VME Communication with LBNL ABCD Tester System

v. 3.7

Version	Date, Mo/Day/Year	Who	Description
1.0	March 2000	H. Niggli	Creation
2.0	10/20/2000	V. Fadeyev	Updated list of VME commands; note on TestVectors
2.1	10/24/2000	V. Fadeyev	Added ADCs description
2.2	11/01/2000	V. Fadeyev	Fixed TV Masking description
3.0	03/04/2001	V. Fadeyev	FPGA register and ADC readout modification, more more description
3.0	03/20/2001	A. Ciocio	Review
3.1	03/13/2001	V. Fadeyev	ADC table modification, additional vme call (pd inhibit)
3.2	03/29/2001	C. Flacco	Review
3.3	04/03/2001	V. Fadeyev	ADC units, new VME call (hit pattern)
3.4	04/11/2001	V. Fadeyev	Histo error flags, decoding description, id note
3.5	04/16/2001	V. Fadeyev	Current TV content description
3.6	05/07/2001	V. Fadeyev	Added info on window comparators.
3.7	05/24/2001	V. Fadeyev	Added version call info

Organization of this document

The information in this document is arranged as follows. First we give summary tables of VME commands used to talk with the system. Then we describe different functionalities of the system. At the end there is a section containing a sample list of actions one can use to do a threshold scan and a section with VME interface implementation details.

We will refer to the chip being tested as ABCD throughout this document, thus skipping the version suffixes (ABCD2NT, ABCD3T etc).

The supplement information, such as ABCD chip specifications and ABCD test specifications, is available from the CERN Chip Information page:

http://chipinfo.web.cern.ch/chipinfo/

VME Commands

VME Address:

Bits 31..25	=	8-bit VME board address = set by dip switches on the board
Bits 24..10	=	don't care
Bits 9..4	=>	Execute one of the VME commands listed in Tables 1,2,3
Bits 3..0	=	don't care

Table 1. VME Commands for Board Operation.

Bits 9..4 hex	Read/Write	Description	VME Data Bus
00	Read	Read FPGA Status Register	Bits 11..0 = ACE Bit 12 = HISTO_1_BUSY Bit 13 = HISTO_2_BUSY Bit 14 = DAC_ADC_BUSY Bit 15 = TV_BUSY Bit 16 = BUSY (4 BUSY bits above OR'd together) Bit 17=TVDifferenceFound Bits 31..18 = 0
01	Write	Reset Test Vector Memory Counter to 0	X
02	Write	Reset Simulation Vector Memory Cnt to 0	X
03	Write	Clear Histogram Memory	X
04	Read	Read from Histogram Memory and Increment Memory Pointer	Bits 15..0 = Stream B Bits 31..16 = Stream A
05	Write	Set Base Address for Histogramming and Reset Histogram Memory Counter to xaa000000, where "aa" stands for the Base Address	Bits 7..0 = Base Address Bits 31..8 = X
06	Write	Send Test Vector	
07	Write	Write to Test Vector Memory and Increment the Test Vector Memory Counter	Bits 17..0 = data Bits 31..18 = X
08	Write	Write to Simuation Vector Memory and Increment the Simulation Vector Memory Counter	Bits 17..0 = data Bits 31..18 = X
09	Write	Reset the ReSync FIFO read pointer	X
0a	Write	Set DAC	Bits 9..0 = data Bits 22..19 = dac address Bits 18..16 = dac channel other bits: X
0b	Write	Send Convert Signal to ADC	Bits 2..0 = ADC number
0c	Read	Read ADC Data from last Conversion	Bits 11..0 = data
0d	Read	Read from ReSync FIFO	Bits 8..0 = data Bits 17 = FIFO empty flag:

			0 = empty, 1 = not empty Bits 16..9, 31..18 = X
0e	Write	Start sending triggers and decode and histogram the data	X
0f	Write	Set Frequency	Bits 8..0 = M Bits 10..9 = N Bits 13..11 = T

Table 2. VME Commands used to issue control sequences to the chip or module to be tested. For details on the sequence, consult the ABCD specifications document.

Bits 9..4 hex	Read/Write	Description	VME Data Bus
10	Write	Set Trigger-To-Trigger Delay	Bits 15..0 = delay in cc (25ns) Bits 31..16 = X
11	Write	Set Number of Triggers to be sent per Burst	Bits 15..0 = #triggers Bits 31..16 = X
12	Write	Send Soft Reset (all chips)	X
13	Write	Send BC Reset (all chips)	X
14	Write	Write to Configuration Register	Bits 21..16 = chip addr. Bits 15..0 = config reg data Bits 31..22 = X
15	Write	Reset FPGA-Mask Register Pointer	X
16	Write	Write 32 bits to FPGA-Mask Register	31..0 = mask reg data
17	Write	Send FPGA-Mask Register to a chip	Bits 21..16 = chip addr. Bits 15..0 = X Bits 31..16 = X
18	Write	Load Strobe Delay Register	Bits 15..0 = reg data Bits 21..16 = chip addr. Bits 31..22 = X
19	Write	Load Threshold/Cal DAC Ampl Reg	Bits 15..0 = reg data Bits 21..16 = chip addr. Bits 31..22 = X
1a	Write	Enable Data Taking	Bits 21..16 = chip addr. Bits 15..0 = X Bits 31..16 = X
1b	Write	Select/Strobe Enable	Bits 1: abcd select bit Bits 0: strobe enable bit
1c	Write	Issue Hard Reset	not implemented
1d	Write	Load Bias DAC	Bits 15..0 = reg data Bits 21..16 = chip addr. Bits 31..22 = X
1e	Write	Load Trim DAC	Bits 15..0 = reg data Bits 21..16 = chip addr. Bits 31..22 = X

1f	Write	Set Strobe-To-Trigger Delay	Bits 7..0 = delay in cc Bits 31..8 = X

Table 3. Additional VME Commands.

