCM. Bit 5 (STATUS[0]) indicates the overflow of the phase shift numerator and that the absolute delay range of the phase shift delay line is exceeded. Bit 6 (STATUS[1]) indicates the loss of the input clock, CLKIN, to the DCM. Bit 7 (STATUS[2]) indicates that CLKFX has stopped.

Under normal operating conditions, the data read from bits [7:4] should be 0x1.

RECV_ERR 0X10040003 (dec=268697603)

This register holds error information from the serial Xbus receiver. Its usefulness is questionable, since we would need to use the serial Xbus to read it.

GLOBAL_TURN_CNT 0X10040004 (dec=268697604)

This read-only register holds a 20-bit turn counter. The counter can be reset globally by a command from the control system. This allows the counters in every module to be synchronized. With 20 bits, the counter wraps around every 2.5 seconds. The size of the counter could be extended, but then it would not be compatible with the counter in the first generation DSP modules.

GLOBAL_TURN_DAT 0X10040005 (dec=268697605)

This read-only register holds the 27-bit data stream that is sent with the 24 MHz (25 MHz for ERL) system clock on every turn. The packing of data is:

bit 0:	hardware trigger 0
bit 1:	hardware trigger 1
bit 9..2:	command (bit 9=MSB, bit 2=LSB)
bit 18..10:	vertical phase data (bit 18=MSB, bit 10=LSB)
bit 27..19:	horizontal phase data (bit 27=MSB, bit 19=LSB)

A signal will be provided to the DSP every time GLOBAL_TURN_DAT is updated. The DSP may need to examine trigger bits or the command word, and it may need to store the phase data.

TOD_SECONDS 0X10040006 (dec=268697606)

This register holds a 17-bit counter that is incremented once per second. It can provide a time-of-day timestamp. The current time can be written to this register, and should be written at least once per day, as well as whenever power is cycled.

This register is currently decoupled from the GLOBAL_TURN_CNT register and does not change when the GLOBAL_TURN_CNT in all modules is synchronized.

SEMAPHORE 0X10040007 (dec=268697607)

This register is used to provide a semaphore to guard 'critical sections' for xbus vs. dsp access. When read, it will return its current state. The act of reading it will change its state to '1'. The act of writing to it will change its state to '0'. If a '0' is read, that indicates that the semaphore was free, and it is now owned by the reader. If a '1' is read, that indicates that the semaphore was already owned, and the reader should try again.

DIMM_RESET 0X10040008 (dec=268697608)

This register controls the /DIMM_RESET pin to the DIMM module on the auxiliary I/O board. If data bit D0 is zero, the reset signal is asserted. This stops the DIMM. When data bit D0 changes to a one, the DIMM booting and configuration process begins.

When the board is initialized (power-up or front panel pushbutton), data bit D0 will be set to one. This differs from the DSP_RESET, which is set to zero.

UNUSED 0X10040009 (dec=268697609)
UNUSED 0X1004000A (dec=268697610)

UNUSED	0X1004000B	(dec=268697611)
UNUSED	0X1004000C	(dec=268697612)
UNUSED	0X1004000D	(dec=268697613)
UNUSED	0X1004000E	(dec=268697614)
UNUSED	0X1004000F	(dec=268697615)

These registers may be defined in the future.

BUNCH_PATTERN_A	0X10040010	(dec=268697616)
BUNCH_PATTERN_B	0X10040011	(dec=268697617)
BUNCH_PATTERN_C	0X10040012	(dec=268697618)
BUNCH_PATTERN_D	0X10040013	(dec=268697619)
BUNCH_PATTERN_E	0X10040014	(dec=268697620)
BUNCH_PATTERN_F	0X10040015	(dec=268697621)

These 6 registers hold the bunch pattern.

CESR: The revolution time of CESR is divided into 183 slots that are spaced 14 nsec apart. To store the data for a particular slot, a '1' is programmed into the BUNCH_PATTERN register. No data is stored when the register value is '0'. The first 5 registers are 32-bit and the 6th is 23 bits.

ERL: The "revolution" time of the ERL is divided into 122 slots that are spaced 20 nsec apart. To store the data for a particular slot, a '1' is programmed into the BUNCH_PATTERN register. No data is stored when the register value is '0'. The first 3 registers are 32-bit and the 4th is 26 bits.

These registers must be loaded prior to collecting data. Their contents will be used to initialize a large shift register.

The LSB of BUNCH_PATTERN_A will control storage of the first slot after the turn marker. Higher bits control slots that occur later in time. Notice that the LSB of BUNCH_PATTERN_A ***does not*** correspond to "train 1, bunch 1". The user must start with a pattern that has "train 1 bunch 1" in the first location, then rotate that pattern until the 'train 1 bunch 1" bit matches the arrival time of "train 1 bunch 1" at a given detector, relative to the time that the turn marker arrives at the same detector. A timing calibration routine should be provided which can determine the appropriate shift at a given detector.

ADC_MEM_FIRST	0X10040016	(dec=268697622)

This register holds the initial memory address for an acquisition sequence. This address will be loaded into the address counter at the start of every acquisition sequence.

The address counter is a 19-bit counter that generates memory addresses. This register also holds 19 bits.

ACQ_TURN_REQ	0X10040017	(dec=268697623)

This register holds the requested number of turns to be acquired. This count will be loaded into the acquisition turn counter at the start of every acquisition block.

The acquisition turn counter is a 20-bit counter that controls how many turns of data are stored in memory. The counter will initially be loaded from ACQ_TURN_REQ. It will decrement every time a turn marker arrives (unless ACQ_SKIP_CNT is not zero). When the counter has decremented to zero, the acquisition sequence is finished.

Remember that more than one data point can be acquired per turn (up to 183 for CESR and 122 for ERL). To avoid memory address wrap-around, be sure that the product of ACQ_TURN_REQ time the number of '1's in the BUNCH_PATTERN is less than 512k.

ACQ_SKIP_CNT 0X10040018 (dec=268697624)

This function has not been implemented yet

This register controls how many turns are skipped over between turns that are stored in memory. It is an 8-bit register, allowing up to 255 turns to be skipped. It is initialized to zero, meaning that no turns are skipped. A skip count value of 1 means that every-other turn is stored. This is a different meaning than in the 1st generation of DSP-based BPMs.

TURN_MARK_DELAY 0X10040019 (dec=268697625)

This register controls the arrival time of the turn marker used by the acquisition logic. A '0' specifies 'normal' arrival time, while a '1' specifies 'delayed' arrival.

