Introduction

The CFT Axial AFE board is connected to the rest of the D0 experiment by a MIL-STD 1553 interface. In this protocol, each CFT board looks like one **Remote Terminal** (RT), with up to 30 registers. Each of the 30 registers should be viewed as a FIFO up to 32 words deep. Whenever a particular register is accessed, it is read from one to 32 times by the host software. Each register is sixteen bits wide.

Jamieson Olsen has written a couple of notes describing the implementation of the 1553 interface; this document attempts to look at the non-1553 side of his logic to show how the various sections of the AFE map out. A full discussion of the microcontroller command structure and a breakdown of how the commands use the memory is within.

Board Architecture – Hardware Perspective

The AFE has a small microcontroller – a PIC16F877A – whose basic function is to provide a simplified interface to the various DAC and ADC channels on the board. A secondary function of this microcontroller is to allow each AFE which is installed as the Right-Hand board in a CFT cassette to act as a cassette closed-loop temperature control system. A tertiary function is to provide a control mechanism for sequencing power application to the SIFT and SVX chips on the CFT board. This microcontroller is attached to the main board data bus via a dual-port memory. The dual-port memory allows the 1553 interface to access the microcontroller data in 16-bit chunks, but allows the microcontroller to use the more convenient byte-wide architecture on its side. Further, the dual-port decouples the timing of the two systems. Figure 1 shows a quick sketch of the board architecture, showing the microcontroller and its interface at the top of the picture.

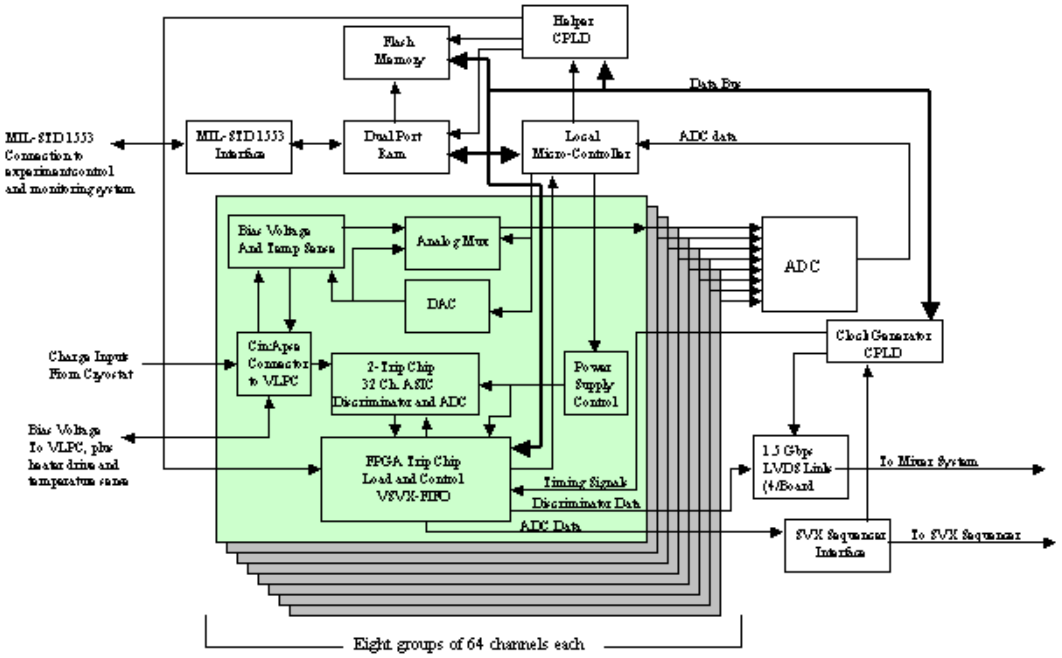

Figure 1Board Architecture – Software Perspective

The AFE contains numerous functional blocks, but many of these are only indirectly connected to the software. Thus, the software perspective of the board is very different from the schematic's block-level view. As seen by the software, the CFT Axial card looks like a base address register and a single I/O port.

Register Address	Register Name	Functional Description	Depth on reads	Depth on writes
0x10	BASE_ADDR	Register which contains base address of window into board memory space, accessed through register 0x11	One word	One word
0x11	DATA_PORT	I/O port through which 1553 accesses up to 32 locations of board dual-port RAM	Up to 32	Up to 32

Table 1

The 1553 software loads the BASE_ADDR register with the initial address within the dual-ported RAM that is to be modified, and then writes up to 32 words to the DATA_PORT register. The 1553 interface keeps a local count of how many words are written or read and drives the internal address bus appropriately. The address written to BASE_ADDR is the first address in the dual-port RAM which will be read or written by the access to the DATA_PORT; if the 1553 transaction calls for more than one word to be transferred, the internal address applied to the dual-port RAM is internally incremented with each word transferred through the DATA_PORT such that a block transfer is effected.