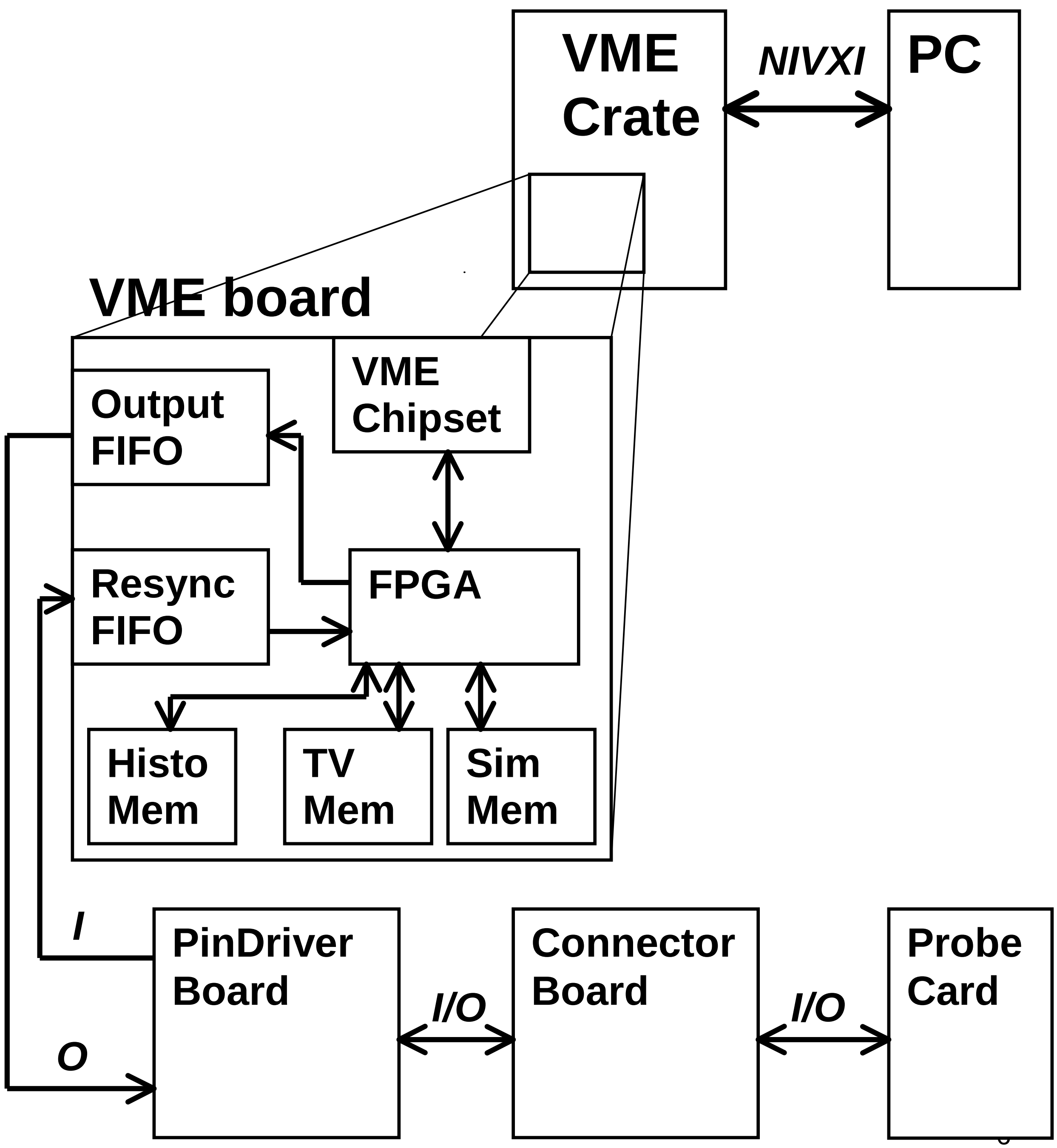
Bits 9..4 hex	Read/Write	Description	VME Data Bus
20	Read	TV Difference location in time sequence	Bits 17..0 = Data (=0 if no difference) Bits 31..18 = 0
21	Read	TV Difference Word	Bits 7..0 = difference seen on the corresp. TV output lines at the 1st difference location; (bit #N = 1 if there was a difference on line #N; =0 otherwise) Bits 31..8 = 0
22	Write	Reset both Output and Resync FIFOs	X
23	N/A	Nothing is implemented here	N/A
24	Write	Tristate ABCD input signals	Bit 0 is data, the input signals are valid if 0, tristated if 1.
25	Write	Select hit pattern to decode the data for the histogramming	Bits 2..0 (see Table 4)
26	Read	Histogramming data decoding (error) flags	Bits 2..0
27	Read	Read the firmware version date.	Bits 2..0 = Year Bits 6..3 = Month Bits 11..7 = Day

Table 4. Hit pattern selection.