The turn marker is synchronous to the 24 MHz CESR clock (25 MHz ERL) extracted from the coax timing cable. The turn marker is used in logic that is synchronous to the ADC clock. However, the delay setting of the ADC clock can be anywhere in a 14 nsec. Window (20 nsec ERL), resulting in a possible situation where the setup and hold times of the turn marker relative to the ADC clock are not met. This register provides a way to shift the time that the setup and hold is not met by 7 nsec for CESR or 10 nsec for ERL. Some testing will need to be performed to decide what range of the global delay can be safely used with each setting of the TURN_MARK_DELAY.

ADC_CLOCK_CONTROL 0X1004001A (dec=268697626)

This register is present only on the BPM modules or other modules that use a 2-channel ADC card with independent clocking. It does not exist on the BSM/FLM modules, nor on other modules that use 8-channel ADC cards with ganged clocking.

This register is used to select which ADC clock drives the Xilinx electronics, and to set the clock phase for clocking the data from the ADCs into the ADC data registers and then writing the data to memory.

The timing card generates two ADC clocks, and the phase of each clock relative to the fixed incoming CESR clock can be set independently on the timing card. The data from each ADC is then clocked into a register. Finally, the data from both registers is written to memory. The timing of clocking the data into the registers and writing the data to memory can be adjusted in ¼ period intervals. The trick is to choose one of the ADC clocks and the appropriate setting for the register and memory timing so that all setup and hold times are met.

This register is divided into the following bit fields:

bits 1..0 = ADC0_REG_CLK:

Selects one of four delay settings relative to the chosen ADC clock for clocking data from ADC #0 into the data register. With a 72 MHz CESR sample rate, each increment in the settings of these bits represents an additional delay of 3.5 nsec. With a 50 MHz ERL sample rate, each increment in the settings of these bits represents an additional delay of 5.0 nsec.

bits 3..2 = ADC1_REG_CLK

Selects one of four delay settings relative to the chosen ADC clock for clocking data from ADC #1 into the data register. With a 72 MHz CESR sample rate, each increment in the settings of these bits represents an additional delay of 3.5 nsec. With a 50 MHz ERL sample rate, each increment in the settings of these bits represents an additional delay of 5.0 nsec.

bits 5..4 = MEM_WR_CLK

Selects one of four delay settings relative to the chosen ADC clock for writing data from both ADC data registers into the memory. With a 72 MHz CESR sample rate, each increment in the settings of these bits represents an additional delay of 3.5 nsec. With a 50 MHz ERL sample rate, each increment in the settings of these bits represents an additional delay of 5.0 nsec.

bits 6 = ADC_CLK_SEL

Chooses which ADC clock is the master clock for clocking data into the ADC data registers and writing it to memory. A value of '0' corresponds to ADC #0 and timing card block 'A'. A value of '1' corresponds to ADC #1 and timing card block 'B'.

Refer to the section on *Setting the ADC_CLOCK_CONTROL Register* for examples of how to set up this register.

RESERVED	0X1004001B	(dec=268697627)
RESERVED	0X1004001C	(dec=268697628)
RESERVED	0X1004001D	(dec=268697629)

ACQ_TURN_CNT	0X1004001E	(dec=268697630)

This is a read-only register that shows the current value of the acquisition turn counter. Since the turn counter counts backwards, the number of turns of data in memory can be calculated by subtracting the value of ACQ_TURN_CNT from the initial value specified in ACQ_TURN_REQ.

NEXT_MEM_ADR	0X1004001F	(dec=268697631)

This is a read-only register that shows the memory location where the next sample will be stored.

TEMPERATURE_CSR	0X10040020	(dec=268697632)

This register controls the readout of the AD7814 temperature chips. Writing any data to this register will initiate reading of the serial data from all six chips. Bit 0 of this register will be a '1' while serial data is being transferred from the temperature chips. Once the transfer is complete, bit 0 will change back to a '0'. This will indicate that the data is stable and can be read from the individual temperature registers.

The readout time from the AD7814 chip is about 2 usec. Since the DSP can access data more quickly than that, the DSP should check the status bit. Accesses from the XBUS are slow enough so that the data will be ready by the time one tries to access it.

ANA_0_TEMPERATURE	0X10040021	(dec=268697633)
ANA_1_TEMPERATURE	0X10040022	(dec=268697634)
ANA_2_TEMPERATURE	0X10040023	(dec=268697635)
ANA_3_TEMPERATURE	0X10040024	(dec=268697636)
TIMING_TEMPERATURE	0X10040025	(dec=268697637)
DIGITAL_TEMPERATURE	0X10040026	(dec=268697638)

These read-only registers contain the data read from the AD7814 temperature chips. The data is in units of 0.25 degrees centigrade. It is assumed that the temperature will never be below 32 degrees F, so this number will always be positive.

MODULE_TYPE	0X10040027	(dec=268697639)

MAJOR_REV	0X10040028	(dec=268697640)
MINOR_REV	0X10040029	(dec=268697641)

These are read-only registers that show the type of Xilinx code that is running on a module, as well as the major and minor revision numbers. For this document:

MODULE_TYPE
= 1 BSM
= 2 BPM
= 3 FLM
= 4 FLMA
= 5 ERL_BPM

MAJOR_REV = 5
MINOR_REV = 4

These values are defined in a ‘Constants.txt’ file that is included by the main Verilog program for each product type. They should correspond to notes in the file “VersionChanges.txt”

RESERVED	0X1004002A	(dec=268697642)
RESERVED	0X1004002B	(dec=268697643)
RESERVED	0X1004002C	(dec=268697644)
RESERVED	0X1004002D	(dec=268697645)
RESERVED	0X1004002E	(dec=268697646)
RESERVED	0X1004002F	(dec=268697647)

CUR_MON_STAT	0X10040030	(dec=268697648)
CUR_MON_WR_DAT	0X10040031	(dec=268697649)
CUR_MON_WR_ADR	0X10040032	(dec=268697650)
CUR_MON_RD_ADR	0X10040033	(dec=268697651)
CUR_MON_RD_DAT	0X10040034	(dec=268697652)

The current monitor board interface uses indirect addressing to access data within the current monitor chip. First, the program must read from the CUR_MON_STAT register to be sure that the serial link from the DSP board to the current monitor board is idle, and is not in a timeout error condition. A value of '0' (‘0x00000000’) indicates that the interface is ready to receive commands and/or data. A value of '1' (‘0x00000001’) indicates that the interface is busy. A value of '-32768' (‘0xffff8000’) indicates that a timeout has occurred.