If external software wishes to read or write a block of more than 32 contiguous words of data to the dual-port RAM, the internal address counter should be preserved between accesses to the DATA_PORT, so simply setting the BASE_ADDR once and then accessing the DATA_PORT multiple times should suffice.[1]

Internal Memory Map

The AFE actually contains 2K X 16 bits of dual-port RAM, which the on-board microcontroller views as 4K X 8. Each table of addresses in the following sections show what the various locations do and gives the address(es) as viewed from both ports of the dual-port RAM. Every 16-bit word in 1553 space maps to two adjacent bytes in microcontroller space. The 'even' byte maps to bits 7..0 of the 16-bit word, with the next-higher 'odd' byte mapping to bits 15..8 of the 16-bit word.

For example: Address 0x57 in 1553 space is the 'command flag' location. From the microcontroller's viewpoint, that 16-bit value is two eight-bit values at addresses 0xAE and 0xAF. Location 0xAE corresponds to bits 7..0 of the 16-bit word, and location 0xAF corresponds to bits 15..8 of the 16-bit word.

The memory map given below is broken in to segments based upon the general set of functions associated with that block of the dual-port RAM space. For each segment, detailed descriptions of each location are given. ***Any location listed as reserved or unused must not be written to or read from by external software (e.g., MIL-STD 1553 or event monitoring).***

Additions or modifications to the memory map must be approved by the keeper of the AFE microcontroller code, currently

John T. Anderson

Fermilab D-Zero project
P.O. Box 500 M/S 352
Batavia, Il 60510
(630) 840-8885

email: janderson@fnal.gov

The latest version of this document is available on the Internet at
http://d0server1.fnal.gov/users/janderson/Public_Eng_Notes/default.html

[1] This is the way the logic should work, but in practice all engineering tests have explicitly set the BASE_ADDR before accessing the DATA_PORT.

Overview of Memory Map

The sections to follow will give exact addresses. This table shows the general organization of the dual-port RAM as a quick reference. Unused sections are greyed out, but do *not* assume they are free for use!

Address(es) as programmed from microcontroller	Address(es) as programmed from 1553	Function
0x0000 – 0x00BF	0x0000 – 0x005F	Microcontroller general control and command queue
0x00C0 – 0x00FF	0x0060 – 0x007F	Currently unused.
0x0100 – 0x016F	0x0080 – 0x00B7	A/D calibration, manual A/D conversion data area
0x0170 – 0x01FF	0x00B8 – 0x00FF	Currently unused.
0x0200 – 0x02A1	0x0100 – 0x0150	Multi-chip Module (MCM) control
0x02A2 – 0x02FF	0x0151 – 0x017F	Currently unused.
0x0300 – 0x0306	0x0180 – 0x0182	Virtual SVX (VSVX) control
0x0307 – 0x037F	0x0183 – 0x01BF	Currently unused.
0x0380 – 0x0393	0x01C0 – 0x01C9	LVDS control and status
0x0394 – 0x03FF	0x01CA – 0x01FF	Currently unused.
0x0400 – 0x04FF	0x0200 – 0x027F	Cryostat control loop parameters and status
0x0500 – 0x057F	0x0280 – 0x02BF	Reserved for engineering DAC tests
0x0580 – 0x05FF	0x02C0 – 0x02FF	Currently unused.
0x0600 – 0x0603	0x0300 – 0x0301	Reserved for engineering clock control tests.
0x0604 – 0x061F	0x0302 – 0x030F	Currently unused.
0x0620 – 0x06FF	0x0310 – 0x037F	Reserved for use with AFE Test Module.
0x0700 – 0x07FF	0x0380 – 0x03FF	Reserved for engineering memory and register diagnostics.
0x0800 – 0x08FF	0x0400 – 0x047F	Buffer for Flash Data
0x0900 - 0x0901	0x0480	Flash Program Number 1-11
0x0902 – 0x0903	0x0481	Starting Sector in Flash program area. 1-384
0x0904 – 0x0905	0x0482	Total number of sectors to be written. Value between 1-384
0x0906 – 0x0907	0x0483	FPGA to be configured and Flash Program Number
0x0908 – 0x0909	0x0484	FPGARegister select and FPGA to be Programmed,
0x090A – 0x090B	0x0485	FPGA Data Register Access
0x090C – 0x090F		Spare
0x0910 – 0x0911	0x0488	Checksum sent with Last Program Area 1 download from 1553 2-bytes
0x0912 – 0x0913	0x0489	Checksum sent with Last Program Area 2 download from 1553 2-bytes
0x0914 – 0x0915	0x0490	Checksum sent with Last Program Area 3 download from 1553 2-bytes
0x0916 – 0x0917	0x0491	Checksum sent with Last Program Area 4 download from 1553 2-bytes
0x0918 – 0x09FF	spare	
0x0A00 – 0x0AFF	0x500 – 057F	Checksum of Flash Program Area - 1st 256 Sectors of Programs 1,2,3,or 4

0x0B00 – 0x0BFF	0x580-05FF	Checksum of Flash Program Area - 2nd 256 Sectors of Programs 1,2,3,or 4
0x0C00 – 0x0CFF	0X600 – 0x67F	Checksum of Flash Program Area - 3rd 256 Sectors of Programs 1,2,3,or 4
0x0D00 – 0x0DFF	0x0680 – 0x6FF	Checksum of Flash Program Area – Remaining Program Blocks (see table 5)
0x0E00 – 0x0FFF	0x700 – 0x7FF	Spare

Table 2

Flash Memory Map

This table shows the general organization of the Flash RAM as a quick reference. Unused sections are labeled spare sector.