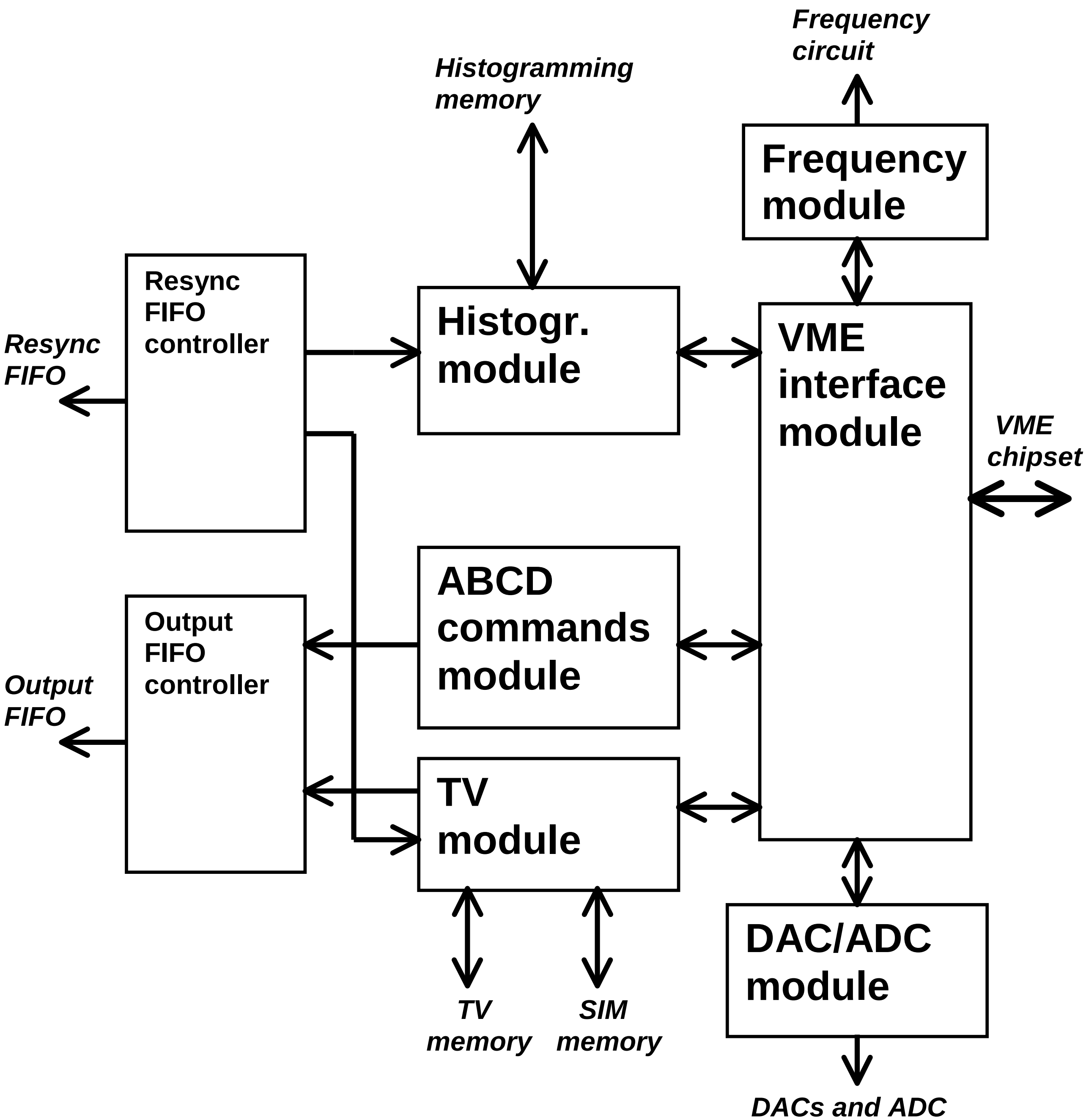
Value	Pattern	Name
0 (default)	1XX or X1X or XX1	Hit mode
1	X1X	Level mode
2	01X	Edge mode
3	XXX	Test mode
4	111	All hits
5	100	1st hit only
6	010	2nd hit only
7	001	3rd hit only

Communication with the Test System

All VME operations are done via programmable IO (no block transfers). That is, we talk to the system by simply writing to and reading from VME addresses, one at a time. These read/write commands get passed to FPGA on the VME board, which is the "brain" of the system. The FPGA controls the data flow, does histogramming, test vector comparison etc. The VME-board-centric view of the data flow is shown in the figure below.



The FPGA firmware consists of several pieces (modules). The simplified diagram of the FPGA partitioning is shown in the figure below. The arrows indicate the major data flow directions. Control signals are not shown.



The major FPGA functional parts are the following.

VME interface

This module decodes the addresses of the VME read/write operations. It is used as an interface to transfer the data and control signals to and from the other modules.

This module has a Status Register, read from address 0x00, which contains the following information:

- *a fixed pattern 0xACE on the lowest bits*

This is useful for testing the availability of the VME communication.

- *busy bits for the histogramming, TV and DAC/ADC modules*

These tell if a module is still performing the previous command and cannot interpret a new one. Note that there are two such bits for the histogramming module. One should OR these bits to decide if the module is still busy.

- *the result of running a test vector*

If a difference between the chip response and the expectation is found, then bit 17 is set to 1. The default is 0.

Frequency

Most parts on the VME board, including the FPGA, run at a fixed frequency of 40 MHz. However, the FIFOs can talk to the outside world at higher frequency. In this case the rest of the hardware (pin driver and connector boards and probe card) will be forced to operate at this high frequency. We use this functionality for running test vectors only. The purpose of the high frequency operations is to provide a non-destructive test of a radiation damage of the ABCD chip, as it was noted that some parts of the chip slows down after receiving the radiation dose.

To obtain the high frequency, we use a phase-lock-loop (PLL) with 20 MHz clock (40 MHz divided in half) as an input reference. The frequency module in the FPGA transforms the input parameters needed to program PLL into a serial bitstream and sends it out. The output frequency (in MHz) is

$$\mathbf{Fout = (Fin/8) * (M/N)}$$

Here,
Fin is the 20 MHz input reference frequency,
M is the frequency multiplier (between 2 and 511),
N is the post divider (2, 4, 8 or 16).

There is a constraint: the value of internal frequency, **(Fin/8) * M**, must be between 400 and 800 MHz. To satisfy this requirement the values in Table 5 are recommended to program the PLL. Note that the values of N used in the formula are used internally by the frequency synthesizer chip. To set those values, we program the chip with the related values of N(load).

Table 5. Input parameters for frequency synthesis.

F, MHz	N(formula)	N(load)	M	F step, MHz
25 < F < 50	16	3	16*F*(8/20)	0.156
50 < F < 100	8	2	8*F*(8/20)	0.313
100 < F < 200	4	1	4*F*(8/20)	0.625

Histogramming

The purpose of this module is to provide the logic for on-board histogramming of the data coming from a threshold scan. The module is able to interpret the data format coming from the ABCD on the datalink/LED line, extract the numbers of channels having hits from physics data packets, and increment the number of hits for these channels in the histogramming memory on the VME board. It can also clean up (write zeros to) the memory (write to 0x03) or read the number of hits for a channel and pass this information to the VME module (read from 0x04).

