VLBA Technical Report No. 46
THE VLBA CORRELATOR DIGITAL FILTER
VOLUME 1 of 2

Joseph Greenberg

February 1, 1999

firmanv1.doc Microsoft Word not under SCCS control

Master file stored in /corrdwgs/manuals/firman/SCCS
(where the SCCS file would have been stored)

This page intentionally left blank

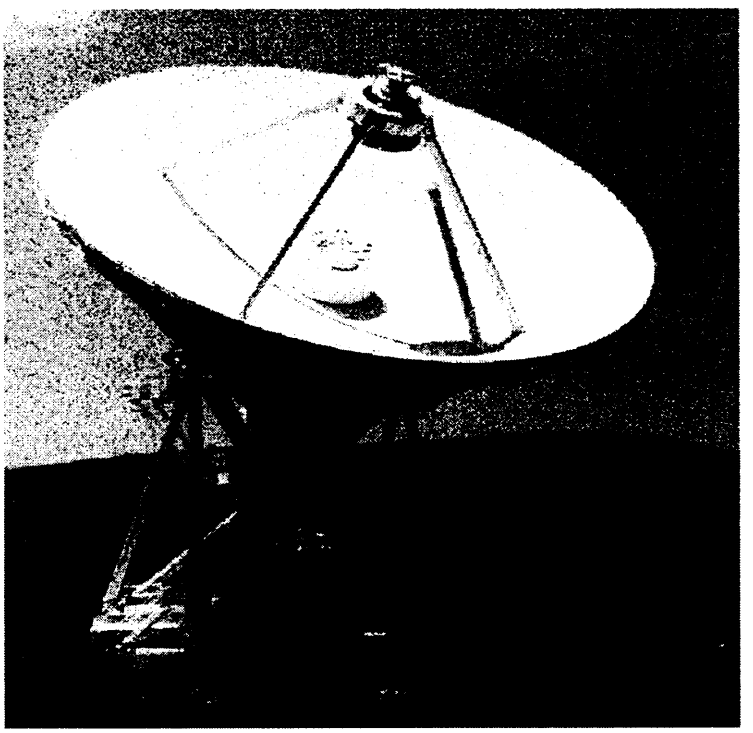


TABLE OF CONTENTS

TABLE OF FIGURES

1 Introduction

1.1 Functions of the FIR Filter Card

The FIR Filter Card (FIR) has two main functions. In normal operation, the FIR is an interface between the LTA (Long Term Accumulator) and the VME system. In this case, the FIR reads results from the LTA and transfers them unaltered to the VME system. Alternatively, the FIR can be operated as a Finite Impulse Response Filter that filters the data from the LTA before passing it on to the VME system. The primary purpose of this filtering is to reduce the output data rate while retaining fringe frequency response.

The interface from the FIR to the VME system is by way of a commercial DMA controller card installed in the VME chassis. This is the Ironics IV-3272 card with the IV-3272-PIO Parallel I/O Daughter Board installed. Signals between the FIR and the Ironics are buffered on the Transition Module for Ironics IV-3272, NRAO schematic drawing number 56000L044 (Orcad file name L044D01.SCH). This transition module is located in the rear section of the VME chassis, with the other system transition modules.

Each spectral point, of each baseline requested, is filtered. The data streams are filtered in time, where one unit of time is one integration period. The minimum LTA integration period is 131 ms.

2 The FIR Card

2.1 Top Level FIR Block Diagram Description

For the following discussion, refer to 1031d25.sch in Volume II, the Top Level FIR Block Diagram. This diagram shows the overall data and address flow in the FIR.

2.1.1 Reading Data from the LTA

The LTA is read out, one complex result at a time. A complex result consists of two, 32 bit IEEE floating point numbers, real and imaginary. A 19 bit address is supplied to the LTA. The address consists of an eight bit baseline (0-255), and an eleven bit result (0-2047).

The FIR operates in a batch mode, in that it is given a list of instructions to process every 131 msec. Once the instructions are processed, the FIR is idle, until given more work. The instructions are queued up in Fifos on the FIR.

An instruction consists of the following;

The LTA baseline to be read.
The 'modulus' which tells how many results are in the baseline.
The 'tap weight index' telling which tap weights to use.
The 'stoploop bit' signaling the end of the list of baselines.
The 'DRAM baseline' specifying the FIR memory baseline to place the results in.

The modulus controls a programmable counter, to count out the number of results assigned to that baseline. This eleven bit result field becomes part of the address to the LTA. A sequencer controls the sending of the addresses from the FIFO to the LTA, along with a RQSTSTB\ signal to request reading the LTA. The IEEE complex results from the LTA are strobed into an FIR input fifo, by the signal

FIRSI\. See HCB Function 16, LDFIFOS, in Appendix I, the HCB Protocol, for more details.

2.1.2 Processing the LTA Data

2.1.2.1 Straight thru Mode

The simplest mode is Straight thru Mode. The data is passed straight thru the 29C327 Floating Point Chip (FPC). This is accomplished by setting the tap weights equal to one. The output of the FPC is captured in the Snapshot Register. The Snapshot Register output is transferred to the Output Buffer. The Output Buffer is a double buffer in DRAM, labeled R8,I8,R9,I9 on the diagram. Each half of the Output Buffer has storage for 128 baselines of 2048 results each. Depending on the state of BLMSB, each baseline can be broken in half to form two baselines of 1024 results, providing storage for 256 baselines of 1024 results each.

The address for writing into the Output Buffer is generated as follows. The DRAM Baseline FIFO supplies the baseline number, and the control bit, BLMSB. BLMSB=1 for 256 baselines and 1024 results. BLMSB=0 for 128 baselines and 2048 results. The result portion of the address comes from the same counter that generated the result address to the LTA. A FIFO delays the result field for timing. The Output Buffer address comes from a multiplexer, which chooses the correct address source. The opposite half of the double buffered Output Buffer is read by the Real Time System (RTS). This is done via the Ironics Card. The Output buffer is read into a FIFO for timing purposes. The Ironics Card drains the FIFO. This will be described in more detail in Section 11.

2.1.2.2 Filter Mode

Figure 1 presents an overview of how the FIR digital filter works.

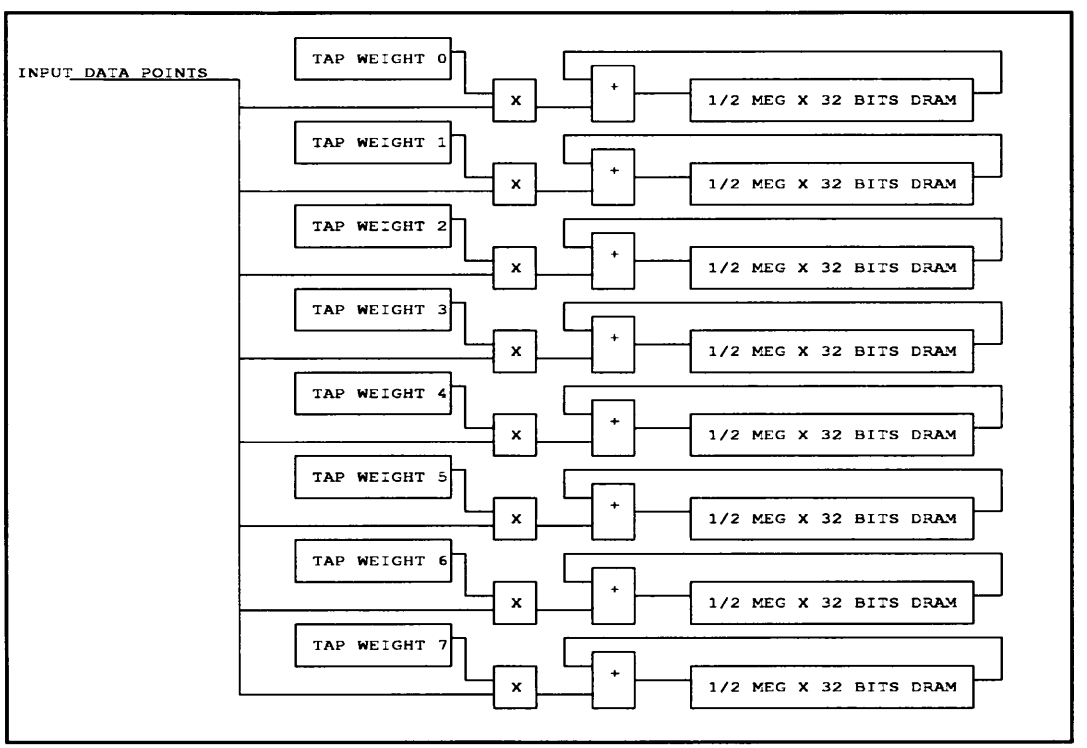


Figure 1 Basic FIR Structure

Each input data point is multiplied by eight distinct tap weights. A running sum of these products is stored in a distinct ram. This is done for all the results of all the baselines being processed in the 131 ms time tic. The eight ram sections correspond to R0/I0 through R7/I7 on the right edge of the block diagram, 1031d25.sch.

Actually, the tap weights 0 through 7 are reloaded from the microprocessor for each time tic. This reloading allows different values of tap weights to be used each tic. For an N tap filter, the N tap weight would be rotated through in N tics. After N products have been summed in ram, the value can be read out as the filtered result. After that, the CLAC signal clears the accumulator and the process repeats. Notice the data rate can be reduced, since data is only output every N tics. By having the process going on concurrently in 8 ram banks, with the tap weights used staggered among the ram banks, we multiply the output data rate by 8. Thus, the decimation factor is equal to N/8. The decimation factor is the amount the data rate has been reduced, and is shown below for each value of N.