Table 2B

Address	*Program Type*	*Program / Sector Number*
0x00000 - 0x180FF	*Standard Program 1 for FPGA/TRIP Chips –384 sectors*	*Pgm#1 – Sector 1- 384*
0x18100 - 0x181FF	*LVDS fake track info storage 70bytes*	*Test data 1 – Sector 385*
0x18200 - 0x184FF	*Ssequencer Test Pattern 3 sectors*	*Test data 2 – Sector 386-388*
0x18500 - 0x186FF	*Trip Chip Static pedestal subtraction allocation (4-sectors)*	*Setup Data #3 – Sector 389-392*
0x18700 - 0x196FF	*Trip Chip power-on initialization parameters (16 sectors)*	*Setup Data #4 – Sectors 393-408*
0x19700 - 0x197FF	*Trip Chip Clock Initialization (1 sector)*	*Setup data #5- Sectors 409*
0x19800 – 0x1FFFF	*Spare sectors (101)*	*Sectors 410 - 511*
	Flash User Area Sectors 512 – 2048 Not LockedOut	
0x20000 - 0x37FFF	*User Program 2 for FPGA/Trip Chips – 384 sectors*	*Pgm #2 Sectors 512 - 895*
0x38000 - 0x4FFFF	*User Program 3 for FPGA/Trip Chips – 384 sectors*	*Pgm #3 sectors 896 - 1279*
0x50000 - 0x67FFF	*User Program 4 for FPGA/Trip Chips – 384 sectors*	*Pgm #4 Sector 1280 - 1663*
0x68000 - 0x681FF	*Dynamic Pedestal subtraction (2 sectors)*	*Setup Data #6 Sector 1664 - 1665*
0x68200 - 0x7FFFF	*spare sectors (383) of (total 2048)*	*Spare - / Sect 1666 - 2048*

Microcontroller Command Processing Memory Block

Upon power up the microcontroller resets and then enters an idle loop. As part of the reset, locations 0x00A0 through 0x00AF (Microncontroller command block) and locations 0x00B0-0x00BF (Microcontroller Status Block) are cleared to zeroes. In the idle loop, the microcontroller polls location 0x00AF, and if it is non-zero, interprets the values found in 0x00A0 through 0x00AE as a command sequence. The commands stored in these locations are processed in order. After the queue is empty the microcontroller re-enters the idle state.

Address(es) as programmed from microcontroller	Address(es) as programmed from 1553	Function
0x0000 – 0x009F	0x0000 – 0x004F	Reserved.
0x00A0 – 0x00AD	0x0050 – 0x0056	Microcontroller command queue. Each word is a separate command. Only the lower byte of each word is examined by microcontroller.
0x00AE	0x0057 (LSByte)	'Execute Command List' location. Set to non-zero value to force microcontroller to execute commands previously stored in queue above.
0x00AF	0x0057 (MSByte)	Unused by microcontroller.
0x00B0 – 0x00BB	0x0058 – 0x005D	Reserved for microcontroller internal diagnostics.
0x00BC – 0x00BD	0x005E	Microcontroller Command Error Buffer. LSByte will contain command value which had an internal error, MSByte will contain command-specific error code.
0x00BE	0x005F (LSByte)	Microcontroller Command Loopback. Contains value of last command processed by microcontroller.
0x00BF	0x005F (MSByte)	Microcontroller heartbeat. Regularly incremented by microcontroller; if not changing, micro program is stuck and board requires reboot.

Table 3

Microcontroller Command Queue Processing Description

The microcontroller, when triggered by writing a non-zero value to location 0x0057 (1553 side), begins processing by first reading the data value stored at location 0x0050 (1553 side). The microcontroller executes each command after it is read, then advances to the next location. As a general rule the micro is infinitely fast compared to 1553 speeds, so that the entire list will typically be processed before the 1553 bus can check to see if the first command has been processed.

The command list must be terminated by a zero. The microcontroller will continue to process commands in the list until it finds a value of zero, which it interprets as both 'no-op' and 'terminate list'. As a safeguard, the micro clears the trigger location (0x0057 from the 1553 side) to zero as soon as it sees the trigger. An additional safeguard is that each command in the queue is also cleared to zero as it is processed. *Best practice, however, is to always write the entire queue and fill unused commands with zeroes to avoid erroneous operation.*

Although real-time monitoring of the microcontroller is impossible from the 1553 side, because the micro is so much faster, a minor diagnostic is available at location 0x005F (1553 side). The least significant byte of this location will always contain the last command processed by the micro, and the most significant byte will contain a heartbeat that is continuously incremented by the micro. Should the event monitoring system at any time suspect that the microcontroller is 'hung', it may interrogate this location.