If the interface is busy, any data written to the CUR_MON_WR_DAT, CUR_MON_WR_ADR, or CUR_MON_RD_ADR registers will be ignored. Data read from the CUR_MON_RD_DAT register will be undefined.

If a timeout has occurred, the only way to clear the error code and re-enable the interface is to write a value of ‘-1’ (0xffffffff) to the CUR_MON_STAT register. This will reset all of the logic on the DSP board side of the current monitor board interface. The timeout time is about 2.5 usec. A timeout is a serious condition that requires investigation.

Do not confuse the CUR_MON_STAT register with the CSR register on the current monitor board. The CUR_MON_STAT register is solely concerned with the communications link to the current monitor board.

To write data, the data is written to the CUR_MON_WR_DAT register. The destination address is then written to the CUR_MON_WR_ADR register. The serial transfer will then start and the BUSY bit in the CUR_MON_STAT register will be set. When the transfer is complete, the BUSY bit will be cleared.

To read data, the destination address is written to the CUR_MON_RD_ADR register. The serial transfer will then start and the BUSY bit in the CUR_MON_STAT register will be set. When the transfer is complete, the BUSY bit will be cleared. The data can then be read from the CUR_MON_RD_DAT register.

Any operations that write to the CUR_MON_WR_DAT, CUR_MON_WR_ADR, and CUR_MON_RD_ADR registers will be ignored if the interface is busy.

Addresses written to the CUR_MON_WR_ADR and CUR_MON_RD_ADR registers are 8-bit. All data written to the CUR_MON_WR_DAT register will be truncated at 16-bits. All data read from the CUR_MON_RD_DAT register will be 16-bit data that is sign-extended to 32-bits.

.

Setting the ADC_CLOCK_CONTROL Register

There are a few correct settings for the ADC_CLOCK_CONTROL register and the registers on the timing board; there are a great many incorrect settings. These guidelines will help you to do things correctly. They are divided into single ADC vs. dual ADC cases.

For all cases, set the 8 card delay registers on the timing board to a center value of 700. Then limit excursions to the range of 600 to 800. This will allow for +/- 1 nsec. adjustment for cable and card delay mismatches. If you go beyond this range then you will end up clocking data into a register or writing data to memory when the data is changing, resulting in invalid data.

Single ADC

Set the ADC_CLK_SEL bit (bit 6) to ‘0’ if you are controlling the conversion time with ‘Block A’ from the timing card. Set the ADC_CLK_SEL bit to ‘1’ if you are using ‘Block B’ from the timing card.

Set the ADC0_REG_CLK bits (bits 1..0) and/or ADC1_REG_CLK bits (bits 3..2) to binary ‘11’. This will cause the ADC data to be clocked into the register about 1.5 nsec before the ADC clock goes high. With the +/- 1 nsec range for the individual cards, the setup and hold times of the data register will always be met.

Set the MEM_WR_CLK bits (bits 5..4) to binary ‘00’.

The final result is to write the binary pattern ‘0001111’ for ‘Block A’ timing or ‘1001111’ for ‘Block B’ timing.

Dual ADC

To be written

Interrupts

Various signals in the Xilinx chip are connected to interrupt lines on the DSP. The DSP can be configured to actually generate interrupts when these signals are asserted, or it can simply examine the state of the signal. If interrupts are being generated, the lowest priority interrupt is IRQ0 and the highest is IRQ3. The state of the interrupt signal can be observed in the DSP’s ILAT register. If the interrupt is edge-sensitive, the signal observed in the ILAT register can be cleared by writing a ‘1’ to the corresponding bit in the ILATCL register. Level sensitive interrupts should be cleared by removing the interrupting condition. Edge vs. level sensitivity is determined by the DSP’s SQCTL register.

Enabling of interrupt generation is controlled by the DSP's IMASK and PMASK registers. Bit 60 of the IMASK register needs to be set as well to enable any hardware interrupts.

IRQ0

Not connected. Associated with bit 41 of the DSP's ILAT/IMASK/PMASK registers.

IRQ1

This interrupt is connected to the ACQ_DONE bit of the DATA_ACQ register. It will be asserted when data collection is finished. This interrupt is associated with bit 42 of the DSP's ILAT/IMASK/PMASK registers.

IRQ2

This interrupt is connected to the 'turn_mark' signal in the Xilinx chip. It will be asserted for 42 nsec when the turn marker arrives. A new value for GLOBAL_TURN_DAT is then available. This interrupt is associated with bit 43 of the DSP's ILAT/IMASK/PMASK registers.

IRQ3

Not connected. Associated with bit 44 of the DSP's ILAT/IMASK/PMASK registers.

Auxiliary I/O Card LEDs and Connectors

There are nine LEDs on the front of the auxiliary I/O card. Starting near the power connector and at the position closest to the circuit board, the LEDs indicate:

+4 Volt DC Power OK
+5 Volt DC Power OK
+12 Volt DC Power OK

Turn Marker Detected
-5 Volt DC Power OK
+2 Volt DC Power OK

DSP Activity Detected
Xbus Activity Detected
ColdFire (Ethernet) Activity Detected

Local Bus Masters

There are several devices which can be "bus master" on the local address and data bus. Arbitration logic within the XILINX chip, along with the /HBR and /HBG signals on the DSP, is used to determine which device has bus ownership at any instant. The potential bus masters are:

DSP
XBUS Controller
Ethernet ColdFire controller
FLASH Programmer

ColdFire Programming

Base address = 0x80000000

The ColdFire 'dimm' is a 1-byte oriented machine, while the BPM/BSM/FLM modules are 4-byte (32-bit word) oriented machines. The two LSBs of the 'dimm_adr' bus will specify the byte within a 4-byte word. The coldfire is "big-endian", so when a 32-bit word is moved over the 8-bit bus, the MSB goes with byte address '00' and the LSB goes with byte address '11'. The address map seen by the ColdFire is:

Adr	BinAdr	Register
00	0000xx	fix_adr32
04	0001xx	fix_dat32
08	0010xx	inc_adr32
0C	0011xx	inc_dat32
10	0100xx	fix_adr16
14	0101xx	fix_dat16
18	0110xx	inc_adr16
1C	0111xx	inc_dat16
20-3C		reserved

The programmer needs to write the address of the BPM/BSM/FLM location that he wants to access to one of the 4 "adr" registers. He then reads from or writes to the associated "dat" register to transfer data. The address to use is that of the 4-byte entity (use the same address that would be used by the XBUS or the DSP)

If the programmer is using a register with the "fix_dat" name, the BPM/BSM/FLM address will remain at the value that he last programmed. If the programmer is using a register with the "inc_dat" name, the BPM/BSM/FLM address will be incremented by 1 at the end of the operation. So to read from a series of consecutive BPM/BSM/FLM locations, the programmer would write the initial address to the "inc_adr" register, then repeatedly access the "inc_dat" register to get the data. To read over and over again from the same address, like for testing a status bit, the programmer would write the desired address to the "fix_adr" register, then repeatedly access the "fix_dat" register to get the data.