N tap Filter	Decimation Factor
8	1
16	2
32	4
64	8

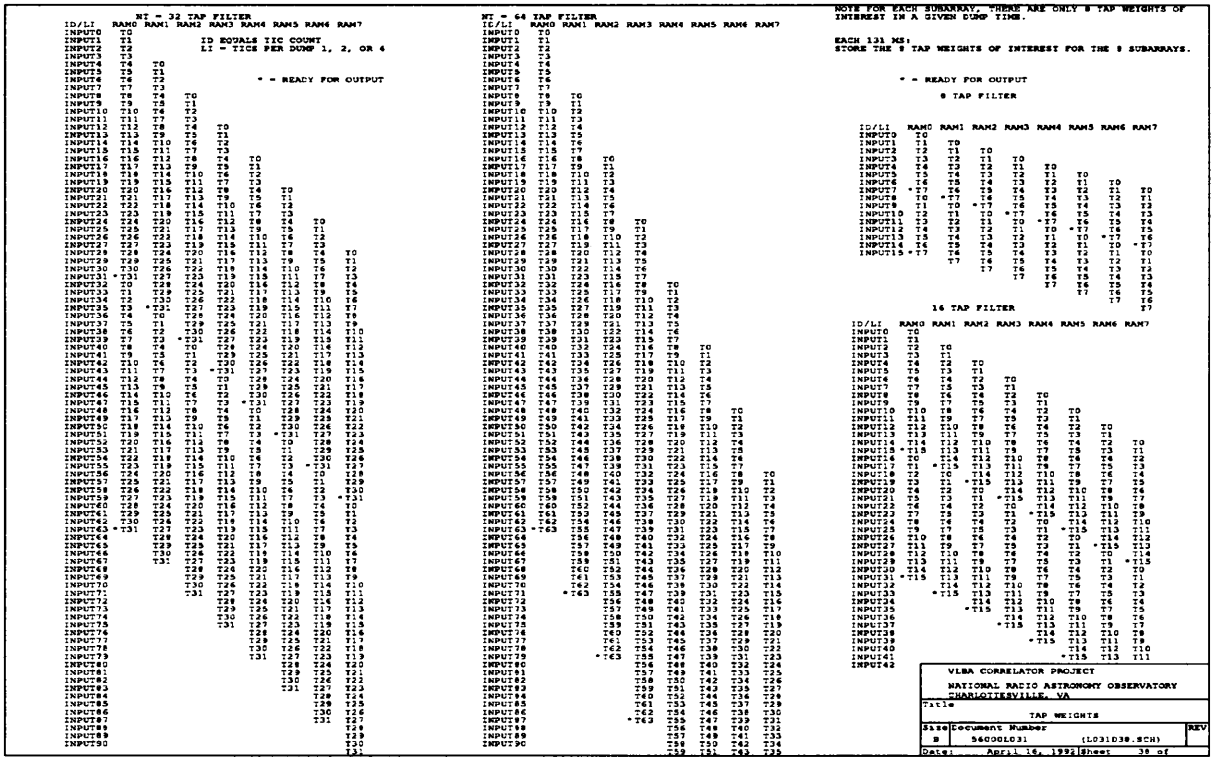


Figure 2 1031d38 Tap Weights

L031d38.sch here shows the tap weights used for N=8, 32, and N=64.

For the 8 tap filter on the right, each ram uses tap weights T0 through T7 over eight tics. Notice the staggering of tap weights between rams. When each ram has had the data points multiplied by T7 added in, then that ram's data is ready for output, as indicated by the *. After the initial eight tics, there is data output every tic.

For the 32 tap filter on the left, ram0 shows the rotation of T0 through T31. The * at T31 indicates the summation is ready for output. In Ram 1, we see the tap weights occurring 4 tics later. The next point output would be from Ram 1 when it gets to T31. This yields a decimation by four.

The factor LI is the tics per dump. For simplicity in the above explanation, 1 tic per dump was assumed. However, LI=2 or LI=4 are allowed. For LI=2, baselines are processed every other 131 ms tic. The basic batch cycle of the FIR is always 131 ms. However, we can do different baseline lists each 131 ms, to give an effective LI of 2 or 4. Alternatively, the FIR could be idle for some of the tics. The batch nature of the FIR allows the RTS full control of the work load distribution to the FIR. The RTS must be careful not to give the FIR more work than can be done in 131 ms.

The following table shows an example of distributing the FIR workload when there are multiple subarrays.

SUBARRAY EXAMPLE:
SUBARRAY 0 FILTERED THRU AN 8 TAP FILTER 16 BASELINES, DUMPED EVERY 131 MS. 16*2048 RESULTS*8 BYTES /.131 SEC = 2 MBYTES/SEC
SUBARRAY 1 IN STRAIGHT THRU MODE 136 BASELINES, DUMPED EVERY 20 * 131 MS. 136*2048 RESULTS*8 BYTES /20*.131 SEC = 0.85 MBYTES/SEC DUMP 6 OR 7 BASELINES PER CYCLE
4 MBYTES/SEC AVAILABLE FROM LTA. VS (2 + 0.85) MBYTES/SEC NEEDED
DUMP SUB 0 BASELINES 0-15 DUMP SUB 1 BASELINES 0-5
DUMP SUB 0 BASELINES 0-15 DUMP SUB 1 BASELINES 6-10
DUMP SUB 0 BASELINES 0-15 DUMP SUB 1 BASELINES 11-16
DUMP SUB 0 BASELINES 0-15 DUMP SUB 1 BASELINES 17-23
DUMP SUB 0 BASELINES 0-15 DUMP SUB 1 BASELINES 24-30
DUMP SUB 0 BASELINES 0-15 DUMP SUB 1 BASELINES 31-35
DUMP SUB 0 BASELINES 0-15 DUMP SUB 1 BASELINES 36-42
DUMP SUB 0 BASELINES 0-15 OUTPUT INITIAL FILTERED SUB0 DUMP DUMP SUB 1 BASELINES 43-49

2.1.2.2.1 Tap Weight Ram Loading

Return to 1031d25.sch in Volume II, the Top Level Block Diagram. The Tap Weight Ram is the 2Kx32 Ram in the center of the drawing. The tap weights are loaded into the ram via HCB Function 10, DWNLDTAPS (Refer to Appendix I, the HCB Protocol). The tap weights are stored as 32 bit IEEE floating point numbers. They are written via the Microprocessor Write register, which parallels the data path from the LTA. The data goes, via the FIFO, to the tap weight ram.
The 11 bit address to the tap weight ram has the 4 msb's pulled high. In the remaining 7 bits, there is a 4 bit Tap Weight Index(TWI). This allows 16 different versions of the tap weights to be stored in the ram at once. The RTS could load a version of the tap weights, in a non-active TWI ahead of time. A 3 bit address

from the sequencer chooses among the eight active tap weights. This same address bus is used for reading the tap weights when the filter is active.

2.1.2.2.2 The CLAC Ram

The Clear Accumulator (CLAC) Ram shares its address bus with the 7 active bits of the Tap Weight Ram. It is also loaded from the microprocessor. The ram provides these three control bits:

CLAC - Controls an instruction to the floating point chip, to zero the ram accumulator contents, coincident with the T0 tap weight.

SNAPCE\ - Snapshot Register Clock enable causes the Snapshot Register to capture the data which is ready for transfer to the output buffer. That is the data which has the complete summation of tap weights times data points.

SNAPOE\ - Snapshot Register Output Enable puts the Snapshot Register stored data on the bus for transfer to the Output Buffer.

2.1.2.2.3 The DRAM Address Select Multiplexer

The address multiplexer provides the addresses to the DRAMs. One function of the multiplexer is to break the 20 bit DRAM address into the two, 10 bit sections (ROW and COL) the DRAM requires.

The Multiplexer supplies DRAM addresses for the following purposes:

1) The DRAMs can be written to or read by the microprocessor.
2) The DRAMs can be written with results from the LTA.
3) The DRAMs can be read by the backend logic.
4) The DRAMs are systematically refreshed.

3 Overview of the FIR Card

3.1 Data Rates

3.1.1 LTA to FIR Data Rates

Each dump from all the MAC cards, every 131 ms produces 210 baselines times 256 spectral points per baseline times 8 channels. This results in 430,080 complex floating point numbers per dump, or 860,160 single precision IEEE values per dump, or 3,440,640 bytes per dump. These values are integrated in the LTA.

The LTA has a maximum output rate of 1 complex floating point number / 2.0156 usec or 3.969 Mbytes/sec. At least one 2 usec period per 16 usec fft cycle is needed for refresh. This degrades the LTA maximum rate to 7/8 * 3.969 = 3.473 Mbytes/sec.

The LTA transfers one complex point to the FIR every 2 usec, if requested. The FIR software uses 5, 2 usec cycles to process 4 data points. This degrades the maximum input rate to 4/5 * 3.969 = 3.175 Mbytes/sec. This rate allows the LTA a refresh cycle every 5, 2 usec cycles. This satisfies the requirement that a refresh cycle be allowed every 8, 2 usec cycles. The LTA imposes an up to 20 ms. pause each tic for house keeping. This can degrade the input rate by 111ms/131ms to 2.7 Mbytes/sec. A 32 tap filter would decimate this to 0.67 Mbytes/sec.

3.1.2 FIR Output Data Rate

The maximum output data rate of the fir loop is determined by how much work the fir is given each tic. A 2 usec period, which provides backend access, gives 7, 4 byte reads in the 2 usec. This is based on one backend ram read every 250 ns, for 1.750 usec. The extra 250 ns cycle, in the 2 usec pause, is used to refresh all DRAM's. When the fir is filtering, it uses the above 10 usec cycle, which contains one 2 usec output cycle. This gives 2.8 Mbytes/sec maximum output rate while running the 10 usec loop. During the portion of the tic, in which the fir is not filtering, a 4 usec idle loop is run. This has one, 2 usec output cycle every 4 usec, for an output rate of 6.9 Mbytes/sec. See 1031d14 in Volume II for the sequence.

Each complex number gets multiplied by 8 real tap weights, and accumulated in 8 separate ram banks. Each output is the weighted sum of 32 inputs. For the output data rate to be ¼ the input, 32/4=8. i.e. in order to store 32 points worth, when we output every 4^{th} dump, we need 8 ram banks. Each ram bank size is half the maximum dump size or 1,720,320 bytes.

3.2 Basic FIR Program Loop

The following shows the basic 29C327 program loop for filtering data.

The 16 clock cycles represent 2 usec at a 125 ns clock period.
T1,T2... represent the tap weights for successive filter banks.
ACCr1,ACCr2... represents the accumulated value from memory.

This first section has 16 clock cycles to load R0-R7. The validities are input, then the 4 complex values are input, normalized and stored. R IN and S IN straight thru give a window for a microprocessor read of S & R.