Microcontroller Reboot/Restart

The microcontroller reboots upon power up or when the manual reset pushbutton on the board is depressed. Should remote reboot/restart be required for any reason, it must be accomplished by using the Rack Monitor system to shut down the power supply associated with the board in question. A 'soft reboot' command is implemented which will restart the microcontroller's program and reset the various registers and DACs to standard values, assuming that the micro is capable of responding to the command.

Microcontroller Command Format

Each 16-bit value written to the command queue is interpreted as a single command value for the microcontroller. The lower 6 bits are the actual command, and at present the upper bits are ignored. The commands implemented are as shown in Table 4. The majority of commands read and/or write locations in the dual port RAM. The details of how those sections of RAM are implemented are in the following sections of this document.

I've made some modifications to the table based on how the commands in the board were *actually* implemented as opposed to what we *planned* to implement. There will also be a bunch of changes because the TRiP is different than the MCM. You'll want to check the list of commands in here versus the list of CMD-xx.ASM files in the actual code to see if my changes are correct. Once the 'dead' commands are identified then the lower sections of this document associated with those commands need be adjusted to match the revised table.

Value	Command Function	Notes
0x0	No-op / terminate command list	
0x1	Useable	
0x2	Useable	
0x3	Useable	
0x4	Useable	
0x5	Unuseable	AFE II has no external DACs but the TriPs have internal DACs. The number of DACs and their resolution will be different in AFE II.
0x6	Useable	
0x7	Same as AFE 1	We ended up not being able to do this on AFEI because of noise issues, however, in AFE II this will be implemented.
0x8	Useable	
0x9	Useable	
0xA	'download Sequencer Test Pattern'.	This command goes away as we have removed the Inbound FIFO. However, we may bring it back in a new form as something like 'download Sequencer Test Pattern'.
0xB	Unuseable	There is no Monitor FIFO in AFE II.
0xC	Set VSVX Operating Parameters	Loads VSVX control byte from dual-port RAM into VSVX control register 1 per VSVX documentation.
0xD	Useable	
0xE	Used in both AFE1 and 2	Transfers list of eight seed bytes from dual-port RAM to the eight LVDS Data Mux CPLDs on board; seed used for test data sent during Sync Gap.
0xF	Preferred for Cryo	
0x10	Preferred for Cryo	
0x11	**Useable**	
0x12	Not useable	Manual read ADC AFE-1
0x13	**Preserve**	Clear real-time clock in 1553 PLD AFE-1
0x14	**Useable**	
0x15	**Useable**	
0x16	**Useable**	
0x17	**Useable**	

0x18	Reserve	
0x19	Preserve	Clears all ADC readback data to insure that any later reads are fresh. Zero out ADC readback values
0x1A	Not useable	While I believe that some code is still present in the PIC to perform tests, I think it is now decoupled from the command processor section and can't be called externally.
0x1B	Useable	
0x1C	Useable	
0x1D	Useable	
–0x1E	Reserved for future expansion.	Do not use.
0x1F	Request Soft Reboot	Microcontroller will go through reset sequence, to leave board as close to power-up state as possible.
0x20 –	**Load Flash Command**	**Read byte from 1553 then write DPRAM location 0800-08FF**
0x21	**Read Flash Sector**	**Read Out 1 Flash sector Based on same addressing as 0x20**
0x22	**Perform CHECKSUM on Flash Sectors**	**Routine to check data integrity**
0x23	**Load Trip Chip Configuration Parameters**	**Read Trip Data from Flash to Load FPGA registers.**
0x24	**Configure FPGAs**	**Commands to perform FPGA Configuration from Flash**
0x25	**FPGA register Control**	**Xilinx command to modify values in control register**
0x26		
0x27	Manual ADC readback	Manual ADC Readback
0x28		
0x29	Useable	
0x2A	Useable	
0x2B	Useable	
0x2C	Useable	
0x2D	Useable	
0x2E	Useable	
0x2E	Useable	
0x30	Reserved	Used on AFE Test Module only. Load pattern for test charge delivery in AFE Test Module.
0x31	Reserved	Used on AFE Test Module only. Load Test Charge Amplitudes
0x32	reserved	Used on AFE Test Module only. Set Temperature Sense Feedback
0x33	Reserved	Used on AFE Test Module only. Run Test Cycle
0x34	Reserved	Used on AFE Test Module only. Convert Heater Drive
0x35-	Reserved	Used on AFE Test Module only. Reserved for future expansion.
0x36	Reserved	
0x37	Reserved	
0x38 –	Reserved for future expansion.	Do not use.
0x39	Reserved	
0x3A	Reserved	
0x3B	Useable	
0x3C	Useable	

0x3D	Useable	
0x3E	Useable	
0x3F	Useable	

Table 4

Microcontroller Status Information

Memory locations 0x00B0 through 0x00BF are reserved for microcontroller status. Locations 0x00B0 through 0x00BB are used for internal, command-specific information, for use with engineering testing. Do not rely on these locations to contain any consistent information. Location 0x00BE is filled with the last command processed by the microcontroller; should the micro ever hang up, this location may give a clue to what died. Location 0x00BF is a heartbeat location which is simply incremented by the microcontroller every time the code loops through the polling loop that looks for new commands. Since the 1553 interface will be asynchronous to this loop, monitor code should simply look for change in the heartbeat rather than a predictable increment.