The registers that end with "32" are used to transfer 4-byte data to and from BPM/BSM/FLM locations. Registers that end with "16" are used to transfer the 2 lower bytes of a BPM/BSM/FLM location. Remember that ALL data in the BPM/BSM/FLM is 32-bit data, but if you are just reading raw ADC values, there is no need to waste bandwidth moving a bunch of zeros around.

The DIMM module needs to be configured for TCP/IP applications. The TCP/IP parameters are initialized with environment variables and then used by the 'rtems_nvram' function. The board must be configured by using appropriate values in the following set of commands:

```
setenv HOSTNAME klybpm08
setenv IPADDR0 192.168.7.118
setenv GATEWAY 192.168.7.1              ! use 172.17.0.1 for ERL subnet
setenv NETMASK 255.255.255.0            ! use 255.255.0.0 for ERL subnet
setenv SERVER 192.168.1.80              ! BOOTP server?
setenv NAMESERVER 128.84.47.200         ! or 128.84.46.26
setenv NTPSERVER 128.84.46.17           ! or 128.84.46.26 or 128.84.46.181
setenv NFSMOUNT lnx180c:/mnt/instr:/mnt/instr  ! Server:Path:MountPoint
```

And if you are supporting EPICS:

```
setenv CMDLINE /mnt/instr/epics/example/iocBoot/ioctest/st.cmd
setenv BOOTFILE /mnt/instr/epics/example/bin/RTEMS-uC5282/test.boot
```

To compile and download a program to the ColdFire, see the Twiki entry at:

https://wiki.lepp.cornell.edu/lepp/bin/view/CESR/Bunch/RTEMS

Accumulator Programming

ANALOG CARD 0		
ADR	0x0A000000	(dec=167772160)
CSR	0x0A000001	(dec=167772161)
LO_DAT	0x0A000002	(dec=167772162)
HI_DAT	0x0A000003	(dec=167772163)
ANALOG CARD 1		
ADR	0x0A400000	(dec=171966464)
CSR	0x0A400001	(dec=171966465)
LO_DAT	0x0A400002	(dec=171966466)
HI_DAT	0x0A400003	(dec=171966467)
ANALOG CARD 2		
ADR	0x0A800000	(dec=176160768)
CSR	0x0A800001	(dec=176160769)
LO_DAT	0x0A800002	(dec=176160770)
HI_DAT	0x0A800003	(dec=176160771)
ANALOG CARD 3		
ADR	0x0AC00000	(dec=180355072)
CSR	0x0AC00001	(dec=180355073)
LO_DAT	0x0AC00002	(dec=180355074)
HI_DAT	0x0AC00003	(dec=180355075)

Each accumulator card has 4 registers. The register names and the offset from the base address for each board are;

0 = ADR
1 = CSR
2 = LO_DAT
3 = HI_DAT

The accumulator boards use indirect addressing to access data within the accumulator chip. The address must be written to the ADR register, after which data can be written to the LO_DAT register or read from the LO_DAT and HI_DAT registers. All data transfers are 16-bit. All write operations are only 16 bits. Some read operations are 16 bit while others are 32 bits. For the 32 bit cases, the high 16 bits will be latched when the low 16 bits are read, and can be accessed from the HI_DAT register.

NOTE: AUTO-INCREMENT HAS NOT BEEN IMPLEMENTED YET. When using the auto-incrementing mode, one first writes the initial address to the ADR register, then repeatedly reads or writes the LO_DAT register. The address *always* gets incremented after access to the low data register. If 32-bit data is being read, the HI_DAT register should be read after the LO_DAT register.

Up to 4 accumulator boards can be used in a system. Each board has a 2 pin ID header that is used to set the board address. When writing to the ADR register, the write operation happens on **all** accumulator boards. All other operations (read ADR, read/write CSR, LO_DAT, HI_DAT) involve only one board. The two MSBs written to the ADR register must match the ID header to select a particular board. The remaining 14 bits in the ADR register are use to access some resource on the selected board. This results in a 16k address space per board.

The accumulator chip can be read from or written to while data is being collected in the ADC memory. Readout of the ADC memory or the accumulator can be interspersed.

CSR offset = 1

This register controls or monitors overall operation of the accumulator chip. It is a 16-bit register that can be written or read.

bit 0 = Nreset: This bit will power up low (asserted). It must have a '1' written to it to take the chip out of the ‘reset’ mode.

bit 1 = Nsetup: This bit will power up low (asserted). It must have a '1' written to it to take the chip out of the ‘setup’ mode.

The intended use of these bits is that they will initially both be zero. The programmer will set the ‘Nreset’ bit to one and then proceed to configure the chip by programming the lookup table and setting the various registers. When the programmer is ready to start collecting data, the ‘Nsetup’ bit will also be set to one. This causes all of the various state machines and counters to start together.

ACCUMULATOR ADDRESS MAP

Block 0: 1 sector of 4096 addresses
Decode ADR[13:12] Pass ADR[11:0]

Start	End	Size	Function
0x0000 {dec=0}	0x0fff {dec=4095)	4096	Hit Lookup Table (Read/Write)

Block 0 contains the 4k-by-2 bit Hit Lookup table. It is implemented as a dual-port memory, where one port provides read/write access for initializing and verifying the data and the other port is used to determine how many photons have hit the detector based on a combination of high and low threshold signal comparisons. When writing to it, data bits [1..0] contain the data to be stored. Data bits [15..2] are discarded. When reading, data bits [1..0] will contain the stored value. Data bits [15..2] will always read as zeroes.