CLOCK	R-INPUT	S-INPUT	INSTRUCTION INPUT	SOURCE OUTPUT
0	DUMMY	INSTRUCTION	FOLLOWING LAST MAC	76 T4*R7+ACCi28
1	VAL0		STRAIGHT THRU	77 T5*R7+ACCi29
2	VAL1		STRAIGHT THRU	78 T6*R7+ACCi30
3	VAL2		STRAIGHT THRU	79 T7*R7+ACCi31
4	VAL3		STRAIGHT THRU	1 R4=VAL0
5	RE0		RE0*R4	2 R5=VAL1
6	IM0		IM0*R4	3 R6=VAL2
7	RE1		RE1*R5	4 R7=VAL3
8	IM1		IM1*R5	5 R0=RE0*VAL0
9	RE2		RE2*R6	6 R1=IM0*VAL0
10	IM2		IM2*R6	7 R2=RE1*VAL1
11	RE3		RE3*R7	8 R3=IM1*VAL1
12	IM3		IM3*R7	9 R4=RE2*VAL2
13	R IN		STRAIGHT THRU	10 R5=IM2*VAL2
14		S IN	STRAIGHT THRU	11 R6=RE3*VAL3
15				12 R7=IM3*VAL3

NOW FILTER 32 RESULTS			OUTPUT =
16 T0	ACCr0	T0*R0+0	13 R IN
17 T1	ACCr1	T1*R0+ACCr1	14 S IN
18 T2	ACCr2	T2*R0+ACCr2	
19 T3	ACCr3	T3*R0+ACCr3	
20 T4	ACCr4	T4*R0+ACCr4	16 T0*R0+0
21 T5	ACCr5	T5*R0+ACCr5	17 T1*R0+ACCr1
22 T6	ACCr6	T6*R0+ACCr6	18 T2*R0+ACCr2
23 T7	ACCr7	T7*R0+ACCr7	19 T3*R0+ACCr3
24 T0	ACCi0	T0*R1+0	20 T4*R0+ACCr4
25 T1	ACCi1	T1*R1+ACCi1	21 T5*R0+ACCr5
26 T2	ACCi2	T2*R1+ACCi2	22 T6*R0+ACCr6
27 T3	ACCi3	T3*R1+ACCi3	23 T7*R0+ACCr7
28 T4	ACCi4	T4*R1+ACCi4	24 T0*R1+0
29 T5	ACCi5	T5*R1+ACCi5	25 T1*R1+ACCi1
30 T6	ACCi6	T6*R1+ACCi6	26 T2*R1+ACCi2
31 T7	ACCi7	T7*R1+ACCi7	27 T3*R1+ACCi3
			28 T4*R1+ACCi4
			29 T5*R1+ACCi5
			30 T6*R1+ACCi6
			31 T7*R1+ACCi7

CLOCKS 32 TO 47 REPEAT ABOVE USING R2 & R3.
CLOCKS 48 TO 63 REPEAT ABOVE USING R4 & R5.
CLOCKS 64 TO 79 REPEAT ABOVE USING R6 & R7.

NOTE: A 0 IS ADDED ONLY WHEN CLAC IS USED TO CLEAR ACCUMULATOR.

This has a cycle of inputting 4 complex points and their validities in 2 usec. Then 8 usec is spent MACing these to the 8 rams. The program timing and addressing to the 8 rams is fixed and set by the sequencer. See L031D32 in Volume II for more detailed timing.

The actual sequencer code is in vlbsoft/pal_prom/proms/firfpc/pg0.wk3, a Lotus worksheet. The worksheet is divided into 3 pages. Page 0 is test and discrete functions. Page 1 is the main loop processing, defined above. Page 2 is the validity loop processing.

The program fircode.c translates the printout of the above worksheet, fir.prn, to the binary files to be downloaded into the discrete rams. The command line is: fircode fir.prn. At the filter maximum input rate of 3.175 Mbytes/sec, it takes 542 ms, or 4.14 131 ms cycles, to fill a FIR ram bank of 1,720,320 bytes. Thus, if the dump time is 4 cycles, the maximum input rate can be sustained for 4 cycles without filling up the ram. Dump times of 4, 2, or 1 cycles will be allowed through the filter. Longer dump times can be used in straight thru mode.

3.3 Straight Thru Mode

In straight thru mode, there are unity tap weights downloaded into the tap weight ram, as specified by the microprocessor routine LDTWBYSA. Whatever goes into ram bank 0, also is written to the output buffer. We do not need to worry if the total dump is bigger than the output buffer size, since only a portion of it can be read out each tic anyway. The specified LTA and DRAM baseline numbers, being different, allows translating from the LTA memory bank, to the FIR memory bank. An FIR memory bank is half the size of an LTA memory bank. The information stored in the other eight, FIR DRAM banks, is of no interest in straight thru mode.

3.4 Backend Buffer

The output buffer to the backend consists of a 32 bit x 1 Meg DRAM. This is divided into 2 banks. One is output to, while the other is read by the backend. The bank switching occurs every 131 ms cycle, by P1.6 beBANK from the microprocessor. For a 32 tap filter, the filter outputs every 4^{th} 131 ms cycle, for a 131 ms dump time from the LTA. The bank will switch, and the backend will have the subsequent 131 ms cycles to read out these results. The bank will again switch, and nothing new will be written. The bank will switch back, and results could be read if they had not all been read out two tics ago.

3.5 Subarrays

To handle subarrays, using different dump times, the fastest dump time subarray will be completely output each of its dump times. Portions of the slower dump times will also be output. Example:
Subarray 0 is baselines 0-15, dumping every cycle.
Subarray 1 is baselines 16-46, dumping every 2 cycles.
Then successive cycles might be:
Cycle 0: output 0-15 of Sub 0, output 16-31 of Sub 1.
Cycle 1: output 0-15 of Sub 0, output 32-46 of Sub 1.
Then repeat.

Figure 3, which follows, shows another example of dividing the work of the subarrays among the sequence of tics.

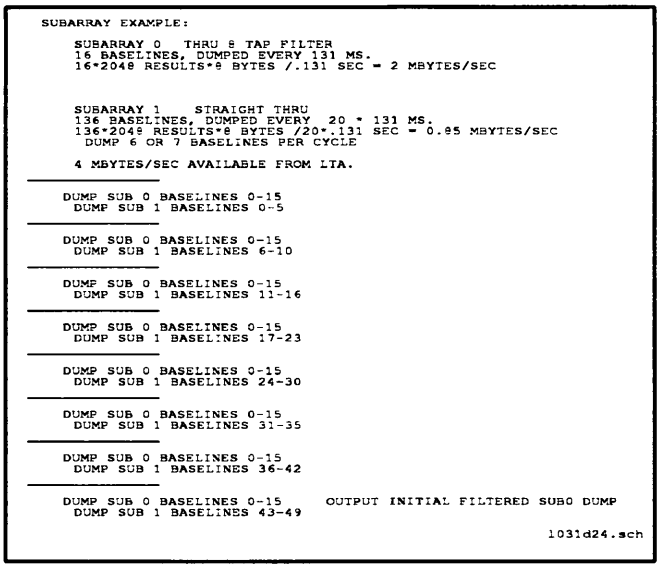


Figure 3 FIR Subarray Example

A more comprehensive example can be found on 1031d23 in Volume II.

It does not matter that the portions move around in the buffer, since the backend will figure it out. Since the output buffer and filter rams share an address bus, the results must go in the same address, in the respective rams. In the case where a subarray is in Straight Thru Mode, then the portion of the result can be placed wherever desired in the output buffer, provided there is not a conflict with other subarrays. For a subarray bigger than the half size output buffer, the portion output each dump could be placed in the same spot in the output buffer. The Real Time System (RTS) will have to keep track of which portion is which. Note that unless we reload the result ram, the baselines must be placed in spots designated for the correct subarray.

1,720,320 bytes represents 105 baselines of 2048 results. Since the bank size is 2 Mbytes, 128 baselines could be stored if desired. Alternately, 256 baselines of 1024 results could be stored.

3.6 *Validity Counts*

V is the tape validity count on a baseline and channel basis. That is the number of valid FFT cycles in a LTA dump. If either tape drive of the baseline was bad, it inhibits the count. For a given baseline and channel, we can do all 256 spectral points, without this changing. The validity comes in from the LTA as IEEE 32 bit floating point numbers.

I is the sub-array integration count. That is the length of the LTA integration time, in units of FFT cycles. V is the number of valid FFT cycles. V=I if all test good. It is not necessary to multiply by I since it is a constant for an observation.

Sx is the output of the short term integrator for a given spectral point x.
Normalized spectra = [Sx]*I/V = Nx
Filter Fx = Summation i=0 to 31 WiNxi
Where Wi are the 32 tap weights.
Nxi is subsequent dumps of a given spectral point x.
The validity counts come over the same 32 bit data bus as the LTA dumps. This data bus has a max data rate of a complex point of 2, 32 bit values, every 2 usec. If the LTA is dumping at its maximum rate, there is not time to send over the validity points. There is redundancy in the validity counts. For the tape validity, each count is good for 256 spectral points. 8 channels * 256 points = 2048 results. Thus it pays to store the counts in a ram, to reduce data transfer from the LTA.

3.6.1 Normalizing the Data, Using Validities

Dividing by validities is provided as an option for normalizing the data. A TRW TMC3210 floating point 32 bit divide chip is used to do the floating point divides for the validities. APPENDIX III TRW TMC3210 CMOS Floating Point Divider contains the data sheet for the TMC3210. L031d10 in Volume II shows divider timing, and the block diagram. The divide chip has glue logic to convert its 16 bit bus to a 32 bit bus. Currently, the divide chip normalization function is not used. Instead, the validities are filtered by the FIR.

If the lsb's of the validity count are thrown away (Ex. 20 instead of 24 bits), then the floating point representation will still not lose any precision until the larger count. This is since 5 is represented .0100000000 (with a phantom bit) where it does not matter how many zeros follow. For high validity counts, the I/V factor is near 1 and does not change rapidly, so the loss of precision is washed out.

If the validity count is 0, the divide chip will give a 1/V of infinite. A Nan is desired for the final output. By setting the M5 bit on the 327 low, the outputs for dealing with infinities become Nans.

3.6.2 The Validity Ram

The validity ram provides the storage for validities, so they can be reused for all 256 spectral points. To load the validity ram, the LTA baseline is jammed to 210, which is where the LTA stores the validities. The jamming is done by a pal, on the output of the FIR address to the LTA. In the FIR, the LTA baseline fifo and result counter increment the address, while the data comes from the LTA. The output of the LTA baseline fifo is switched to the 8 msb's of the result field. This picks out the validities for a specific baseline from the special LTA baseline (210). See the block diagram L031D25. This result, with embedded baselines, also goes to the DRAM addressing, to allow the filtering of validities while reading them in.

To load all 1680 validities, at a rate of 1 per 2 usec takes 3.36 ms. That is 2.6% of a 131 ms cycle. Time could be saved by loading only the validities needed for that particular cycle. That is for only the baselines dumped.

While processing LTA data, to read the validities out of the validity RAM, the delayed baseline from the LTA request forms part of the address. The remaining 3 bits use the 3 lsb's of the delayed LTA result counter. These 3 bits are now indicating channel instead of result. A separate delay fifo is used for these 3 bits, in parallel with the delay fifo sending an address to the DRAMS. That is because in the program loop, first the 4 validities are read into the 29C327 at 8 MHz, along with the 4 complex data points from the input fifo. Then, in doing the filtering, the 4 DRAM addresses are accessed. Note each DRAM address is good for 16 cycles. That is 8 real macs and 8 imaginary macs. The IMAG bit from the program sequencer toggles to show the difference. The baseline will not be changing through all this, so a common baseline fifo could be used. However, a separate DRAM baseline fifo is already used for reordering. If we are in less than 8 channel mode, duplicate validities will be in the validity ram, so the msb's of the 3 bit field are don't care. The duplicated validities automatically come directly from the Playback Interface (PBI).