If any command cannot complete or detects a fatal error condition, location 0x005E (1553) will contain a non-zero value. The lower eight bits will contain the command which had the error and the upper eight bits will contain a command-specific error code. These error codes will be appended to this document as development ensues. ***If a command detects a fatal error condition, the entire command queue is cleared and any commands which followed the one that found the error will not be executed.***

A word about Microcontroller Speed

Simply put, this is not a fast processor. In comparison to 1553, things go quickly, but a typical command will take microseconds or even milliseconds to execute. The processor clock is a 12 MHZ clock (~80nsec per instruction), and most commands will require a few hundred to a few thousand instructions. In addition, instruction pipelining is never perfect, so the average instruction time is probably more like 150 nsec each. If a command takes 1000 instructions to perform, that's going to take 150 usec or so.

The 16F877A doesn't have an internal A/D; we'll be using an external A/D converter instead.

This feature was disabled in the AFEI after I implemented it, as the #$%(&#$% physicists couldn't leave their %^$!%^ hands off the firmware.

Method to Insure Flash Data Integrity

1) Command 0x22 will perform a 16-bit checksum of the FPGA configure program in Flash. And store the results starting at DPRAM location 0x0xA00-0x0DFF. Typically these should contain the same values as the control system but if a special configuration was used there will be a record.

2) Command 0x21 will read a sector of Flash. The user will be able to compare these values with system values. This does not guarantee that the FPGA is programmed correctly but it does give some confidence on the data transfer integrity.

3) The Done bit of the FPGA will be tested after program to determine if the FPGA had a successful program cycle. If, Done is incomplete it is assumed that the FPGA's are not programmed.

4). One-fourth of Flash has been reserved as a Flash Lockout area of Default Programs. Standard program One and certain setup data will not be downloadable through the control system. This section of Flash will controlled isolated by a hardware jumper that will not be installed during normal operation of the AFE-2. This is a protection to the user in case there is an upset of the system that makes it necessary to return to a nominal operating mode. It is advised not to try and override this feature.

5) Flash Program areas 1,7,8,and 9 checksums are written to DPRAM from 1553 to provide a way of determining the last program downloaded from the control system. Checksums are stored at 0x0910 – 0x0917.

AFE-2 Dual Port Ram, Flash, and FPGA Loading Algorithm

This document explains from top down the process that is necessary to program dual port ram, flash, and download FPGA configuration memory from dual port ram to flash . See Figure 2 for Funcional Block diagram.

The devices that will be used are

- Cypress CY7C136 – 2Kx8 Dual Port Static Ram. Two are used. The 1553 side is a 2Kx16 bit bus whereas the PIC Processor side is a 4Kx 8-bit bus (DPRAM).
- Flash memory is Atmel AT29C040-15 - 4M bit or 512K x 8 bits, arranged as 2048 - 256 byte sectors. Flash memory write time is 11ms per sector.
- Micro-Controller is Microchip PIC16F877A-20I/PT
- Xilinx FPGA XC2S100-5PQ208C 2.5v core with 40K block Ram (configuation file size 97808 bytes)

Loading Dual Port Ram to Flash

The current architecture is such that the 1553 interface has direct access to one side of the DPRAM. Commands from 1553 are executed through the 1553 CPLD that allows memory access directly without any involvement of the PIC Micro-controller.

It was determined that it was most efficient to load 128- 16 bit words from 1553 into 256 address locations into DPRAM. Address 0x400 – 0x047F will be used. Since the DPRAM is 8- bits wide, high-byte and low-byte are written to adjacent locations.

Since 1553 has a maximum of 32- 16-bit words per transfer (64 bytes) it will require 4- transfers (256 bytes or 1 block) from 1553 to fill the DPRAM address field with data. The Helper CPLD generates address and control to the DPRAM

A 4-byte control word transfer from 1553 will help identify the a valid Flash starting sector address table 2A. The download program will have to correlate the program that is to be loaded with the actual program area of the Flash to be programmed. See tabe 2B.

The first control word has the program number which has the range of 1 to 15. This value is loaded at DPRAM 0x900-0x901.

A second control word will have the starting sector number within the program block area. The sector numbers are a number between 0 and the size, in sectors of the block. locations inside the Flash. This value is stored at DPRAM address 0x902 – 0x903. The program block number isolates a unique section of Flash, so that the corresponding Flash sector must be within the programming range of sectors. A table will be used to compare the program number versus the hardware definition of the Flash. If this check is passed then a second table that gives the starting address in program number will be compared to the values in 0x902-0x903. See Table 5A for the sector address ranges.