The histogramming memory is treated as 256 blocks of 16x128 addresses. Here 16 is the max number of chips on a module that this system could potentially work with, and 128 is the number of channels per chip. The *Set Base Address* command (write to 0x05) sets the internal pointer to one of the 16x128 blocks. This is the location where the module would store the data from a threshold scan. When one reads the number of hits from the memory (read from 0x04), the values for different channels are read out sequentially starting from address 0 in this block.

The only way to erase the memory is via the *Clean Memory* command (write to 0x03), which cleans up all the memory. This takes a rather long time (13 ms), which means that:

- one has to check for the BUSY flag to go down before issuing any other VME command later on,
- it makes sense to issue this command only when all or nearly all of the 256 base addresses have been used up.

When working with the data, the module

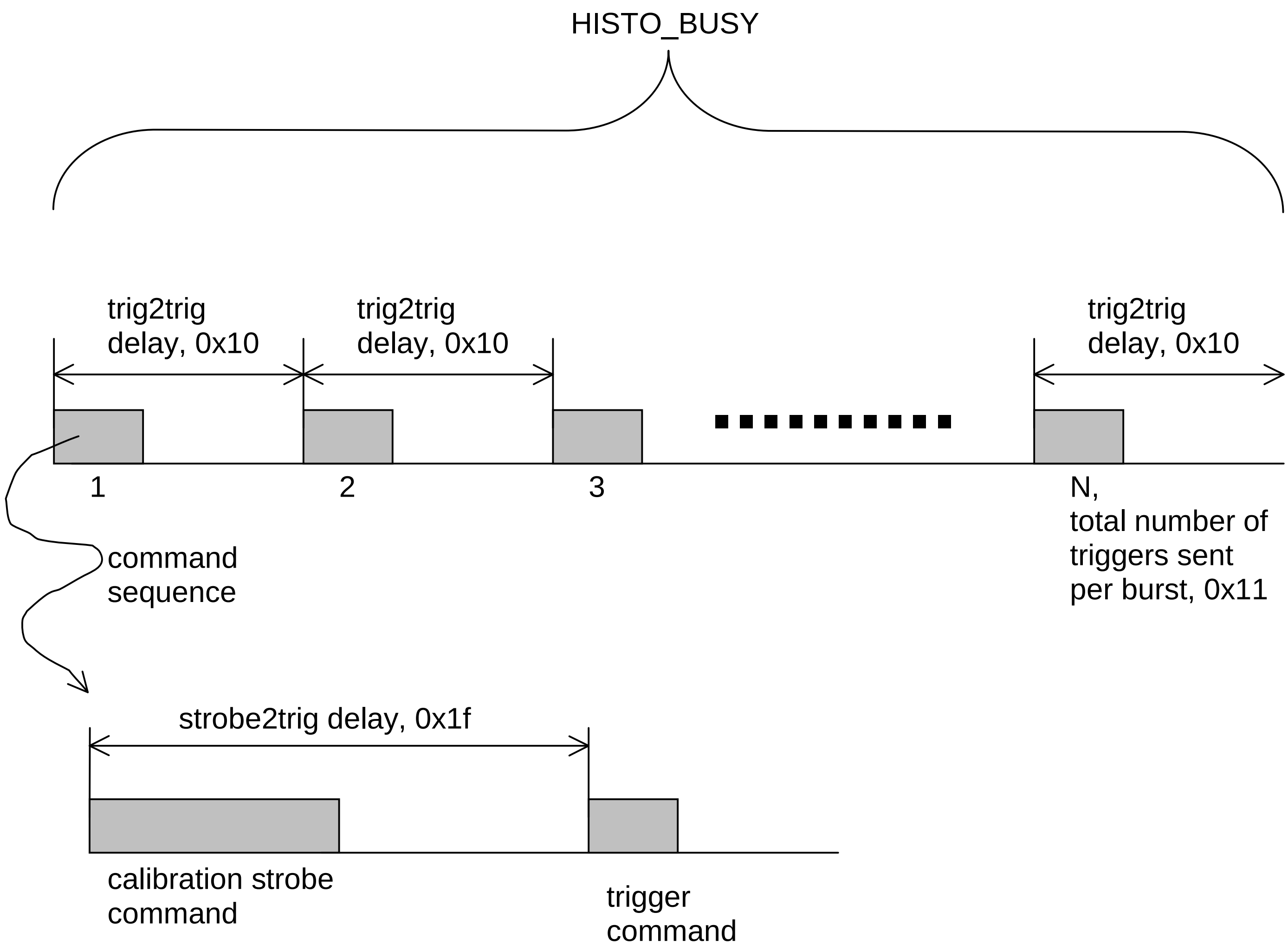
1) extracts a channel number from the bitstream,
2) reads the number of hits for this channel from the histogramming memory,
3) increments the number of hits,
4) writes the new value back to the memory location.

The steps (2)-(4) take 6 clock cycles to complete. The channel extraction is concurrent with the data stream, therefore it does not contribute here. This number is smaller than the 17 bits describing an isolated hit data packet and larger than the 4 bits describing a hit in the non-isolated hit data packet. To prevent the data loss in the latter case, the extracted channel numbers are passed through an internal 64-cell deep FILO.

Caution: the trig2trig delay, described in the next section, has to account for this ≈64 clock cycles delay when reading out all channels from ABCD (the worst case).

ABCD commands

The main purpose of this module is to compose and send out the bitstream patterns used to communicate with the ABCD chip. It has the most relevance for making the threshold scans. The commands are listed in Table 2.



For the purposes of the threshold scan, the module is made capable of issuing multiple triggers ("burst"), interspersed with calibration strobe commands (shown in the picture above). The user-tunable parameters are:

- *the number of triggers sent,*
  The command is write to 0x11. The corresponding register has 16 bits.

- *the spacing between the beginnings of two identical command sequences, in clock cycles,*
  The command is write to 0x10. The corresponding register has 16 bits. Note that this time includes the length of the commands bitstream.

- *the delay between the beginning of the calibration strobe command and the beginning of the trigger command, in clock cycles,*
  The command is write to 0x1f. The corresponding register has 8 bits. Note that this time includes the length of the calibration strobe command.