3.7 Tap Weights

The tap weights are stored in the validity ram as well. There is a separate Clear Accumulator (CLAC) Ram. The 64 tap weights, and CLAC Ram values, are loaded by the microprocessor into the fifos. The address consists of the tap weight index field from the fifo, and the 3 bit tap weight field from the sequencer. The 4 msbs of the tap weight address will normally be 1111 for the upper part of the ram where tap weights are stored. The PC will load all 64 tap weights. To read out the tap weights, a mux will jam the 4 lsb's to 1. 3 lsbs come from the program sequencer, and a 4 bit tap weight index field from a fifo. Each 131 ms., a set of 8 tap weights, for each of 8 tap weight indexes, is loaded into the ram. Note the 4^{th} bit allows an additional 8 tap weight indexes if needed. Referring to Figure 2, 1031d38 Tap Weights, note that these 8 tap weights correspond to what each input number is multiplied by, to be accumulated in the respective ram. The different

size filters can be automatically accommodated, based on what is loaded from the microprocessor each 131 ms. The set of tap weights desired is based on tap weight index. The tap weight index field chooses the correct set from the ram. The 4 bit tap weight index field comes from a delay fifo, to allow for the time in the LTA. The undelayed 4 bit, tap weight index field comes from the same fifo that contains the result counter modulus field.

Note for straight thru mode, the tap weight can be 1 for one ram and 0 for all the others.

3.8 Clear Accumulator (CLAC)

In the main program loop, in a single cycle, a DRAM input is added to a stored normalized value times a tap weight. For a tap weight of 0, a DRAM input of 0 is desired to clear the accumulator. The accumulator is cleared every n accumulations for an n tap filter. The filter size changes as a f(tap weight index). The CLAC function is implemented by means of the TSEL field of the 327 Floating Point Chip (FPC) program word. A value of 0010 causes the DRAM input to be forced to 0. The TSEL field normally comes from the program sequencer. A CLAC signal, from the ram, throws a mux to give the 0010 input.

The CLAC signal is derived as follows. A CLAC ram has as an address, the 3 bit tap weight address from the program sequencer, plus the 4 bit tap weight index field. CLAC\ from the ram will be a 0 when a tap weight 0 (which requires a CLAC) is requested. CLACENBL from the program sequencer enables the CLAC\ signal, to only allow an output when tap weights are being used. CLAC then throws the above mux.

The CLAC ram also supplies the signals SNAPCE\ and SNAPOE\. These control the snapshot register which chooses the appropriate ram bank signal to transfer to the backend buffer. SNAPCE\ takes the snapshot. SNAPOE\ puts the snapshot on the bus and WE\'s the DRAM. During a tic when nothing is written to the output buffer (as in 3 of the 4 tics when there is a data reduction), no SNAPOE\ would be present. The absence of SNAPOE\ is controlled by reloading the clac ram between tics. For a tap weight index in a straight thru mode, each point can get a CLAC\ to have nothing added to it, and a SNAPCE\ and SNAPOE\ to transfer it to the output buffer. This could be on any one of the writes to the 8 ram banks.

3.9 Microprocessor Interface

3.9.1 Microprocessor Loading of Fifos

One task of the microprocessor is to load the fifo's which provide the 19 bit address for accessing the LTA. The baselines are stored directly in a fifo. The faster changing, 11 bit result field is supplied by a counter. The RESCNT signal, on Schematic 1031d01 in Volume II, clocks the result counter. RESCNT is given 4 times, every 10 usec, at 2 usec spacings. This is due to the basic software loop of skipping every 5^{th} LTA request cycle. The RESCNT pulse is one 8 MHz clock wide. The 3 bit modulus is output by a fifo. Moduli of 2048, 1024, 512, 256, 128, 64, 32, and (2048 with LTABLRD's every 8), correspond to modulus field values of 7,6,5,4,3,2,1, and 0, as implemented in a 22V10. The 0 value is used during loading validities. That is since the full 11 bit count goes to the LTA, while only the 3 lsb's are used by the validity ram. The LTA BL fifo is read every 8 times to get the validity ram's msb's. See 1031d15, Figure 5, The Seven Bit Modulus Counter Pal (Location 6E) for the block diagram and timing. This same pal generates the clocks to read the baseline and modulus fifos. The positive read pulses are 1 clock cycle wide. This allows the output of the fifo to not be high impedance, when clocked by a 2 usec period clock. The fifo's are initially read when the counter is cleared. Subsequent reads occur when the counter is the highest number for the current modulus. The input RCO*RESCNT is gated in the 6E pal to give a 125 ns read pulse. Thus, the frequency of the read pulse depends on the modulus. The DRAM baseline read pulse is delayed an integral number of 2 usec cycles. RS[0..3] come into the pal so it can act on the appropriate delay. That is, generate the baseline read pulse when the msb's are zero, and the lsb's are the appropriate value.

The same fifo that provides the counter modulus also provides the tap weight index for the tap weight address logic. Also the STOPLOOP signal, from the fifo, tells the sequencer when to stop a loop, due to reaching the designated output from the fifo.

A conservative estimate of the time required to load the fifo's is as follows. A MOVX takes 1.5 usec. Assume it takes 6 usec, per number to the microprocessor. To load the fifos with 209 validity baselines, 210 LTA, and 210 DRAM baselines (for 629 numbers) should take 3.7 ms. There is 131 ms available, so no problem.

3.9.2 Microprocessor DRAM Address Generation

The address for the drams is generated as follows. For storing the LTA data in the FIR memory, the baselines are output by a separate fifo. The separate fifo allows a rearranging of the storage locations of the baselines. This is necessary since the LTA memory is twice as big as the filter ram banks. This allows separate portions, of a large subarray from the LTA, to be subsequently stored in the same filter locations. The resulting address field is delayed by a fifo and used for the drams as well. A control bit, BLMSB, allows selection of throwing away either the msb of the baseline or result field. This is necessary to reduce the 19 bit address field to 18 bits. That is, the FIR memory can be partitioned with 128 baselines, or 256 baselines of 1024 results. A mixture of the two is also possible, but care must be taken to not overlap a 1024 result baseline with the end of a 2048 result baseline.

The microprocessor can be used to test the data path via the 32 bit read and write registers. A write into the input fifo, can input into the divide chip, the validity rams, or the 327 FPC. Values can be loaded into the fifo, 64 at a time. This can give test bursts of high speed data, under microprocessor control. The microprocessor read register is part of the data crossbar pals. It reads the S bus, out of the 327 FPC. The cross bar can be positioned so the register reads the F bus, the A ram bank bus, or the B ram bank bus. If the 327 is put into straight thru mode, the R bus can be read as well. There is a straight thru window in the main program. A 20 bit address would have been queued up. The software cycles 2 usec for inputting 4 points, then 8 usec for mac's. The 2 usec could be used for microprocessor DRAM access as well as the backend reading the output buffer. Output banks R8 & R9 can be read directly via the backend.

3.10 Dram Address Generation

The 20 bit DRAM address is fed as a 10 bit ROW and COL by the address mux, to the 10 bit DRAM address input. The sources of address are related to the addresses given to the LTA, the microprocessor, the backend access logic, and the refresh counter. BLMSB and beBLMSB alter the ROW addresses, to tell whether to throw away the MSB of the result or the baseline. In partitioning the DRAM among subarrays, think of the DRAM as being a matrix 256 baselines by 1024 results. First assign the baselines with 2048 results. Since the msb of the result replaces the msb of the baseline, pairs of 1024 result baselines are used. The pairing is baseline x with baseline x + 128. The lower 1024 results go in baseline x, while the upper 1024 results go in baseline x + 128. In partitioning the DRAM among the current subarrays, first assign all the 2048 RESULT BASELINES. These use the pairs of baselines. Then fill in the remaining spots in the upper and lower halves of memory with the 1024 result baselines.

Refer to 1031d14 in Volume II for the data and addressing order to the DRAMS. The address for the backend buffer, when writing to it, is derived from the delayed LTA address. The LTA address is the same for the entire accumulation, to the 8 ram banks. The address can be directly used, after both the real, and imaginary values have been input. That is since the address is the same for all 8 values, there is no need to differentiate it. The data changes, since it has been multiplied by different tap weights. A snapshot of the chosen data for a subarray is stored in a register. This snapshot is stored at the above address in the backend buffer.

3.11 DRAM Printed Circuit Boards

See 1031d03, the FIR DRAM PC Board Schematics. The DRAM Modules are mounted on PC boards. There are two identical boards, that each hold eight DRAM Modules. A third PC board is half the size, and holds four DRAM Modules. The PC boards plug into the FIR Board.

3.12 Backend Address Generation

(See 1031d06.sch for the schematic, and 1031d07.sch for the block diagram)

We need to generate the address, for reading out backend buffers 8 and 9. A baseline and its Backend Result Ram Index (BERRI) are sent to the fir from the VME. BERRI is a number 0 to 15, specifying the table in the result ram to use. Upon receiving the correct number of results for the baseline requested, the VME can send the next baseline to be read.

Since the backend DRAM is read into the fifo, 7 values per 10 usec cycle, it will take a non-integral number of cycles to read a baseline's results. The backend fifo is 4K, so it will hold the largest baseline. At the start of a baseline, RCNTCLR\ clears the result counter and resets the backend fifo. EMPTY\, from the backend fifo, has the VME only read the fifo when there is something in it. Normally, the VME will read each 7 value burst as it is put into the fifo. The counter in the 29MA16 counts down the number of results, based on VME acknowledges to the fifo. When the count is done, the signal DONE inhibits fifo reads, and shuts off the bus to the VME. Since there are no more reads, DONE remains until a new baseline is requested.

The result ram supplies beBLMSB which specifies, use the MSB from the baseline, and throw away the msb from the result field. The beBLMSB switching is done at the ADDRESS MUX. The result ram determines the result ordering based on a count, on a BERRI basis. The backend result counter gets its clear signal from the logic that loads a new baseline from the VME. Thus, the new baseline counts from result count 0. The counter will count until a new baseline is requested by the VME. DONE prevents the VME from handshaking beyond the requested count. The result counter counting beyond that count does not hurt anything. STOPLOOP stops the main program loop, after all baselines have been requested from the LTA. The VME could still be requesting baselines beyond the time of STOPLOOP. That is since the backend is asynchronous with the frontend, but has to finish last.

In reading the LTA, the 3 lsb's of the incrementing result field are the channel. This is used to determine validities. Having the ram for the backend, allows flexibility in the order the results are output. The result ram allows 16 subarrays of 2048 results, specified by the BERRI. Spare subarrays could be used for bank switching. A spare subarray could be used to regroup the results in the validity baseline. This would allow the validities, for the baselines of interest, to be grouped in the lower contiguous count.

The beBANK\ bit determines which bank (8 or 9) is currently being output. This is switched by the microprocessor every 131 ms. beIMAG toggles to get the real and imaginary half of complex words. beIMAG is the lsb of the backend result counter. beIMAG, beBANK\, the baseline, and the result, all combine to produce the 20 bit, DRAM address.