On power-up to the DPRAM is loaded at addresses starting at 0x0901 - 0x0900 the value x0001 which is Program area 1. See table 2A. Next DPRAM 0x902-0x903 is loaded with the starting sector of Program area 1. A look-up table in PIC Ram determine the starting address in Flash.

Flash Download from Control System – Load Flash Command 0x20

1) Implement one and only one command that allows the end user to download one 256-byte sector into the Flash. The external program must, prior to the execution of the command, write one 16-bit word into a location in dual port ram to indicate which "program number" they want to play with and a second 16-bit word to indicate the "sector number within that program" they want to change. The external program then loads up the 256 bytes followed by asking the PIC to do the transfer.

2) Upon beginning the command handler, the PIC looks at location 0x0900. Which should contain a number between 1 and the maximal value in table 2B (14 or whatever it is). If the number read is 0 or 1, the PIC exits with an error message and the Flash is not changed. If the number read is greater than the maximal value in table 2B, same error. If the number read is legal, hard-coded constants in the PIC code look up the correct values for the Port B and sector address latch bits for the FIRST sector of that block,storing them in internal PIC register locations SEC_ADDR_HIGH and SEC_ADDR_LOW. You can implement any number of read-only lookup tables of any sort you want in the PIC code trivially by just storing the data in

the program space of the processor.

3) If step 2 didn't exit in error, the value read from 0x0900 is also used to look up in another hard-coded table to obtain a two-byte maximum number of sectors in this block. For the program blocks (values 1,7,8,9) that's obviously 384 The other values have smaller numbers. A value of zero is legal; these relative sector numbers are counted from 0 to the maximal sector number. Thus, a value of zero means that the block is exactly one sector long.

4) If step 2 didn't exit in error, then the PIC reads locations 0x0902 and 0x0903. If the number stored therein is greater than the maximal number of sectors looked up in step 3, the PIC doesn't touch the Flash and returns an error message. If the number stored is between 0 and the maximum number of sectors, then the PIC adds the data in 0x0902 and 0x0903 to SEC_ADDR_HIGH and SEC_ADDR_LOW using 16-bit addition. The PIC then writes the appropriate bits of SEC_ADDR_HIGH to Port B and writes SEC_ADDR_LOW to the external sector address latch. By definition they have to be valid as they've been created from a hard-coded lookup table and a checked input value.

5) If neither step 2 nor step 4 had an error, transfer the sector. Upon completion of the transfer, zero out locations 0x0900 - 0x0903 such that the end user is required to re-load these in order to execute the command again.

6)There are no other commands. To transfer a bunch of sectors, the external program has to do them all one at a time, each time loading all the information required in steps 1 - 5 above. The above algorithm handles every possible situation. It allows for there to be up to 255 "blocks" in the Flash, each of which may be any size
from 1 up to 16384 sectors long.

Note that 384 Sectors of (256 bytes) ares loaded for each Program Block area. Total time to load 256 bytes (1 sector) from DPRAM to Flash is 11ms.

DPRAM address 0x0901-0x0900 Flash Program Area (0-15)

XX	xx	xx	xx	xx	xx	xx	xx	xx	xx	XX	XX	8	4	2	1

Table 2A

DPRAM address 0x903-0x902 Range 1 to 512 . Starting Flash Sector

xx	xx	xx	xx	xx	xx	xx	256	128	64	32	16	8	4	2	1

Read Flash Sector – Command 0x21

This command enables the system to read a sector of flash to DPRAM at locations 0x800 – 08FF which is the Flash buffer,will also be used to read back a given sector in Flash memory. This feature will help assure system stability as actual comparisons can be made by reading back the data sored in any area of the Flash memory. Table 2A above will be used to setup the sector of Flash. The Flash Program area can be any of the 10 that are currently defined. The starting Flash sector can be any. Total number of sectors to be written is always 1 A sector of Flash data will be available to the system in DPRAM for the read after the command is done. The Upload program of the AFE system will need to read all 256 bytes and compare them to the known values in order to make a comparison.

Command 0x21 is written to the command register by the user.

Read first table for Program # starting address

Generate address and control for FLASH read

FLASH prepared for a read while the DPRAM will be set for a write.

DMA from FLASH to DPRAM to locations 0800 – 08FF

Flash Memory Program Block and Lookup Table

Table 5 Flash Memory Data Type , Program Block and Checksum

Program Block 0x901-0x900	Flash Memory Program Data Type	Starting Flash Address in Program Block	Starting Sector 0x903-0x902	Sectors max count 0x905-0x904	Checksum Result Address DPRAM	DPRAM Bytes count
00	Not Used					
01	Standard program 1	0x00000	0-384	384	0xA00- 0xCFF	768
02	LVDS fake track	0x18100	0	1	0xD00 - 0xD03	4
03	Sequencer Test Pattern	0x18200	0-2	3	0xD04 - 0xD0B	8
04	Trip Chip Static Ped Sub	0x18500	0-1	2	0xD0C - 0xD13	6
05	Trip Chip power-on init	0x18700	0-15	16	0xD14 - 0xD35	33
06	Trip Chip Clock Initialization	0x19700	0	1	0xD36 - 0xD39	4
	Spare	0x19800 – 0x1FFFF				
07	User Program 2	0x20000	0-384	384	0xA00 - 0xCFF	**
08	User Program 3	0x38000	0-384	384	0xA00 - 0xCFF	**
09	User Program 4	0x50000	0-384	384	0xA00 - 0xCFF	**
10	Dynamic Pedestal Subtraction	0x68000	0-1	2	0xD3A - 0xD3F	6
	Spare	0x68200-0x7FFFF				