- *whether the calibration strobe command is present in the sequence.*
  The command is write to 0x1b. The calibration strobe command will be present in the command sequence if the bit 0 is set to 1.

TV

This module is responsible for the test vector functionality in the system. It can load the test vector content into a dedicated memory on the VME board, send it out, receive the data from the chip, compare with the expected data, and provide the result of the comparison (weather the bitstreams matched).

A Test Vector can be thought of as a sequence of numbers which define the state of the control lines going to chip being tested on consecutive clock cycles. The correspondence between the bits in a number and the control lines going to the chip is given in Table 6. The bits 14 through 16 are not used; they remain only for historical reasons.

The Simulation Vector defines the expected chip response to the corresponding Test Vector. The data are presented on the lower 4 bytes, of which bits 5..0 define the response per se (Table 7) and bits 12..8 act as a mask for them:

Data = xxxM MMMM xxxD DDDD

where D's are data lines and M's are mask lines.

The bits are defined in Table 7. As an example, datalink/LED line is used in the comparison if bit 8 is equal to one and not used otherwise.

The test and simulation vectors are usually stored as a sequence of numbers in an ASCII file.

To load the test or simulation vector into corresponding memory on the VME board, one can perform the following actions:

- reset TV/SIM memory (write to 0x01 or 0x02),
- sequentially write to TV/SIM memory the numbers comprising a vector (writes to 0x07 or 0x08),
- write to memory the value of 0x20000, signifying the end of the vector for the FPGA logic (writes to 0x07 or 0x08),
- reset TV/SIM memory (write to 0x01 or 0x02).

To exercise the test vector functionality, one needs to take the following sequence of actions:

- *load the test vector into TV memory on the VME board and simulation vector into SIM memory,*
- *set the frequency and frequency-dependent delays,*
- *reset both the Output and Resync FIFOs (write to 0x22)*
- *send the TV to the ABCD chip via Send TV command (write to 0x06),*
- *wait until the TV finishes running (TV_BUSY flags in the FPGA SR clears) and get the value of the difference in the same register. If a difference is found then the TV failed.*

The first two actions in this list can be swapped.

Table 6. Test Vector bits definition.

Bit	Control Line
0	com0
1	com1
2	tokenin0
3	tokenin1
4	datain0
5	datain1
6	resetB
7	masterB
8	select
9	id0
10	id1
11	id2
12	id3
13	id4
14	power_on_rst
15	test_clk
16	test_rstB

Table 7. Simulation Vector bits definition.

Bit	Control Line
0	datalink/LED
1	dataout0
2	dataout1
3	tokenout0
4	tokenout1

Since the TV/SIM vectors are resident on the VME board, one can run the same TV multiple times without reloading it.

As mentioned before, to achieve high speed the TV module uses the VME board FIFOs as fast buffers. The Send TV command triggers the following sequence of actions on the VME board:

1) test vector is loaded in the output FIFO from the TV memory,
2) the FIFO is opened for reading and the TV starts to get sent out,
3) concurrently with (2) the Resync FIFO is opened for writing and it starts to accept the incoming data from ABCD,
4) when the entire TV is sent out, the Resync FIFO is closed for writing,
5) TV comparison algorithm starts to read the Resync FIFO data looking for the valid header sequence ("011101") in the datalink/LED line bitstream,
6) as soon as the header is found, the algorithm starts reading the data from the SIM memory and comparing them with the Resync FIFO data (the TV difference bit is asserted if the header has not been found by the end

of the FIFO data stream),

7) the data comparison is finished if either of these three events occur:
   - a difference between the two data streams has been found,
   - the Resync FIFO data stream is finished,
   - the entire simulation vector have been read out from the SIM memory.

Caution: *the simulation vector must start very close to the header (with sequence of "011101..." on the datalink/LED line), since it is at that moment when the vector would be read out from SIM memory.*

The maximum TV size is about 256 K clock cycles.

The Table 8 contains the current list of test vectors, developed at CERN.

Table 8. Test Vectors description courtesy F. Anghinolfi. Also shown are stimulated I/O lines. Each TV has the resetB input line de-asserted soon after the beginning (not indicated in this Table).

TV #	Stimulated Input Lines	Affected Output Lines	Purpose/Description
1	com0	datalink	Configuration register Write/Read test. Bits 0 thru 10 are scanned.
2	com0, id0, id1, id2, id3, id4	datalink	BC counter test (all 8 bits are checked). ID address bits test. Overflow function and error code test.
3	com0	datalink	Data Compression Logic tests with random channel mask. Having "one"s in different bits of the 3-bit hit discription.
4	com0, com1, select	datalink	Digital pipeline test. Accumulate function test with Com1/clock1 circuit.
5	com0, tokenin0, tokenin1, datain0, datain1	datalink, tokenout0, tokenout1, dataout0, dataout1	Data/tocken bypassing circuitry test.

DAC/ADC

The purpose of this module is to measure some signals using 8-channel, 12-bit ADC placed on the connector board, and to set certain parameters using many DACs in the system.

The ADC channels are listed in Table 9. The first two channels measure the analog and digital voltages applied to the ABCD. The third channel measures temperature on the connector board. It is provided as feedback about the measurement environment. The main idea is to avoid damage to the electronics in case if the cooling fan stops working. The next three channels measure internal quantities of the ABCD via probe pads. The last two channels provide the values of analog and digital currents consumed by the chip by measuring the voltage across 5 Ω resistor placed in series with the ABCD.

The signals measured are amplified before the ADC. The Coeff column in the table gives the number one should multiply the measured value by to get the physical value, in ***mV***. Sometimes the quantity of interest is current. In these cases we also provide the resistance value one should divide the voltage by in order to get the current.

Table 9. ADC channels. Quantity (Measurement*Coeff) gives the physical value in mV.