In filtering, when an accumulation of values times tap weights is complete, it is ready for outputting. Which ram bank is ready for outputting rotates. The Real Time System keeps track of the baseline it wants to request, at each point in the cycle. The selection is accomplished by means of a snapshot register on the F bus. This takes a snapshot of the final sum of products on its way to the ram banks. The ram banks written to are banks 0 through 7, one each clock cycle. Using SNAPCE\ from the CLAC ram, to the snapshot register, the appropriate bank can be captured.

The LTA Baseline and MOD fifos output once a baseline. This is accomplished by generating DRMBLRD and LTABLRD in a 22V10, based on the baseline modulus and result count. See 1031d15, Figure 5, for a pal description. Refer to 1031d14 in Volume II and the block diagram. The row and column address to ram 89 is a delayed version of that to the 57 ram bank. This allows time for the data to pipeline into the snapshot register. The worst case is taking a snapshot of ram bank 7. The bank switch bit comes via the beADR9 mux pal. This address, along with SNAPOE\, is used to transfer the snapshot to the appropriate place in the 89 ram.

There is a 32 bit output data FIFO to the backend. The data is input into the fifo in bursts of 7, during every 5th, 2 usec cycle. Since this cycle is used to input data into the 327, the dram address bus is not being used. DRAM 89 can use the bus to output its data to the backend.

3.13 Backend Interface via the Ironics Board

3.13.1 Hardware Interface Between the FIR and the Backend

The IRONICS IV-3272, with the IV-3272-PIO Parallel Interface Daughter Board, is the interface from the FIR into the VME.

The PIO board provides a TTL interface at the VME P2 connector. (See APPENDIX IV Ironics IV-3272-PIO Parallel Interface Daughter Board). This transition module, in the rear of the VME chassis, provides a buffer module to convert the interface to RS-485 differential. The RS-485 signals are carried on two 25 pair cables, with 3M50 connectors, pin out as follows:

3M CONNECTOR # 1

PIN	SIGNAL	PIN	SIGNAL
1	beD15	2	beD\15
3	beD14	4	beD\14
5	beD13	6	beD\13
7	beD12	8	beD\12
9	beD11	10	beD\11
11	beD10	12	beD\10
13	beD9	14	beD\9
15	beD8	16	beD\8
17	beD7	18	beD\7
19	beD6	20	beD\6
21	beD5	22	beD\5
23	beD4	24	beD\4
25	beD3	26	beD\3
27	beD2	28	beD\2
29	beD1	30	beD\1
31	beD0	32	beD\0
33		34	
35	REQ*	36	REQ*\
37		38	
39	ACK*	40	ACK*\
41		42	
43	ENDIR	44	ENDIR\
45		46	
47		48	
49		50	

3M CONNECTOR # 2

PIN	SIGNAL	PIN	SIGNAL
1	beD31	2	beD\31
3	beD30	4	beD\30
5	beD29	6	beD\29
7	beD28	8	beD\28
9	beD27	10	beD\27
11	beD26	12	beD\26
13	beD25	14	beD\25
15	beD24	16	beD\24
17	beD23	18	beD\23
19	beD22	20	beD\22
21	beD21	22	beD\21
23	beD20	24	beD\20
25	beD19	26	beD\19
27	beD18	28	beD\18
29	beD17	30	beD\17
31	beD16	32	beD\16
33		34	
35	CTRL4	36	CTRL\4
37	CTRL3	38	CTRL\3
39	CTRL2	40	CTRL\2
41	CTRL1	42	CTRL\1
43	STAT4	44	STAT\4
45	STAT3	46	STAT\3
47	STAT2	48	STAT\2
49	STAT1	50	STAT\1

Note:
There are 8 each control and status bits available on the VME P2 connector. The bits 1-4 selected for use here correspond to VBUS bits 9-12 and have been selected so that the control bits in the P2 cable that are next to REQ, ACK and DIR are not active.

REQ* and ACK* are bi-directional, open-collector signals. The device supplying the data (the source) drives REQ*, and the device receiving the data (the sink) drives ACK*. ENDIR signals the direction of the data transfer. ENDIR in the low state indicates a transfer from the FIR to the VME. ENDIR goes high if the transfer is from the VME to the FIR.

3.13.2 Backend Data Transfer Example

The following sequence of events has the VME system request a DMA transfer from the FIR: (See 1031d16 for pal timing)

a) VME sends 2, 12 bit words using the 12 lsb's of the data line. The words consist of: a baseline number, a subarray number aka BERRI, and a result count.

(ENDIR high causes the FIR to clock the word(s) in on the falling edge of REQ* and to acknowledge the data with a pulse on ACK*.)

b) VME sets up to receive the response buffer. The fir writes data to the VME with a falling edge on REQ* and waits for a falling edge on ACK* before writing the next word;
(See the IRONICS IV-3272-PIO manual for details of the timing (page 2-11).)

There is the possibility that the fifo in the Ironics board might fill up due to the VME not reading it out. The FIR should then stop sending data. Since the backend fifo is 4K, it can hold a whole baseline without having to write to the VME.

The lsb of the result counter is BEIMAG, which specifies the real or imaginary word of a complex result. Note that this lsb is not sent to the result ram, since the address out applies to both the real and imaginary values. In loading the result ram, an initial value is loaded from the microprocessor. The PC sequencer can increment the ram address by having RCNTENBL high for 2 counts.

4 Inputting and Filtering Validities

4.1 Validities Introduction

The path of the validities through the system is as follows:

```
To the left of this line  |  To the right of this line
frame based validities    |  FFT based validities
Track clocks-->TRC-->DEF---->MCC-->FFT
Data clocks                      \-->LTA-->FIR-->RTS
```

The TRC gets track data and track clocks. Based on CRC and such, it decides if each track is valid on a frame by frame basis.

This goes to the DEF which apportions the validities on an fft cycle basis. Also the delay, due to the various pipelines which the data follows in the DEF, is inserted.

This goes to the MCC. There are input muxes analogous to the input muxes on the FFT cards. There is then a delay inserted. This then goes to the FFT cards. Each MCC handles two channels worth of validities. All 24 pbi's are input and 20 fft cards. The StarDiagram, which shows the 24 pbi to 20 fft interconnect, is hard wired in the mcc for the validities.

In the MCC, after the input muxes, the validities are accumulated on a per baseline basis. That is if either PBD of a baseline is bad, the accumulated validity is bad. These accumulated validities go to the LTA. The LTA does long term integration of the validities in Baseline 210, on a per channel basis. There are pseudo baselines for the 210 real baselines. The FIR reads the validities out of the LTA, and writes them in FIR baseline 127. The validities can be filtered,

with the unfiltered values placed in the real part of each complex word, and the filtered in the imaginary part. The validities can also be inverted, and used to normalize the data. The real time system reads the validities out of the FIR.

4.2 Filtering Validities

Filtering validities is viewed as a viable alternative to using the validities to normalize the data in the FIR. Instead, the filtered validities can be used to normalize the data in the post processing.

To filter validities do the following. Run a special filter program while loading the normalized validities into the validity ram. The special program is a separate page in the Lotus spreadsheet which represents the sequencer program. The tap weights and CLAC ram for the tic would have to already be loaded. LOCK is a control signal to jam the baseline request to the LTA to 210. The LTA BL FIFO tells the baseline to the validity ram, and gives the sub-baseline portion of the result field to the LTA, telling which sub-baseline in 210. This value also goes to the DRAMs in the result field. The MOD FIFO specifies the subarray, so the correct tap weights and clac could be used in filtering. The modulus out of the MOD fifo is 0, which gives a count up to 2048 with a baseline read every 8 times. The validities are stored 1 per complex value, since the result count to the LTA also goes to the DRAMS. This requires a 2048 result baseline (as opposed to a 1024 result baseline) to store the 1680 validities. The FIR baseline to store the validities, such as 127 is specified by the DRAM baseline fifo. Note this is a 2048 result baseline, so 127 is the highest baseline number available.

A validity comes in from the LTA approximately every 2 usec. The validity is captured in the 327 math chip and the divide chip. The filtering is started, while the divide chip is doing the inversion. The R bus is needed for tap weights. One microsecond is needed to mac into the 8 ram banks. Validities are only updated for the baselines at the end of their 32 accumulations (for a 32 tap filter). The remaining baselines will have older values. The filtered results will be in the real part of the words. The unfiltered validities will be put in the imaginary part of the word in the output buffer and all the rams. This could be useful for debugging. After accumulating the validities in the ram banks for the filter, the R bus is used to write the normalized validity stored in the divide chip's buffer into the validity ram. The R bus is available in the 2^{nd} half of the 2 usec cycle, since that is when the imaginary data is processed. Since the validities go straight thru for the imaginary part, no tap weights are needed, freeing the R bus for the transfer from the divide chip to the validity ram.

Refer to the block diagram 1031d25 in Volume II. Logic switches the LTA baseline fifo output to the result field, so that only the baselines of interest are processed. However, the timing still needs to allow for the worst case, of all 210 validity baselines being processed. Currently, the validity addressing to the LTA is: The baseline is 210. The channel is the 3 lsb's of the result field. The pseudo-baseline of the validity is the remaining 8 bits of the result field.

In the case where the dump time is greater than once per 131 ms, it is necessary to not filter the validities every 131 ms. To do so would multiply the filtered validity by a scale factor. The above switch, which routes the fifo baseline into the result field for validities solves the problem. Only the baselines requested are processed, so baselines are not processed more than once per dump time. Also, this saves time, and makes the validity processing more like the normal filtering.

For a 16, 32, or 64 tap filter, there is a data decimation. Have something output to the output buffer every tic. Thus the backend will have to know which tic to look for data, since there will be garbage other tics. This is required since the actual validities are in the imaginary part each tic, even though the filtered validities don't produce results every tic. Since this requirement is imposed for validity filtering, it will be used for normal filtering as well. This requires a SNAPCE and SNAPOE every tic. The fifo's must be loaded with baselines and their associated subarrays before each validity filtering, just as in a normal data filtering. For the case of a straight thru mode without

normalization, it is desired to not multiply by the normalized validity. When loading the validities, there is a separate microprocessor/sequencer routine for baselines in this mode. The routine used is specified by the function code of hcb function 21 STARTMAINLP. The no normalizing, validity loading routine is on page 3 of the Lotus worksheet. The raw validity still goes to the DRAM as before. Unity is stored in the validity ram, instead of the normalized validity. The unity comes from the divide chip, by loading the same number as the numerator and denominator.

5 FIR Sequencer

5.1 Sequencer Overview

Refer to 1031d09 for a block diagram of the sequencer. The logic is on 1031d02. Both drawings are in Volume II. The sequencers are in levels. On the LTA, there is an FFT cycle sequence repeating every 516 clock cycles. An LTA integration cycle sequencer counts FFT cycles. The FIR sequencer counts the CK8, 8 MHz clocks.