The address used for looking up the hit count is derived from 6 ‘lo_hit’ channels (address bits [11..6]) and 6 ‘hi_hit’ channels (address bits [5..0]). If one wishes to only use the first 6 'hi_hit' channels, then the lookup table only needs to have the first 64 locations programmed. The ‘Hit Mask Register’ would be programmed as 0x007d. Refer to the discussion about the Hit Mask Register later in this document to see the details about the mapping between ADC channels and lookup table address bits.

Block 1: 16 sectors of 256 addresses
Decode ADR[13:8] Pass ADR[7:0]

Start	End	Size	Function
0x1000 {dec=4096}	0x1fff {dec=8191}	4096	16-bit R/W Control Registers

0x1000	{dec=4096}	ADC 0 Pedestal
0x1001	{dec=4097}	ADC 0 Low Threshold
0x1002	{dec=4098}	ADC 0 High Threshold
0x1003	{dec=4099}	Unused
0x1004	{dec=4100}	ADC 1 Pedestal
0x1005	{dec=4101}	ADC 1 Low Threshold
0x1006	{dec=4102}	ADC 1 High Threshold
0x1007	{dec=4103}	Unused

0x1008	{dec=4104}	ADC 2 Pedestal
0x1009	{dec=4105}	ADC 2 Low Threshold
0x100A	{dec=4106}	ADC 2 High Threshold
0x100B	{dec=4107}	Unused
0x100C	{dec=4108}	ADC 3 Pedestal
0x100D	{dec=4109}	ADC 3 Low Threshold
0x100E	{dec=4110}	ADC 3 High Threshold
0x100F	{dec=4111}	Unused
0x1010	{dec=4112}	ADC 4 Pedestal
0x1011	{dec=4113}	ADC 4 Low Threshold
0x1012	{dec=4114}	ADC 4 High Threshold
0x1013	{dec=4115}	Unused
0x1014	{dec=4116}	ADC 5 Pedestal
0x1015	{dec=4117}	ADC 5 Low Threshold
0x1016	{dec=4118}	ADC 5 High Threshold
0x1017	{dec=4119}	Unused
0x1018	{dec=4120}	ADC 6 Pedestal
0x1019	{dec=4121}	ADC 6 Low Threshold
0x101A	{dec=4122}	ADC 6 High Threshold
0x101B	{dec=4123}	Unused
0x101C	{dec=4124}	ADC 7 Pedestal
0x101D	{dec=4125}	ADC 7 Low Threshold
0x101E	{dec=4126}	ADC 7 High Threshold
0x101F	{dec=4127}	Unused

0x1100	{dec=4352}	Bunch Pattern Register, (bunch_pat[15:0])
0x1101	{dec=4353}	Bunch Pattern Register, (bunch_pat[31:16])
0x1102	{dec=4354}	Bunch Pattern Register, (bunch_pat[47:32])
0x1103	{dec=4355}	Bunch Pattern Register, (bunch_pat[63:48])
0x1104	{dec=4356}	Bunch Pattern Register, (bunch_pat[79:64])
0x1105	{dec=4357}	Bunch Pattern Register, (bunch_pat[95:80])
0x1106	{dec=4358}	Bunch Pattern Register, (bunch_pat[111:96])
0x1107	{dec=4359}	Bunch Pattern Register, (bunch_pat[127:112])
0x1108	{dec=4360}	Bunch Pattern Register, (bunch_pat[143:128])
0x1109	{dec=4361}	Bunch Pattern Register, (bunch_pat[159:144])
0x110A	{dec=4362}	Bunch Pattern Register, (bunch_pat[175:160])
0x110B	{dec=4363}	Bunch Pattern Register, (bunch_pat[191:176])

0x1200	{dec=4608}	Hit Mask Register
0x1201	{dec=4609}	Force Hi Hit Register

0x1300	{dec=4864}	Bunch Rate Low Period Register
0x1301	{dec=4865}	Bunch Rate High Period Register

0x1400	{dec=5120}	Fast Global Rate Low Period Register
0x1401	{dec=5121}	Fast Global Rate High Period Register

0x1500	{dec=5376}	Slow Global Rate Low Period Register
0x1501	{dec=5377}	Slow Global Rate High Period Register

0x1600	{dec=5632}	Geo Global Rate Low Period Register
0x1601	{dec=5633}	Geo Global Rate High Period Register

0x1700	{dec=5888}	Geo Bunch Rate Low Period Register
0x1701	{dec=5889}	Geo Bunch Rate High Period Register

Block 1 contains 16-bit control registers. Each register can be read or written.

NOTE: The data from the ADCs is first inverted so that all numbers are positive, a small amplitude is near zero, and a large amplitude is near 65000. The 12-bit ADC data is left-shifted to occupy bits [15:4]. Pedestals and thresholds are treated similarly.

ADC Pedestal registers contain a number that is subtracted from the ADC data. It is a 16-bit unsigned number, but only the 12 MSBs are used. If the result would be less than zero, it is clipped at zero. All bits are cleared to '0' when the 'Nreset' bit in the CSR is zero.

ADC Low Threshold and ADC High Threshold registers contain numbers that the pedestal corrected ADC data is compared with. It is a 16-bit unsigned number, but only the 12 MSBs are used. The comparison is "greater than or equal to". All bits are cleared to '0' when the 'Nreset' bit in the CSR is zero.

Bunch Pattern Registers are used to enable or disable processing of every 7^{th} RF bucket (14 nsec.). It is a 16-bit number, but the last register only uses the 7 LSBs. There are 183 possible bunches specified in bunch_pat[182:0]. A '1' in a bit position enables processing of that bunch while a '0' disables it. All bits are preset to '1' when the 'Nreset' bit in the CSR is zero.