ADC Number	Abbreviation	Coeff	R, Ω	Tcoeff, mV/deg. C	Final quantitiy , Units
0	Vcc	1.5			Meas*Coeff , mV
1	Vdd	1.5			Meas*Coeff , mV
2	Temperature	0.5		10	Meas*Coeff/Tcoeff, deg. C
3	Ipreamp	0.1	250		Meas*Coeff/R , µA
4	Ishaper	0.1	1000		Meas*Coeff/R , µA
5	Vthreshold	0.5			Meas*Coeff , mV
6	Icc	0.2	5		Meas*Coeff/R , mA
7	Idd	0.2	5		Meas*Coeff/R , mA

The power consumption on the chip P = I V. When the current (either the digital Idd or analog Icc) is measured, the power can be obtained by multiplying the value by the voltage. Under usual running conditions, the analog voltage is Vcc = 3.5 V and the digital voltage Vdd = 4.0 V.

To measure a quantity with ADC, one has to

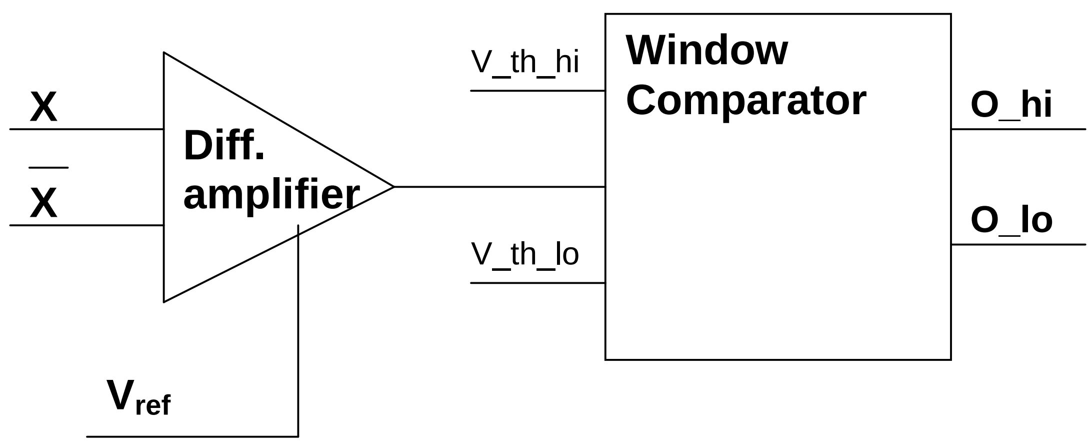
- issue a conversion for the corresponding ADC channel (write to 0xb), this has to be done twice for the measurement to be precise;
- read the data from the conversion (0xc).

DACs are set by a VME write to address 0x0a.
Functionally, they set the following quantities in the system:

- the levels of the input signals to the ABCD, provided by pin drivers on the connector board,
- the window comparators thresholds,
- the timing delays.

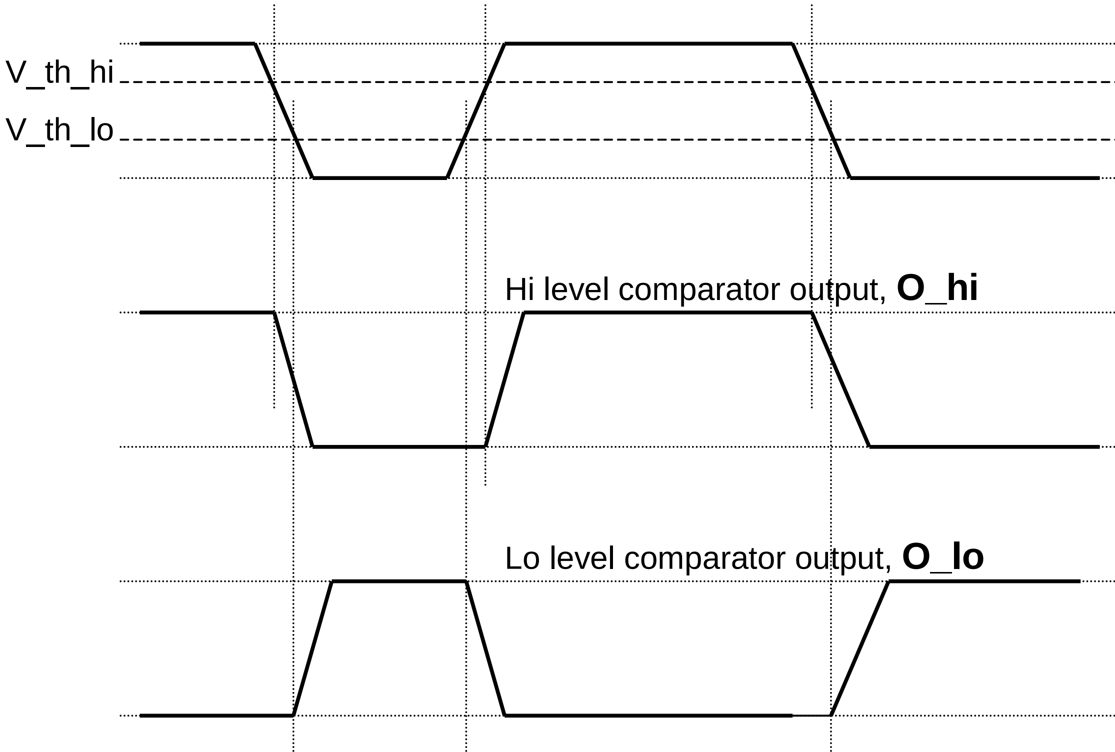
Window Comparators

The window comparators check the levels of the ABCD output. Each (differential) output signal passes through a differential amplifier (and thus becomes single-ended) before arriving to a comparator. The amplitude of the swing increases by the factor of **3**, due to the amplifier. The values of the resistors defining the amplification factor, have the nominal accuracy of **1%**. The "average" value for each signal is defined by the reference voltage applied to the amplifier. For the datalink/LED signal it is **0.29*Vdd**, for data/tokenout signals these are **0.5*Vdd**. The uncertainty of the coefficients is also defined by the accuracy of the resistors in the network.



Two comparators are provided for each signal. One of them has the positive output iff the signal level is higher than its ("high") threshold. The output of the other one is high iff the signal level is below its ("low") threshold. Inside the FPGA, both the output of the first comparator and the negated output of the second comparators are checked against the expectations when running the test vectors. The signal will be valid iff both data streams match the expectations. The picture below illustrates the scheme. For clarity, the signal distortion is greatly exaggerated in this diagram.