5.2 LTA Interface to the Sequencer

The LTA FFT sequencer provides the signals CK16, CK8, and NEWPAGE, to the FIR. The 4 cycle gap is omitted from these to allow staying synchronized with the LTA. There is an FIR program step sequencer, initiated by the NEWPAGE signal from the LTA.

The microprocessor gets INTRST\, an asynchronous clear from the LTA, every 131 ms. This goes to the microprocessor INT0. Currently, INT0 is not implemented in software. Instead, FIRENA is read from the LTA on a poling basis to delineate the tics. The microprocessor provides beBANK the backend bank (8 or 9) toggling every 131 ms.

5.3 FIR Program Counter

The FIR program counter (PC) generates a 8 bit count, allowing a 256 step program. A 3 bit page allows selection of 8 possible programs. These combine to give the 11 bit address to the program roms. The page comes from the microprocessor. The page is written by WR\8302 to REGA. See 1031d09. This sets WRS. The leading edge of NEWPAGE clocks WRN to generate PCCLR\. PCCLR\ clears the PC counter and clock enables FIRPAL7, to update the page. The REGA to REGB buffering allows for the microprocessor being asynchronous.

Within a program, it is possible to have branching either as specified by the microprocessor, or the program. For program control there is an address supplied by the PC rom. This supplies an address for branching from the PC sequencer. This address can be branched to by loading it into the program counter via PCLD\, from the PC rom.

An alternate branching address can be supplied by the microprocessor. WR\8300 writes an 8 bit address to a register, which is wire ored with the address from the PC rom. WR\8301 acts as PCINT\ (program counter interrupt\) to initiate a branching to a microprocessor specified address. The PC could be in an infinite loop specified by the PC rom address. A service routine could be in the microprocessor branch register. Upon the microprocessor issuing a PCINT\, the next time the PC rom issues TOGGLE, the microprocessor Branch Reg. will be output enabled. The next PCLD\ will branch to that address. The service routine will contain a DONE\ signal, from the PC ROM, to reset the PCINT register. At the end of the service routine, a PCLD\ will get an address from the PC rom instruction register.

5.4 Main Loop Structure

This presents the events of a typical 131 ms cycle. First, the microprocessor loads the fifos and validities. Then, the main loop runs until STOPLOOP from the MOD FIFO, at the last baseline in the fifo. See 1031d09, 1031d27 in Volume II, and Figure 13, The Sequencer Control PAL Description. The DRAM BL fifo will empty after the wrap around, DRAM writes, at the beginning of the main program loop. That is the DRAM writes complete in the beginning of the next loop cycle. PCLD\ (Program counter load\) and SEQuPBR\ (Sequencer microprocessor branch) being present at the same time, indicate check the STOPLOOP and INPUTAFE as a condition for letting them through. If there is a STOPLOOP, then uPBR\ sets the register to switch the branch register to the microprocessor branch register, which will have been previously loaded to the branch routine. Also, a LOAD\ pulse is generated. The branch routine then puts things in idle until the next 131 ms cycle.

5.5 Tasks for the Sequencer to Do Broken up into HCB Tasks

I. Load the result backend ram. This is good till the setup changes. Load the 327 and divide chip mode words.

II. Do observation broken into 131 ms cycles Every 131 ms the microprocessor toggles beBANK.

Then every 131 ms:

IIa. Input from the LTA, normalize, filter and store the validities in ram.

IIb. Input the up to 128 tap weights and CLAC ram values (CLAC, SNAPOE\ & SNAPOE\) and store in rams. In the microprocessor the whole table could be stored.

IIc. Microprocessor Load the LTA BASELINE, COUNTER MODULUS, and DRAM fifos with values from the RTS via the HCB. These could have been input on previous tics.

IId. Do the main filter loop with backend accesses.

5.6 How the Sequencer Tasks are Implemented

```
                THIS RAM IS GOOD UNTIL THE SETUP CHANGES

   AN ARBITRARY SECTION OF MEMORY CAN BE LOADED AT ANY ONE TIME.
  TO LOAD AND CHECK THE RESULT RAM
* - THE uP REGISTERS GET A BRANCH ADDRESS & PAGE.
 uADR[0..15] PROVIDES DATA TO WRITE IN THE RAM.
 SET TEST=1 (P1.7)(SEE L031D21) SO THAT uADR[16..19] ALWAYS PROVIDES ADDRESS BITS A[11..14]
* - ON WRITING A NEW PAGE, NEWPAGE HAS PCCLR\ RESET THE PC COUNTER, STARTING THE PROGRAM IN IDLE LOOP 0.
beRESLD\' (WR\8107 THRU 2 REGS) LOADS uADR[0..11] AS THE INITIAL RESULT COUNT THEN THE PC SITS IN IDLE LOOP 0.
* - BRANCH TO IDLE LOOP AT 8 WHERE PREASEL[0..2]= 3 OR 5 TO GIVE uADROE=1 TO ENABLE uADR[0..15]
* - THE uP GIVES beRAMWE\(WR\8104), THEN THE NEXT DATA VIA uADR[0..15]
* - PCINT\ FROM THE uP, BRANCHES TO A ROUTINE 12 TO INCREMENT
 THE COUNTER VIA RCNTENBL, THEN THE PC IDLES AT 8.  P1.1=0.
* - uP INCREMENTS ITS COUNT AND ADDRESS BITA A[11..14] IF NECESSARY.  IF < DESIGNATED NUMBER GO TO

* - DONE WRITING TO CHECK RESULTS:
beRESLD\' (WR\8107 THRU 2 REGS) LOADS uADR[0..11] AS THE INITIAL RESULT COUNT THEN THE PC SITS IN IDLE LOOP 0.
 uP REGISTER GETS A NEW BRANCH ADDRESS
 uADR[0..15] PROVIDES RAM ADDRESS A[11..14] SINCE TEST = 1
* - PCCLR\ RESETS THE PC COUNTER, STARTING THE PROGRAM IN IDLE LOOP 0.
 PREASEL[0..2]= 0,1,2,6 OR 7 TO GIVE RSRAMOE

* - uP REGISTER GETS A NEW BRANCH ADDRESS FOR ROUTINE 15
* - RD\8004 & 5 READ uADR[0..15] TO CHECK RESULTS.
* - PCINT\ FROM THE uP, BRANCHES TO A ROUTINE 15 TO INCREMENT
 THE COUNTER VIA RCNTENBL, THEN THE PC IDLES BACK AT 0.  P1.1=0.
* - RD\8004 & 5 READ uADR[0..15] TO CHECK RESULTS.
* - uP INCREMENTS ITS COUNT.  IF < DESIGNATED COUNT GO TO
```

Figure 4 Backend Result Ram Loading

I. This Figure shows details of loading the result ram. To load the mode words, the microprocessor could load them in the input fifo, then a PC routine could read them out of the fifo, and strobe them into the ram.

II. FIRENA is seen to change, indicating the 131ms tic. The microprocessor requests the initialization page.

IIa. We input from the LTA, normalize and store validities in RAM.

See l031d10 in Volume II for the divide chip timing. The microprocessor branch register specifies a routine to do the validity processing. On PCINT\, the routine processes the validities, looping via the PC rom instruction register. The validities could also be stored in a spare baseline in the 89 DRAM for later retrieval.

The microprocessor branch register specifies the address of the routine to escape from the loop. This is branched to when the STOPLOOP is received from the MOD fifo.

The escape routine resets the fifo's, sends uPINT\0=0 to the microprocessor to give the microprocessor back control, then idles. During the idle, the DRAM should still be doing refreshes.

IIb. We load the tap weights and CLAC data into the rams from the microprocessor. The program counter (PC) starts at 0. The microprocessor page select register is written to via WR\8302, then strobed in by WR\8303. The PC sequencer resets the input fifo and CLAC fifo via FIFORST\, then goes in an idle loop, with refreshes, by looping on its PC rom instruction register.

The microprocessor outputs up to 64 tap weights into the input fifo, via WR\8000,1,2,3. Up to 64 corresponding CLAC's are written into a fifo via WR\8005.

The microprocessor performs this loop once for each subarray. Reset the MODULUS/subarray fifo via WR\8006. A subarray is written, by the microprocessor via WR\8203, into the modulus/subarray/outbank fifo. The microprocessor branch register specifies a routine to transfer the tap weights into the validity ram.

On PCINT\, the PC seq routine is started. RESCNT simultaneous with RESCLR\ will give an LTABLRD to read the subarray from the fifo (see L031D15, Figure 5).

The subarray delay fifo is reset by ADRFRST\, clocked by LDLYLDCK, then unclocked by SUBAUNCK. FIFOSO reads the tap weight from the input fifo, and also reads the CLAC fifo. QWE writes the tap weight into ram. CLACWE\(ps5) writes the CLAC into its ram. Repeat the read and write instructions, only with an incremented tap weight, for the 8 tap weights of the subarray. Send uPINT\0 = 0 to the microprocessor in an idle loop. Send the done message. The microprocessor will return for the required number of subarrays.

IIc. The microprocessor resets and loads the LTA BASELINE, COUNTER MODULUS, AND DRAM BASELINE fifos. Note, the MOD fifo provides STOPLOOP to allow stopping the PC.

IId. The microprocessor branch register is loaded with the escape routine address for the main filter run. The microprocessor starts a new page for the main filter run. It is necessary to initially have the input fifo precharged with the first 4 complex points. l031d27 in Volume II shows how, by having the fifo precharged with 1 dummy value, 8 more values (4 complex numbers) cause INPUTAFE to toggle, allowing the loop to proceed.

The filter loops using the rom PC address. When the fifo is empty the escape routine is branched to. The escape routine sends uPINT\0 to the microprocessor then idles. During the idle, backend accesses should continue, in case the previous cycle had a lot to transfer. See /vlbsoft/pal_prom/proms/firfpc/pg0.wk3, a Lotus worksheet, for a listing of the 327 instructions for the main filter loop. The microprocessor idles until the next INTRST\ to the microprocessor. It could need to shut down the backend accesses before transferring back to the initialization page.

5.7 Detailed Sequencer Main Filter Loop Structure

5.7.1 PROM Addressing Structure

Refer to l031d14 in Volume II. This shows the actual steps of the main filter loop. There is one row per 8 MHz clock cycle.

The column labeled PROM ADDR is the address to the sequencer PROM. Recall the sequencer is divided into pages, with each page having up to 256 instructions. PROM ADDR is presented as a hex number in the range 00 to FF.

The main loop timing is divided into 2 microsecond sections. Each section contains 16, 8 MHz clock cycles, corresponding to 16 sequencer instructions. On l031d14 in Volume II, the horizontal lines divide the table into the 2 usec sections. Notice the most significant digit of the address increments for each 2 usec section.

5.7.2 The Idle Loop

The bottom two sections, C0 thru DF, represent the idle loop. The sequencer loops through these two sections, when there are no other filtering tasks. Filtering Task refers to a list of baselines to filter, read from the fifo. The reason we need this activity when there are no current filter tasks is to allow backed access to continue. The backend buffer can be read out, whether or not a filtering task is going on.