Lookup Table 1-5

Since the algorithms vary by the type of data, the PIC jumps to the appropriately coded routine based upon the data in this lookup table. The Program number is used to qualify the data type as to what it will be used for. If the values fall within the accepted range of 1-255 at DPRAM location 0x0900 then a second table will be accesses.

Lookup Table 2 has the starting Flash sector information, if the value found in DPRAM location 0x902-0x903 is within the range then a third table is accessed.

Lookup Table 3 is the total number of sectors to be written, if the value found in locations 0x0905-0x0906 is within the range then the steps to create Flash address accesses are implemented.

Lookup Table 4 will be used by the Checksum command to store its results in DPRAM at the proper location. The size of the checksum data returned is based on the Program type so a table jump is needed. These lookup tables are non-volatile, inside the PIC (not in the ram, not in the flash , in the PIC) and unavailable to the end user.

Lookup table 5 will be used to setup the FPGA to be configured used with Command 0x24.Table will accessed for CS.

Loading FPGA Configuration Data

All FPGA's will be initialized with the configuration data for Standard Program 1 for Trip Chips. Any other of the 3 program selections must be programmed through the 1553 interface. The Flash memory holds the configuration data for all the FPGA's.

The FPGA of choice is the Xilinx XC2S100 requires 97,808 bytes (383 sectors @ 256 bytes per sector 98048) of data to configure. The Slave Parallel Mode was chosen for configuring the FPGA's since it is the fastest mode. Set slave parallel mode will be hardwired on the board.

It is assumed that each of the 8- FPGA's will be configured with the same data. There are 4- configuration programs that are possible using the Flash memory now chosen. One of the program areas are defined as standard program, and 3 are defined as User Programs. See Table 2B.

If CCLK (configuration clock) frequency to the FPGA's were to operate at 1Mhz, it would take 100ms to configure all 8-FPGA's since they would be configured in parallel.

It is possible that one or more of the eight may need some special configuration so each FPGA has an independent chip select line that can be enabled for individual programs to be loaded.

The read speed of the Flash is 90ns per access so with a CCLK rate of 1us there would be plenty of setup and hold time for the Flash.

The PIC clock is 12Mhz but typical I/O operation is 150nsec per instruction. Therefore to read Flash; and setup configuration to helper might take 2-3 instructions, plus read delay or about 500nsec. Total configure time about 50 usec.

When a change is needed in the FPGA configuration you must reconfigure the whole device, that will require a download 383 sectors from Flash to the FPGA in that special case.

Power-up or asserting the Program input of the FPGA starts the init configure sequence which clears the configuration memory. At the end of this phase the init pin goes high which indicates that the FPGA is ready for configuration data.

Configure FPGA's – Command 0x24

There is one and only one command implemented to request that the PIC download data from the Flash to some object on the board. The user must pre-load 0x0900 and 0x0901 with the 'program number' and the the FPGA selection bitmap into location 0x0906/0x0907.prior to calling this command.

The PIC performs the same test here to insure that 0x0900/0x0901 are between 1 and the maximal block number as in step 2 above. The only difference is that here, a value of 1 is always OK. An additional test is performed on 0x0906/0x0907 to see if the "Program number" is 1,7,8,9 and if the value has at least one FPGA selected.

The PIC then uses the EXACT SAME hard-coded lookup tables to determine the starting address and number of sectors that it must transfer. A THIRD lookup table is used in the PIC code to jump to the appropriate downloading routine.

Program number 1,7,8,9 are FPGA configure downloads, but other blocks are used for other functions.

The FPGA control data word will have the following design. See Table 2C. The user will be able to select which of the 8-FPGA's will be configured and which Program area will be used.

FPGA Configure Command (0x24) will read from the command location and the data at location 0x900 and select the starting address of Flash Memory that will be used for FPGA configure. Then based on which bits are set in the 0x0906 will set the chip enable to the appropriate FPGA to be programmed.

Addresses to FLASH are setup so that FLASH is prepared for read operation.

Sector Address High and Low are set to the Program Block starting address in FLASH.

The FPGA Init Port (PB5) is tested to confirm status if status is set or reset .

Helper CPLD asserts Program, CS1-8 and CWRT control to FPGA's.

When Program pin of FPGA is asserted low it will cause INIT to go low. When INIT returns high the Load begins. INIT's return to high will cause the DONE, Port (PC6) to go low.

Done Port (PC6) is tested for status.