The signal used in the comparison is inverted.


V_th_hi
V_th_lo
Hi level comparator output, **O_hi**
Lo level comparator output, **O_lo**

A note on BUSY signals

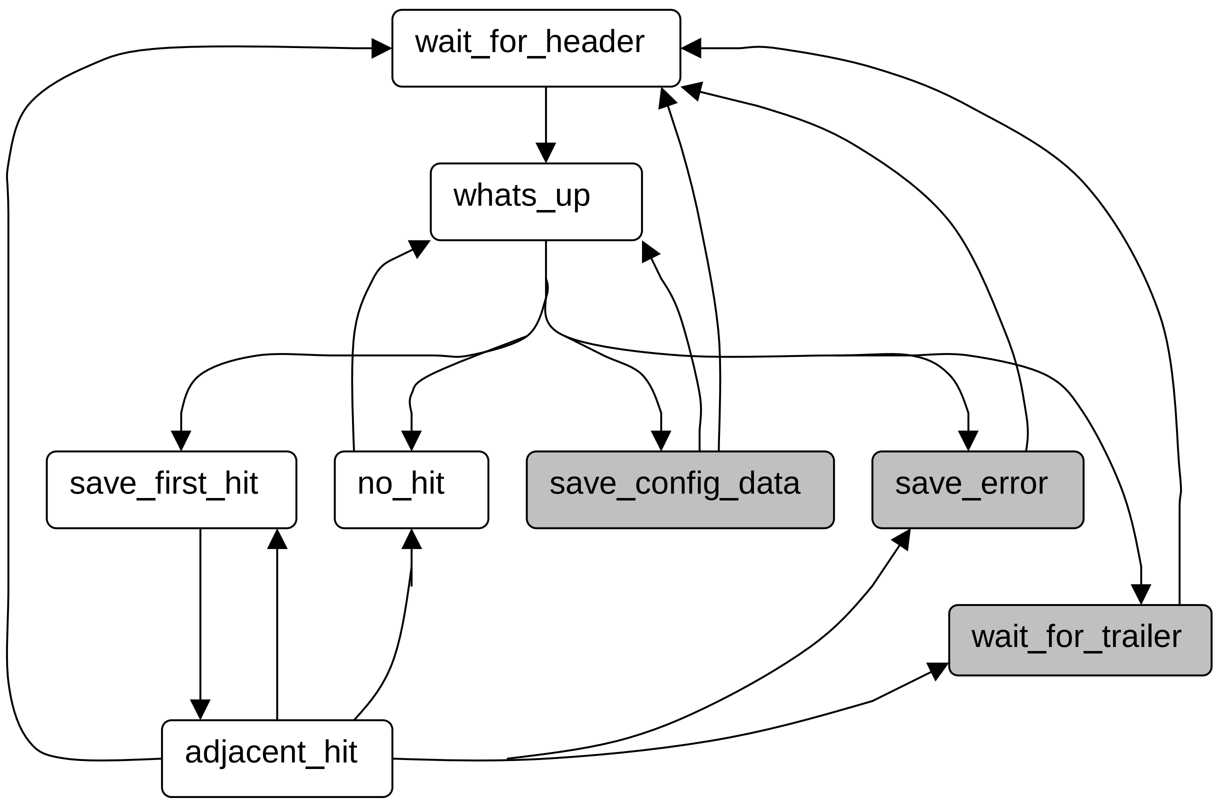
The "Histogramming", "TV", "ABCD commands", "Frequency" and "DAC/ADC" modules inside the FPGA are all independent from each other, with the caveat that the first two use the same I/O lines. One can use DAC/ADC functionality while doing other things. All modules have state machines, which start going through a cycle after the corresponding signal from VME. The cycle is always completed. If a second VME command addressing the ***same*** module happens too soon, then it would be ignored. The minimal measured time between consecutive VME bus accesses is 1.8 μs. This corresponds to 72 clock cycles and is enough for most instructions to be completed. There are four notable exceptions, which may take more time:
1) "start sending triggers and histogram the data" (0x0e),
2) "clear histogram memory" (0x03),
3) "send test vector" (0x06),
4) "send convert signal to ADC" (0x0b).
To provide feedback about the internal states after the four commands, the BUSY signals were introduced. The user must wait for the corresponding BUSY bit to clear after any of the four commands.

Datastream Decoding

The state machine (SM) for interpreting the data coming to the FPGA during a threshold scan is shown below in a simplified form. The purpose of the algorithm is to extract the channel numbers containing hits from the raw data packets (DP). The SM can also recognize the *configuration* and *error* data format, although they are not expected to appear during the scan. In case when the data do not match the considered cases of

- *physics DP,*
- *no hit DP,*
- *configuration DP* or
- *error DP*

the SM goes into wait_for_trailer state, where it stays until the end of the data. It should not go there in case of correct (unscrambled) data stream.



Since only *physics DPs* are expected during a scan, we provide a register (read from 0x26), which has bits indicating if the SM has gone through

- *wait_for_trailer* state (bit 0), or has seen
- *error DP* (bit 1) or
- *configuration DP* (bit 2),

indicated by the gray boxes on the diagram. An asserted bit value is "1". The bits are cleared (initialized to the default zeros) by the *Send Triggers* command. The values are available for the readout after the histogramming process is done and the corresponding BUSY bits are cleared in the FPGA status register.

A Note on ID Lines

When running the test vectors, the ID lines are being controlled by the test vector content, as indicated in Table 6.

When executing any of the commands in the Table 2 (for instance, when running a threshold scan) all ID lines are set to zero. Therefore, the chip address value has to be 32 (= 0x20) for those commands. It is not zero due to the existence of internal bit 5, which is pulled up to high value.

How to program a Scan

A sample sequence of actions issued for a threshold scan is listed below.

Important: Before you issue any command, check with the "read status register" command (x00) that the busy flag is not set (bit 16)!