5.7.3 The Main Loop

Refer back to 3.2, the Basic FIR Program Loop. This also shows the main loop, grouped into the 16 clock sections. The filter main loop consists of five, 2 usec sections. These correspond to PROM ADDR 60 through AF. The sequencer loops through these five sections, until the STOPLOOP signal indicates there are no more baselines from the fifo, to process. STOPLOOP is a bit from the fifo, indicating the end of the list. If STOPLOOP is not yet encountered, the sequencer branches back to the beginning of the loop. If there is a STOPLOOP, then the sequencer continues without branching, and ends up in the idle loop.

l031d12 in Volume II shows the detailed timing of the initialization of the Main Loop.

5.7.4 DRAM Addressing

5.7.4.1 DRAM Addressing Basic Operations

l031d14 in Volume II shows the timing and functionality of the DRAM addressing. The five DRAM banks are shown as the columns 02, 13, 46, 57, and 89. 89 is the backend buffer.

The basic operation at a DRAM address is a multiply/accumulate. That is read the address, add something to the value, then write the sum back into the address.

Each DRAM bank is divided in half, and used as two banks. For example 02 is bank 0 and bank 2, all in one physical DRAM.

The 20 bit DRAM address is delivered over a 10 bit address bus in two pieces, called a row and column.

The sequence of operations at a fixed DRAM address is as follows (DRAM 02 is used in the example):

RR0	Read Row Bank 0
RC0	Read Column Bank 0
RR2	Read Row Bank 2
RC2	Read Column Bank 2
WR0	Write Row Bank 0
WC0	Write Column Bank 0
WR2	Write Row Bank 2
WC2	Write Column Bank2

Thus it takes 8 clock cycles to process an address in 2 ram banks. However, this is going on simultaneously in all 4 pairs of banks, 02, 13, 46, and 57. Thus in 8 clock cycles we are able to process all 8 ram banks. Back in the main loop description in Section 3.2, it was shown that 8 clock cycles are allowed for the 8 multiply/accumulates to the 8 ram banks.

5.7.4.2 Data Path to the DRAMs

The two DATA columns, on the left side of 1031d14, show the data to and from the floating point chip (FPC). The IN column is the FPC input, the OUT column is the FPC output. There is a four clock, pipeline delay in the FPC, between the input and output. The FPC input is read from the DRAM. The FPC output is written back into the DRAM.

Each value is actually a complex number, consisting of a real and imaginary part. These are each multiply/accumulated. Thus it takes 16 clock cycles to process a complex value through the four main DRAM banks. This again is our basic 2usec period.

5.7.4.3 Generating the Addresses for the Main DRAM Banks

The block diagram, 1031d25 in Volume II, shows that the DRAM addresses for the different ram banks are generated by a series of delays from the address out of the 8 to 1 MUX. 1031d14 shows a column with the output of the 8 to 1 MUX. Since the Write address is always the Read address, delayed four clock cycles, there is the Four Cycle Delay Fifo. This splits the address bus into two, with one delayed four cycles. There are multiplexers into the DRAMs, that switch every four cycles, to pick off the Read and Write addresses. The sequencer signal deltoa, can be seen on 1031d14. This signal toggles the multiplexers every four clock cycles.

Ram bank 46 is generated by using a multiplexer 180 degrees out of phase with the multiplexer for bank 02. This has the effect of a four cycle delay.

A single register delay generates the address bus for DRAM Bank 13, from that of Bank 02. Similarly, the address bus of DRAM Bank 57 is one cycle delayed from that of Bank 46.

5.7.4.4 Generating the Address for Writing into the Backend DRAM Buffer

For the DRAM address for the backend buffer, the object is to be able to write into the backend buffer results that have been completely filtered. The snapshot register will have taken a snapshot of the value from the appropriate bank. The value is then written from the snapshot register to the backend DRAM buffer. The worst case is getting the value from the last DRAM bank written. Thus we take the backend address from bank 57, then delay it an additional clock cycle, due to the delay of the snapshot register.

The timing of which value is captured in the snapshot register is specified by SNAPCE\ and SNAPOE\ which are from the CLAC ram. Of course, the CLAC (Clear Accumulator) signal is also from the CLAC RAM. The timing for all these is shown on 1031d14. The CLAC ram is loaded by the microprocessor, along with the Tap Weight Ram.

5.7.4.5 Generating the Address for Reading the Backend DRAM Buffer

The same DRAM address bus is used in the reading out of the backend. See 1031d07, the Backend Block Diagram. Referring back to Section 3.2, on the basic FIR loop, we see that in the beginning of the loop the data is being read in from the LTA and multiplied by normalized validities. The DRAM bus is not being used then, so it is available for backend access.

On 1031d14, be0 through be6 to DRAM bank 89, represent the 7 backend accesses available per loop. Each address is present for 2 clock cycles to represent a row and column. The data output from the backend goes into a fifo, to allow for the asynchronous nature of the sequencer writing the data into the fifo, versus the data being read by the Ironics Card.

In the Idle Loop, the backend access is the only demand on the DRAM address bus. Thus in the Idle Loop, the backend can be read at a higher data rate.

5.7.4.6 Refresh Cycle Address Generation

The backend is read seven times, instead of eight times, to allow time for the refresh cycle. This is marked RFR on 1031d14. The refresh cycle gets its address from the refresh counter. This systematically accesses all the Rows in all the DRAMs. The DRAMs are required to be regularly accessed, so they don't forget their contents.

5.7.5 Additional Signals on l031d14

The columns of l031d14 show many additional signals, not previously discussed . The number in parentheses, by the signal name, tells the bit number in the 103 bit sequencer word. Among the signals are:

5.7.5.1 vadroe(65) The Validity Ram Address Output Enable

vadroe acts as a switch for the source of the address to the Validity Ram. See the Block Diagram l031d25. vadroe=1 has the address come from the Baseline and Channel Fifos, for reading validities from the Ram. vadroe=0 has the address source come from the Tap Weight Address from the Sequencer and Tap Weight Index.

The validities are read at the beginning of the main loop. See Section 3.2. Notice vadroe also is 1 for B1 thru B4, since that is part of the main loop. The loop goes back to address 66.

5.7.5.2 rcntenbl(69) Backend Result Counter Enable

See the Backend Block Diagram, l031d07. This increments the Result Count used in reading the backend buffer.

5.7.5.3 preasel(8-10) Pre 8 to 1 Multiplexer Address Select (3 bit field)

This signal is buffered by a pal, then provides the 8 to 1 Mux address selection. This decides which function the DRAM address bus will be performing. l031d25, the Block Diagram, has a table of the choices.

5.7.5.4 we\ 4 and 0 DRAM Write Enable\ for Banks 46 and 02

These are the DRAM write enables. Banks 0 and 2 are in one physical DRAM, so there is only one write enable. we\1, we\5, and we\8 do not come directly from the sequencer. we\1 and we\5, are we\0 and we\4, delayed one 8 MHz clock cycle. we\8 is a buffered version of the SNAPOE\ signal, which enables the output of the snapshot register, to be written in the output buffer.

5.7.5.5 precas\ 0, 1, 4, 5, and 8 Pre DRAM Column Select

These signal are buffered by a pal (see Figure 14, The CAS Handling PAL, based on l031d17). l031d08 shows the detailed DRAM timing. The pal gates the precas signals with casws. casws is a version of the 8 MHz clock. The result is cas signals with the required 75% duty cycle, as shown on l031d08.

5.7.5.6 dlyrc(71)

See Figure 14, based on l031d17. dlyrc is used to derive beDINCLK, which is used to clock data into the fifo, when reading out the backend buffer. The pal gates with casws, which is an 8MHz clock, and with beLAST.

See l031d07, the Backend Block Diagram, for the enabling signal beLAST. This is based on the LASTRES signal from the Result Ram, indicating all the results have been read out.

The dlyrc signals occur in conjunction with the be addresses to the DRAM address bus.

5.7.5.7 10ustic(70) Ten Microsecond Tic

See Figure 12, The Ironics Handshake Pal Description, based on l031d16. This signal goes high once per 10 usec Main Loop Cycle. 10ustic must equal 0 to enable a handshake and transfer of data to the Ironics Card. 10uSTIC prevents starting a backend request in the middle of a 7 RCNTENBL string, since the address pipeline needs to be filled correctly ahead of time. See l031d07, the Backend Block Diagram.

5.7.5.8 clacenbl(3) Clear Accumulator Enable

See Figure 17, The CLAC PAL, based on l031d39. CLACENBL is gated with CLAC from the CLAC RAM, to do the clear accumulator function.

5.7.5.9 tw(0-2) Tap Weight Selection

This selects which of the eight tap weights from the Tap Weight Ram to use.

5.7.5.10 preaenb\, prebenb\(11-12)

These are buffered in the PAL at U41. Then these two bits control the selection of the DRAM Data Crossbar. See Figure 9, The DRAM Data Crossbar PAL, based on l031d22, for the implementation.

5.7.5.11 rescnt(57) Result Count

See the Block Diagram, l031d25, in Volume II. Rescnt provides the count pulse to the Counter which counts the result for the main filter loop. Rescnt increments the counter every 2 usec in the last 8 usec of the 10 usec Main Loop.

5.7.6 For More Detail Than Shown on l031d14

The fir card sequencer is programmed via the file /vlbsoft/pal_prom/proms/firfpc/pg0.wk3, a Lotus worksheet. On this worksheet, page 0 is miscellaneous functions, page 1 is the main loop, page 2 is the normalized validity loop, and page 3 is the non-normalized validity loop. The macro alt-p generates the print file fir.prn. The file fir.prn can be used as the ultimate reference as to the sequencer functionality.

6 FIR Microprocessor Software

The 87C51 microprocessor on the FIR card has a 32K X 8 main memory RAM. The main program for the FIR microprocessor is downloaded by the RTS into this memory.

6.1 Brief Descriptions of the Software Modules

Below is a brief description of each of the modules listed in the chart. (Memory allocations seen below are as of 3/31/98)

```
*****************************************************************************
*                              L O A D      M A P                           *
*****************************************************************************
*    Section Name              Starting Address    Ending Address    Size    *
*****************************************************************************
* master.obj                                                                 *
*    master_section                 0000                0001         0002    *
*    int0_section                   0003                0005         0003    *
*    timer0_section                 000B                0010         0006    *
*    int1_section                   0013                0015         0003    *
*    timer1_section                 001B                0020         0006    *
*    uart_section                   0023                00A5         0083    *
* monitor.obj                                                                *
*    monitor_section                00A6                0491         03EC    *
* romhcb.obj                                                                 *
*    romhcb_section                 0700                07A2         00A3    *
* test.obj                                                                   *
*    test_section                   0800                0F27         0728    *
* ram.obj                                                                    *
*    ram_section                    2000                2013         0014    *
* ramtest.obj                                                                *
*    ramtest_section                2100                26F7         05F8    *
* ramhcb.obj                                                                 *
*    ramhcb_section                 2800                2C57         0458    *
* help.obj                                                                   *
*    help_section                   2C58                335A         0703    *
*****************************************************************************
```

6.1.1 MASTER.ASM

MASTER.ASM has the software executed after a hardware reset to the microprocessor. This software will initialize the card and initialize microprocessor functions like the interrupt logic, the serial port, etc. This module also has the interrupt vectors. After a reset, the microprocessor will execute from an idle loop in MASTER.ASM, awaiting instructions from the terminal or RTS, conveyed by interrupts.