The Hit Mask Register is used to specify which ADCs are producing valid data to be used in forming the address for the Hit Lookup Table and for counting geometric channel rates. Only ADC channels 1 thru 6 are used to generate the address. The other 2 channels (0 and 7) have signals that are not part of the lookup process. They are still part of the 'hit_mask'. Bits [7..0] enable use of the High Threshold comparison while bits [15..8] enable use of the Low Threshold comparison. All bits are preset to '1' when the 'Nreset' bit in the CSR is zero. The logic for creating the lookup table address is:

```
always @(posedge clk) begin
        if (bunch_en) begin
                hit_lkup_adr[0]  <= hi_hit[3] & hit_mask[3];
                hit_lkup_adr[1]  <= hi_hit[6] & hit_mask[6];
                hit_lkup_adr[2]  <= hi_hit[2] & hit_mask[2];
                hit_lkup_adr[3]  <= hi_hit[5] & hit_mask[5];
                hit_lkup_adr[4]  <= hi_hit[1] & hit_mask[1];
                hit_lkup_adr[5]  <= hi_hit[4] & hit_mask[4];

                hit_lkup_adr[6]  <= lo_hit[3] & hit_mask[11];
                hit_lkup_adr[7]  <= lo_hit[6] & hit_mask[14];
                hit_lkup_adr[8]  <= lo_hit[2] & hit_mask[10];
                hit_lkup_adr[9]  <= lo_hit[5] & hit_mask[13];
                hit_lkup_adr[10] <= lo_hit[1] & hit_mask[9];
                hit_lkup_adr[11] <= lo_hit[4] & hit_mask[12];
        end
        else
                hit_lkup_adr <= 12'h000;
end
```

The Force Hi Hit Register allows one to always force the result of the High Threshold comparison to be true ONLY for Geo Global Rate data and Geo Channel Rate data. It is useful for testing. Bits [7..0] force the corresponding channel. Bits [15..8] are unused. All bits are cleared to '0' when the 'Nreset' bit in the CSR is zero.

The various Rate Period Registers control the duration of the sample period for a rate measurement. The period is specified in units of 'cesr turn time' (2.56 usec). The underlying

counters are 22-bit counters, so the maximum setting is 2**22 x 2.56 usec = 10.7 seconds. The 16 LSBs are specified in the various Low Period Registers, while the 6 MSBs are specified in the various High Period Registers. The number of turns written to the rate registers must be 1 less than the number of turns in the desired period. To get a period of 1 second (390320 turns) program a value of '5' in the High Period Register and a value of '62639' in the Low Period Register. [390320 – 1 = 390319 = 0x5f4af, therefore the High Period Register = 0x05 = 5 and the Low Period Register = 0xf4af = 62639]

Block 2: 16 sectors of 256 addresses
Decode ADR[13:8] Pass ADR[7:0]

Start	End	Size	Function
0x2000 {dec=8192}	0x2fff {dec=12287}	4096	32-bit Rate Registers (Read Only)

0x2000	{dec=8192}	183	Bunch Rate Data
0x2100	{dec=8448}	1	Fast Global Rate Data
0x2200	{dec=8704}	1	Slow Global Rate Data
0x2300	{dec=8960}	8	Geo Global Rate Data (8 chan, all bunches)
0x2800	{dec=10240}	183	Geo Channel 0 Rate Data
0x2900	{dec=10496}	183	Geo Channel 1 Rate Data
0x2A00	{dec=10752}	183	Geo Channel 2 Rate Data
0x2B00	{dec=11008}	183	Geo Channel 3 Rate Data
0x2C00	{dec=11264}	183	Geo Channel 4 Rate Data
0x2D00	{dec=11520}	183	Geo Channel 5 Rate Data
0x2E00	{dec=11776}	183	Geo Channel 6 Rate Data
0x2F00	{dec=12032}	183	Geo Channel 7 Rate Data

Block 2 contains 32-bit Rate Data registers. Each register is read-only. To read any given Rate Data register, setup the ADR register with the address of the desired Rate Data register. Read from the LO_DAT register to get the 16 LSBs. Then read from the HI_DAT register to get the 16 MSBs. The resulting 32-bit number contains the count in the lower 20 bits and a sample tag in the upper 12 bits. Analysis code will need to separate these two numbers.

The 12-bit sample tag is incremented at the end of each sample period. The value will wrap around to zero when it exceeds 4095. Each of the 4 Rate Period Registers drives its own sample tag, so they do not necessarily all count in lockstep order. The sample tags will be reset to zero when either 'Nreset' or 'Nsetup' in the CSR is a zero.

Block 3: 1 sector of 4096 addresses

Start	End	Size	Function
0x3000 {12288}	0x3fff {16383}	4096	Reserved

Nothing is implemented in block 3.

SAMPLE COMMAND FILES TO ACCESS THE ACCUMULATOR BOARD

[cesr.cesrbpm.6048_113]accum_csr_wr.com

```
$! Write to the accumulator CSR and then read back (ADC card 0 only)
$! do accum_csr_wr  accum_csr_dat
$! accum_csr_dat in decimal
$! crs 12/22/05
```

```
$!
$ wo = "write sys$output"
$! set up the address register
$ fff vxputn cbpm adr tst 1 1 167772160
$! write to the csr register
$ fff vxputn cbpm adr tst 1 1 167772161
$ fff vxputn cbpm dat tst 1 1 'p1
$! read the CSR register
$ fff vxgetn cbpm dat tst 1 1
```

[cesr.cesrbpm.6048_113]accum_wr.com

```
$! Write a 16-bit word to an accumulator location (ADC card 0 only)
$! do accum_wr accum_adr accum_dat
$! accum_adr and accum_dat in decimal
$! crs 12/22/05
$!
$ wo = "write sys$output"
$! set up the address register
$ fff vxputn cbpm adr tst 1 1 167772160
$ fff vxputn cbpm dat tst 1 1 'p1
$! write to the lo_dat register
$ fff vxputn cbpm adr tst 1 1 167772162
$ fff vxputn cbpm dat tst 1 1 'p2
```

[cesr.cesrbpm.6048_113]accum_rd.com

```
$! Read a 16-bit word from an accumulator location (ADC card 0 only)
$! do accum_rd accum_adr
$! accum_adr in decimal
$! crs 12/22/05
$!
$ wo = "write sys$output"
$! set up the address register
$ fff vxputn cbpm adr tst 1 1 167772160
$ fff vxputn cbpm dat tst 1 1 'p1
$! read from the lo_dat register
$ fff vxputn cbpm adr tst 1 1 167772162
$ fff vxgetn cbpm dat tst 1 1
```

[cesr.cesrbpm.6048_113]accum_rd2.com

```
$! Read two 16-bit words from an accumulator address (ADC card 0 only)
$! do accum_rd2 accum_adr
$! accum_adr in decimal
$! crs 12/22/05
$!
$ wo = "write sys$output"
$! set up the address register
$ fff vxputn cbpm adr tst 1 1 167772160
$ fff vxputn cbpm dat tst 1 1 'p1
$loop:
$! read from the lo_dat register
$ fff vxputn cbpm adr tst 1 1 167772162
$ fff vxgetn cbpm dat tst 1 1
```

```
$! read from the hi_dat register
$ fff vxputn cbpm adr tst 1 1 167772163
$ fff vxgetn cbpm dat tst 1 1
$ goto loop
```

%%

BSM Current Monitor Programming

The amplifiers on the BSM ADC boards have an input bias current between 20 and 40 uA. This corresponds to a signal between 1 and 2 mV at the input of the Current Monitor Board ADC which has a range of +/- 50mV. This 1 to 2 mV is the DC offset (temperature drift ~ 0.1% per degree). The signal we want to measure is 1 to 10 uA, or 0.05 to 0.5 mV at the input of the Current Monitor Board ADC (a fraction of the pedestal value). There is no specific hardware limit on the current signal, however, only a broken channel should saturate the ADC. It is unlikely that over/under range testing is required.