1) Initialize Module

- Set Frequency to 40 MHz (x0F)
- Set DACs for the power, pin drivers voltage and window comparators
- Load Mask Registers
- Load Trim DAC's of all channels (x1E).
- Load Strobe Delay Register (x18)
- Load Bias DAC's (x1D)
- Load the ABCD Configuration Register

2) Initialize Scan Parameters

- Clear Histogram Memory (x03)
- Set Number of Triggers per scan step ("burst") (x11)
- Set Trigger to Trigger delay (x10). Make sure you allow enough readout time.
- Set Strobe to Trigger delay (x1F). This should be around 129.
- Enable Calibration Strobe.

3) Do the Scan

Loop over the following 5 commands (i = [0..number of scan points-1])
- Enable Data Taking
- Set the scan parameter (eg. threshold) to the i-th scan point (eg. x19)
- Set Base Address to the actual step number i (x05)
- Send burst of triggers (x0E)
- Wait while the Histogramming Module is busy

4) Read Histogram Memory

Loop over the following sequence for each scan point:
- Set base address to the value for this point (x05)
- Read from Histogram Memory (x04) the number of channels you want (128 times if all channels are enabled).

How the VME Interface Works

1) After startup of the FPGA, the mask registers and the compare registers of the CY964's are loaded. The mask registers are all set to 0, and the compare registers are set to:

31..24 = board address (DIP switches)
24..0 = 0

This involves exercising the pins LDS and STROBE* during startup. MWB* is always set to 1. LDS is later connected to the LDS pin of the 960. For the timing, see p 4-8 of the databook. Writing to the compare reg. clears the mask reg. A 0 in the mask reg. means that the bit is used for comparison.

2) The 964's respond by asserting VCOMP* if they're hit by an address which matches the compare register.

3) the VCOMP*'s go to the FPGA. The FPGA outputs SVIC_REGION(3 downto 0) are just the inverted VOMP*(3 downto 0).

4) Since we set the board address on bits 31..24 which corresponds to vcomp*(3), only region "1000"=8 is enabled in the setup for the CY960.

There is a setup program, called WinSvic, which creates the bitstream for the configuration PROM of the CY960, The Program is available on the CYPRESS Website, and is installed on the PC in the office 50B-6220 in the folder C:\hubert\new_DAQ\winswic.

The CY960 does 3 things:

a) It creates a programmable chip-select pattern cs(5..0) from the region(3..0) inputs.
b) For each of the 16 possible region-codes, it can be told which types of VME transfers are allowed.
c) Corresponding to the type of transfer, the data byte enable DBE(3:0) signals are asserted. The DBE signals come later than the CS.

In case of a VME access to the board address set with the DIP switches, the cs(0) will go high and then also some of the DBE, depending on the access. In case of a 32 bit transfer, all DBE's should go high.
I only check for a coincidence of at least one CS and at least one DBE for 2 clock cycles, and then latch data, address and R_W into the FPGA. The 960 then waits for LACK* to go low to acknowledge the data transfer. So in case of a write access, the LACK* is immediately asserted; in case of a read access, it is asserted when the valid data is on the bus.

At the moment, the 960 is configured in the following way:

i) Initialization: Serial PROM Method (on power up, it loads its config data from PROM)
ii) Configuration=IO

iii) IO Menu:
Only for REGION=8 , the chip select cs(0) is activated. It's set to active HIGH.
The DBE assert time is set to 18 clock cycles. (It's the minimum width in case of self-timed access. Since we hand-shake the data transfer, it's not so important, but must certainly be long enough to be detected by the FPGA).

a) AM Code Menu: Only for region 8: all access modes allowed, all other regions: no access allowed
   b) None of the special modes are allowed. (No lock etc).

Miscellaneous control:

a) decode delay: this is the time in clock cycles which the 960 waits upon receipt of a AS* on the VME-bus (that's the signal on the VME bus which tells the 960 that a valid address is available) before sampling the region inputs. We set it to the maximum of 5 cc. (That's 100 ns at 80 MHz..).
b) DBE Polarity. I don't use the convention of the databook: I use DBE=active HIGH.
c) AM Code LA bit off. See p 3-48
d) IRQ level=2. Irrelevant since we don't use interrupts.

   e) Bus holdoff=off, see p 3-51.

   f) IACK LACK response = off. See p 3-62

   g) Master interlave=off (only 961?) See online help in winsvic!

The configration file is called **modules.sv**

Table 10. FPGA pins related to the vme interface.

Signal name	FPGA In/Out	Active	Description
laddr(31:1)	in		local address. Bits 31..24 match the board address for a valid access.
ldata(31:0)	in/out		32 bits of data. Bidirectional.
vme_xcvr_mwb_n	out	low	tie high. Not actively used.
vme_xcvr_lds	out	high	lds signal for the 964. Connected to svic_lds during normal operation. During startup, used to select mask/compare register for loading those.
vme_xcvr_strobe_n	out	low	used to load mask/compare reg on 964 during startup
board_addr(7:0)	in		board address, set by 8 DIP switches
svic_region(3:0)	out	high*	Since only region 8 is enabled in the 960, svic_region(3)=not vcomp(3). svic_region(2 downto 0)=”000”
svic_lack_n	out	low	used to acknowledge local data transfer. Immediately asserted after write access, asserted after valid data has been put on ldata bus after a read access.
svic_lds	in	high	must be tied to vme_xcvr_lds during normal operation.
svic_lirq	out	low	must be high always. Not used.
vcomp(3:0)	input	low	result of address comparison in the Cy964’s. Only vcomp(3) is used.
svic_dbe(3:0)	in	high*	these bits go high during a valid vme access corresponding to the type of vme access (32 bit, 16 bit etc).
svic_cs(5:0)	in	high*	only cs(0) is enabled. See description above.
svic_lden_n	in	low	not used.
svic_pren_n	in	low	asserted during initialization. Not used.
svic_swden_n	in	low	swap data enable. Not used.
svic_r_w*	in	--	write=low, read=high
svic_strobe	in	high	only used if 960 configures 964 on power up. Not used in this design.

- *the polarity is programmable in the CY960 during configuration*