The FPGA is now ready for DMA transfer from FLASH to FPGA.

The PIC controls the DMA process until Done goes back high.

There will be 97808 bytes of Configuration data transferred from FLASH to FPGA.

If Done does not go high, it is assumed that the FPGA did not program. An ERROR is generated.

A timer is set at the start of the DMA process which will give a relative time for download from FLASH to FPGA if the time exceeds the expected default of the process then an ERROR is generated.

Table 2C (0x907) Program Flash Block FPGA to be Configured (0x906)

XX	XX	XX	XX	8	4	2	1	8	7	6	5	4	3	2	1

Checksum Flash Memory Command –0x22

A block of 1024 bytes of DPRAM has been reserved to provide checksum information to the control system. DPRAM addresses 0xA00 – 0xDFF will contain a calculated checksum performed on as many as 512 sectors of Flash by the PIC. Each checksum is a 16-bit word. The command should perform the checksum on the whole Program Area that is requested. In practice a comparison could be made between what is calculated in a given sector of Flash versus what was downloaded from the control system. This function can be used for diagnostic functions on any of the 2048 sectors of the Flash memory. A simple read and sum of all the data in the sector will produce a value where the total may exceed 16-bits, in that case only the lowest 16 bits are retained as valid.

Checksum Flash is a solicited command. If the Program Block type selected is 1,7,8,9, which are 384 sectors the command will generate the checksum for all the sectors and store the values in DPRAM at addresses 0x0A00 – 0x0CFF will be updated with the results. The first 2- byte word stored will be the total number of valid words to follow, which is the total number of sectors that will be summed. The subsequent list will be the individual checksum of all the sectors.

If the Program Block selected is 02, (LVDS Fake Track Data) the the checksum results will be stored at 0xD00-0xD03. The first word is the number of valid words to follow the remaining word is the checksum. See table 5.

Algorithm

Input - Program Area # at x0900 of DPRAM (a number from 1-255)

Output - Number of valid words to follow 2-bytes

And List of checksum words 383 words.

Process

1. Fetch Input from DPRAM.
2. Read first table for Program # starting address
3. Generate address and control for Flash read
4. access 2nd table for the total number of sectors and
5. read 3rd table for first DPRAM location
6. Write #of sectors to DPRAM as first word in list.
7. Checksum first sector
8. Write value in DPRAM
9. Increment to next sector and next DPRAM address
10. Repeat steps 8 and 9 until all sectors are read
11. Done.

FPGA I/O Register Control – Command-0x25

Commands to modify FPGA I/O registers are used to control the TRIP chip functions. Many of these functions are yet to be defined. Flash memory has 4 sections reserved to hold data that can be used to control the operation of the TRIP Chips. See Table 2B. There are 128 I/O r/w registers that control the TRIP Chip'ssThe user will be able to select the I/O registers and direct which FPGA's to modify. The FPGA register I/O function will controlled by ASTRB (address strobe), DSTRB (data strobe) ,PICDATA Bus, and FPGA CS1-8.

The DPRAM is used to hold register control address and data. DPRAM address 0x0909 holds the I/O register address where the upper byte has the MSB to indicate whether the access is a write or a read, the following 7 bits are used to select which register is used. The lower byte (0x908) is used to select which FPGA is selected. Data is stored at location (0x090B). All read and write transfers will be stored in these memory locations. See table below.

DPRAM address 0x908 – 0x909 contains the FPGA I/O Register access format data. Register number, the FPGA number and Data .

(0x909) Register Select for Register Read and Write FPGA to be Programmed 1-8 or all (0x908)

R/W	64	32	16	8	4	2	1	8	7	6	5	4	3	2	1

(0x90B) Data 8-bits (0x90A) not used

x	x	x	x	x	x	x	x	nu	nu	nu	nu	nu	nu	nu	nu

Registers in FPGA

1) ADC data selection (10-8)
2) Predetermined offsets for each channel, address register, and data register
3) DPS threshold
4) Readout Control
5) Trip Download Parameters address and data
6) Trip Download Control
7) VSVX Muxc Control Register
8) Trip Power Control
9) Event Delay Timing Register

Download Trip Data From Flash – Command 0x23

It is necessary to provide a means to download the data stored in Flash in program blocks 4,5,6 and 10 to the FPGA registers. The FPGA is the interface to the Trip Chip which controls the registers inside of the Trip Chip. The Trip Chip command and bus control must be defined to allow the PIC to access and to enable the transfer of data. Currently there are 23 sectors reserved for use as data storage for the Trip Chip. See table 5 for Program Block area descriptions.

AFE-2 Memory Block Diagram

The figure below illustrates the primary blocks and connections that are used to control memory access in the AFE-2 board. Basically there is a common bus called PICDATA that acts as both data and address to all devices. Device strobes are generated by the PIC microcontroller and or the Helper CPLD to perform chip selects. The FPGA's are configured by a DMA process using data stored in Flash Memory. The transceivers are used to isolate the FPGA's from the PICDATA bus and to provide a future method to readout configuration data from the FPGA's if needed.