Full scale:	+/- 50 mV = +/- 1 mA = 16 bits (1 LSB = 1.526 uV = 30.52 nA)
Pedestal:	+/- 1 to 2 mV = +/- 20 to 40 uA (ADC dat = 655 to 1311)
Signal:	0.05 to 0.5 mV = 1 to 10 uA (ADC dat = 32 to 328)

Current Monitor Address Map

Current RAM	0x00-0x1f	(dec=0-31)
Pedestal RAM	0x20-0x3f	(dec=32-63)
Limit RAM	0x40-0x5f	(dec=64-95)
Total Current Limit Setting	0x60	(dec=96)
DAC0 Setting	0x61	(dec=97)
DAC1 Setting	0x62	(dec=98)
HV Readback	0x63	(dec=99)
Shutter Time Setting/Readback	0x64	(dec=100)
CSR (Control and Status Register)	0x65	(dec=101)
Shutter OverCurrent Cycles	0x66	(dec=102)
High Voltage OverCurrent Cycles	0x67	(dec=103)
Total Current	0x68	(dec=104)
Supervisor State Machine	0x69	(dec=105)

Current RAM 0x00-0x1f (dec=0-31)

This is a 16-bit by 32 dual-port memory. If the PROGRAM/RUN bit of the CSR is at the PROGRAM value, it contains the raw ADC value of the current for each of the 32 channels. If the PROGRAM/RUN bit of the CSR is at the RUN value, it contains the corrected (pedestal subtracted) value. The data is 16-bit 2's-complement format. Data is written to this memory by the hardware and is read-only from the DSP. If a High Voltage trip occurs, the data in this memory will be frozen at the values that caused the trip.

Pedestal RAM 0x20-0x3f (dec=32-63)

This is a 16-bit by 32 dual-port memory. It contains the pedestal (or zero current) value to be subtracted from the measured current for each of the 32 channels. The data is 16-bit 2's-complement format. Data is written to this memory by the DSP. If the PROGRAM/RUN bit of the CSR is in the PROGRAM state, then the DSP can read back the memory contents. If the bit is in the RUN state, then the hardware has read access and the DSP cannot read the data.

Limit RAM 0x40-0x5f (dec=64-95)

This is a 16-bit by 32 dual-port memory. It contains the limit value to be compared with the measured and corrected current for each of the 32 channels. The data is 16-bit 2's-complement format. Data is written to this memory by the DSP. If the PROGRAM/RUN bit of the CSR is in the PROGRAM state, then the DSP can read back the memory contents. If the bit is in the RUN state, then the hardware has read access and the DSP cannot read the data.

Total Current Limit Setting 0x60 (dec=96)

This is a 16-bit register. It contains the limit value to be compared with the total measured and corrected current for all of the 32 channels. The data is 16-bit 2's-complement format. Data can be written or read at any time by the DSP.

DAC0 Setting 0x61 (dec=97)
DAC1 Setting 0x62 (dec=98)

These are 14-bit unsigned registers, using bits [13..0]. They drive a Linear Technologies LTC1658 DAC, which provides a 0 to 4.5 volt output. DAC0 is used to set the amplitude of the high voltage supply. DAC1 is currently unused. When read from, the last value that was written to the DACs is returned.

If a High Voltage trip occurs, the DAC0 register will be set to zero and that value will repeatedly be written to the DAC. Readback of the DAC0 Setting register will give a value of zero. The PROGRAM/RUN bit of the CSR needs to be set back to the PROGRAM state before new data that is written to the DAC0 register will take effect.

HV Readback 0x63 (dec=99)

This is a 16-bit register. It contains the value measured by the HighVoltage ADC. There is no pedestal correction. The data is 16-bit 2's-complement format. Data can be read at any time by the DSP. The value in the HV register is updated approximately 6.8 times per second.

Shutter Time Register 0x64 (dec=100)

This register provides a timer for BSM shutter opening. When written to, the input is a 16-bit unsigned number that specifies the number of 10-turn periods that the shutter is open. A value of 1000 will cause the shutter to be open for 10000 turns (actually for 10,132 with roundup error). The 10-turn counter is running asynchronously to the loading of data, so it is possible that the counter will decrement almost immediately after the data is loaded. This would give a runt period of less than 10 turns. An input of "8" would give between 70 and 80 turns. The maximum time that the shutter can be open is 655k turns. When read from, the remaining time in the counter will be returned.

The shutter can be closed immediately by writing a value of zero. The shutter will also be closed when the overcurrent detector finds an overcurrent condition.

Since this register uses an unsigned 16-bit number, a negative number will be read back if the current value is between 32,768 and 65,535. This is due to the sign bit extension that the hardware supplies. Values of 32,767 or less will be read correctly.

CSR (Control and Status Register) 0x65 (dec=101)

This register is divided into the following bit fields:

bit 0 = PROGRAM/RUN:

This read/write bit controls the mode of operation. If the bit is a '0', the system is in the PROGRAM mode. If the bit is a '1', the system is in the RUN mode.

bit 5:1 = RESERVED:

These read/write bits are reserved for future use. Whatever is written to them will be read back.

bit 6 = SHUTTER_OPEN:

This read-only bit shows the state of the control line that opens the shutter. If the bit is ’0’ then the shutter is supposed to be closed. If the bit is ‘1’, then the shutter is supposed to be open.

bit 7 = SHUTTER_OVERCURRENT:

This read-only bit indicates if the overcurrent protection circuit closed the shutter. If the bit is ’0’ then no overcurrent situation has been detected. If the bit is ‘1’, then an overcurrent situation has been detected and the shutter has been closed.

Once an overcurrent situation has occurred, the PROGRAM/RUN bit must be set back to the PROGRAM state to clear the error, and then to the RUN state to enable the protection.

bit 8 = SHUTTER_OVERCURRENT_CAUSE:

This read-only bit what type of overcurrent situation closed the shutter. If the bit is ’0’, then the shutter was closed due to an overcurrent situation on one of the channels. If the bit is ‘1’, then the shutter was closed due to an overcurrent situation on the total sum of all the channels.

bit 9 = HIGH_VOLTAGE_OVERCURRENT:

This read-only bit indicates if the overcurrent protection circuit turned off the High Voltage by setting the DAC to zero. If the bit is ’0’ then no overcurrent situation has been detected. If the bit is ‘1’, then an overcurrent situation has been detected and the High Voltage has been turned off.

Once an overcurrent situation has occurred, the PROGRAM/RUN bit must be set back to the PROGRAM state to clear the error and enable writing a new value to the DAC0 register. The PROGRAM/RUN bit must then be set to the RUN state to enable the protection.

bit 10 = HIGH_VOLTAGE_OVERCURRENT_CAUSE:

This read-only bit what type of overcurrent situation killed the High Voltage. If the bit is ’0’, then the voltage was turned off due to an overcurrent situation on one of the channels. If the bit is ‘1’, then the voltage was turned off due to an overcurrent situation on the total sum of all the channels.

bits 15:11 = OVERCURRENT_CHANNEL_NUM:

These 5 read-only bits contain the channel number of the ADC that caused a SHUTTER_OVERCURRENT trip. The data is only valid if bit 7 is a ‘1’. If bit 8 is a ‘0’, then this is the first channel that had too much current. If bit 8 is a ‘1’, then this is the channel that caused the accumulated total current to exceed the value of the Total Current Limit Setting.

Shutter OverCurrent Cycles 0x66 (dec=102)

This read/write register is used to specify how many consecutive cycles of an overcurrent situation must occur before a SHUTTER_OVERCURRENT trip occurs and causes the shutter to close. Each cycle takes about 140 milliseconds. The maximum value is 255 cycles.

High Voltage OverCurrent Cycles 0x67 (dec=103)

This read/write register is used to specify how many consecutive cycles of an overcurrent situation must occur before a HIGH_VOLTAGE_OVERCURRENT trip occurs and causes the High Voltage to be turned off. Each cycle takes about 140 milliseconds. The maximum value is 255 cycles.

Generally one would program the High Voltage OverCurrent Cycles register with a value that is larger than the value programmed in the Shutter OverCurrent Cycles register. That will cause the shutter to close first, and the high voltage would be turned off later if the overcurrent situation persisted.

Total Current 0x68 (dec=104)

This read-only register contains the total current of all 32 channels. If the PROGRAM/RUN bit of the CSR is at the PROGRAM value, it contains the sum of the raw ADC value of the current for all of the 32 channels. If the PROGRAM/RUN bit of the CSR is at the RUN value, it contains the sum of the corrected (pedestal subtracted) values. The data is 16-bit 2's-complement format. If a High Voltage trip occurs, the data in this register will be frozen at the values that caused the trip.

This register is subject to rollover and incorrect data in the event of very large current values. If the average channel has a current reading of more than +/- 1000 units, the data in this register would be suspect.

Supervisor State Machine 0x69 (dec=105)

This read-only register provides access to the supervisor state machine. It shows the level of each state bit. The correlation between bits in this register and the state of the supervisor is:

```
supv_state[0] = (mon_sm == init);
supv_state[1] = (mon_sm == read1);
supv_state[2] = (mon_sm == read2);
supv_state[3] = (mon_sm == wait1);
supv_state[4] = (mon_sm == wait2);
supv_state[5] = (mon_sm == check1);
supv_state[6] = (mon_sm == check2);
supv_state[7] = (mon_sm == check3);
supv_state[8] = (mon_sm == check4);
supv_state[9] = (mon_sm == check5);
supv_state[10] = run_mode;
supv_state[11] = kill_hv_ff;
```

DSP Wait State Programming

The DSP can address 3 regions of external memory. They are called "MS0", "MS1", and "MSH". The linker "LDF" file defines the boundaries of each region. Each region can have a specific number of wait states for DSP operations. Reading and writing must both use the same number of wait states. A logic analyzer was used to examine all valid operations. The peripherals that the DSP can access and the minimum number of wait states are:

Region "/MS0"

rd ANA_MEM	min waitstates = 1
wr ANA_MEM	min waitstates = 0
wr ANA_REG	min waitstates = 0

Region "/MS1"

rd SRAM	min waitstates = 1
wr SRAM	min waitstates = 1
rd FLASH	min waitstates = 4 !!! TOO SLOW
wr FLASH	min waitstates = 10 !!! TOO SLOW

Region “/MSH”

rd XIL_MEM	min waitstates = 3
wr XIL_MEM	min waitstates = 0
rd XIL_REG	min waitstates = 2
wr XIL_REG	min waitstates = 1
wr TIM	min waitstates = 1

The DSP initially uses 3 wait states for all operations. Without an external circuit to negate the “ACK” signal on the DSP and add wait states, 3 wait states is the maximum. As a result, the DSP cannot access the FLASH memory from a program. It can access the FLASH correctly when booting, because it uses special internal logic that inserts 16 wait states. The number of wait states can be changed by writing to the SYSCON register. The address of the SYSCON register is defined in “defts101.h”. A snippet of code that changes the number of waitstates for “MS0” and “MS1” from 3 to 1 is:

```
#include <defts101.h>

/* Set up the SYSCON register */
/* Configure 1 wait state for MS0 and MS1, 3 wait states for MSH */
/* Configure slow protocol with idle cycles for all segments. */
/* ‘OR’ of all the bits yields 0x000278E3 */

int *syscon_reg = (int *)SYSCON_LOC;

*syscon_reg =
SYSCON_MSH_SLOW | SYSCON_MSH_PIPE1 | SYSCON_MSH_WT3 | SYSCON_MSH_IDLE |
SYSCON_MS1_SLOW | SYSCON_MS1_PIPE1 | SYSCON_MS1_WT1 | SYSCON_MS1_IDLE |
SYSCON_MS0_SLOW | SYSCON_MS0_PIPE1 | SYSCON_MS0_WT1 | SYSCON_MS0_IDLE;
```