6.1.2 MONITOR.ASM

The monitor software package has various terminal options supported by the FIR microprocessor, such as display and modify memory, etc.

6.1.3 ROMHCB.ASM and RAMHCB.ASM

ROMHCB.ASM and RAMHCB.ASM handle the microprocessor communications with the RTS, over the HCB. Some of the basic functions, such as memory load, are handled in the ROM based software, but most protocol support is executed out of the RAM based RAMHCB.ASM program.

6.1.4 TEST.ASM and RAMTEST.ASM

The TEST.ASM module has a number of hardware test functions that allow an operator to test the FIR card either in the FIR test fixture or in the system. Hooks are in place so that software tests can be written in RAM in the RAMTEST.ASM module. This capability allows the generation of new tests without changing the FIR microprocessor ROM.

6.1.5 RAM.ASM

RAM.ASM is mainly used for linking the ROM and RAM based programs of the FIR together. It is not desirable to have to change the microprocessor ROM on the FIR cards, because of a change in a RAM based program. Thus direct linking of the ROM and RAM based modules is not possible. The strategy used in the FIR software is to

always link from the ROM software into the RAM software via jumps to locations that are on page boundaries in the RAM modules. These jumps will not change memory locations as software changes are made to the main bodies of the RAM modules. Hence the link address in the ROM modules will not change because of a change to a RAM subroutine.

6.1.6 HELP.ASM

This module contains ASCII terminal help screens. MONITOR.ASM has an option to display the help screen.

6.2 Microprocessor Interrupt Structure

Almost all of the functions provided by the FIR software are accomplished in interrupt routines. There are several interrupts, and a summary is given in the following table.

interrupt	function	priority
SERIAL	terminal	low
INT0	INTRST\	high
INT1	HCB communication	medium
TIMER0	not used	-
TIMER1	not used	-

The serial port interrupt is not normally used during an observation but supports a terminal and provides the functionality of the MONITOR.ASM software.

The microprocessor gets INTRST\, an asynchronous clear from the LTA, every 131 ms. This goes to the microprocessor INT0. Currently, INT0 is not implemented in software. Instead, FIRENA is read from the LTA on a poling basis to delineate the tics.

The INT1 hardware interrupt is used for RTS HCB communications with software support provided by the ROMHCB.ASM and RAMHCB.ASM software.

7 FIR PAL Descriptions

7.1 PAL File Locations

The PAL files are located in vlbsoft/pal_prom/pals/fir/SCCS. The file names are firpal*.abl. Unless otherwise specified, all PALs are 22V10s.

7.2 FIRPAL1 7 Bit Modulus Counter PAL

See 1031d15, Figure 6 on the next page, for the PAL logic. The PAL is 6E on the lower middle of schematic 1031d01.

This PAL counts out the number of results per baseline. The number of results is defined by the 3 bit modulus MOD which comes from the programming FIFO, as shown on 1031d15.sch, Figure 6.

A 74ALS163 is used as the 4 lsb's of the count. The pal provides the upper 7 bits, to allow up to 2048 results per baseline.

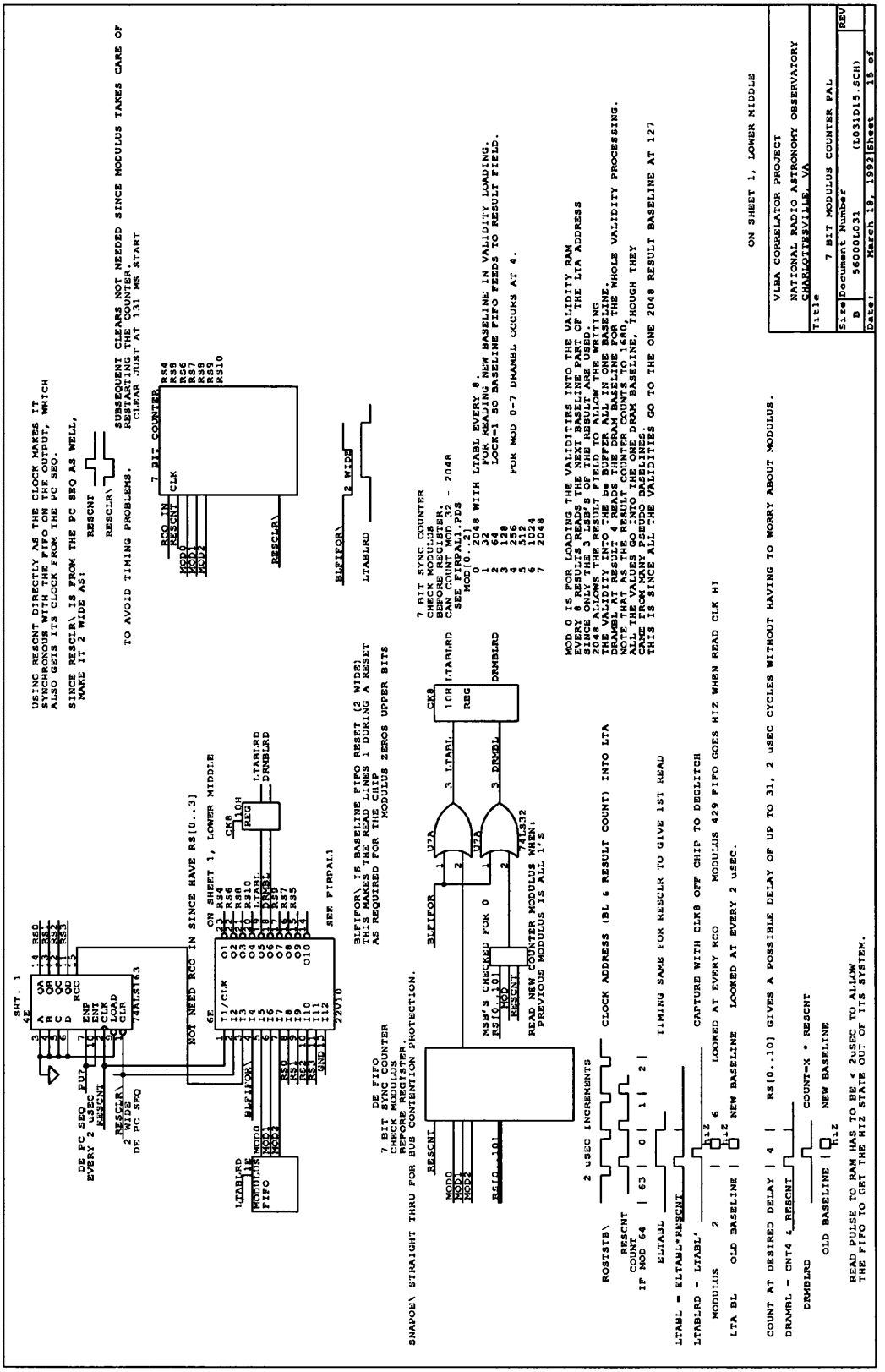


Figure 5 The Seven Bit Modulus Counter Pal

The signals LTABLRD and DRMBLRD are generated, to read out the next LTA or DRAM baseline from the instruction FIFOs. For all but MOD equals 0, the signals are simultaneous. MOD 0 is used while processing validities. For MOD 0, the DRAM baseline is left as a fixed number, while the LTA baseline is read from the fifo every 8 results, as pseudo baselines. That is the validity baseline consists of 256 pseudo baselines of 8 results each. All the validity baselines are stored in one DRAM baseline.

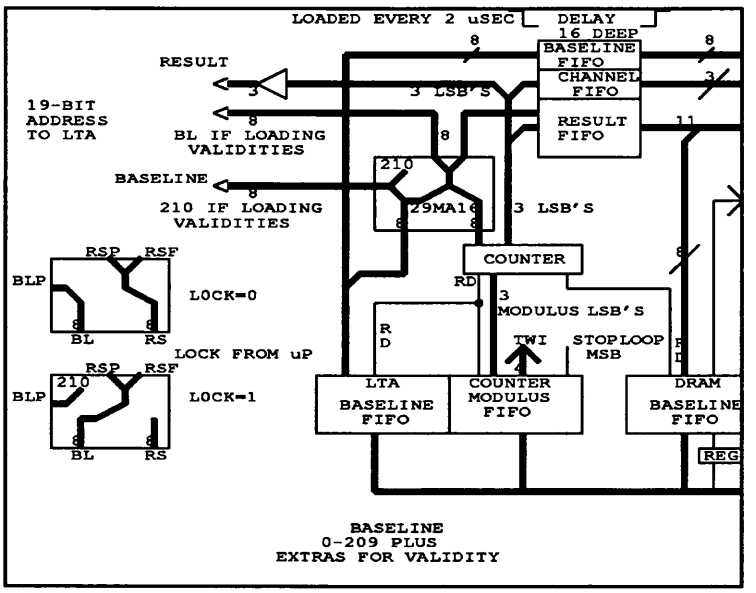


Figure 6 Baseline and Result Count Switch from Block Diagram

7.3 FIRPAL2A & 2B Baseline and Result Count Switch

The PALs are 7E and 8E in the center of schematic 1031d01. The 29MA16's provide the routing of the Baseline and Result Count. See the above section of 1031d25, the block diagram. The box labeled 29MA16 shows this switch. The diagrams in the lower left show the switch signal paths for the two modes. Lock=0 is the normal mode. Lock=1 is the validity mode.

The LTA Baseline Request is switched between coming from the LTA Baseline Fifo, or being jammed to 210 for Validity processing. In validity processing mode, the output of the LTA Baseline Fifo is routed to the 8 msbs of the Result Field request to the LTA. This forms the pseudo baselines to the LTA. Normally, the 8 msbs of the result field come from the Result Counter.

The Result field also goes to a delay fifo, then on to form the addressing to the DRAMs. There are separate outputs from the 29MA16, since the LTA request goes off the card.

7.4 FIRPAL3A & B & C 8 to 1 Mux PALs

These PALs are used to generate the ten bit address MUX to the DRAMs. Each PAL supplies two bits of the MUX. See the right side of 1031d01 for the schematic. Chips 9H, 10H, and 10G use FIRPAL3A. 11H uses FIRPAL3B. 11G uses FIRPAL3C.

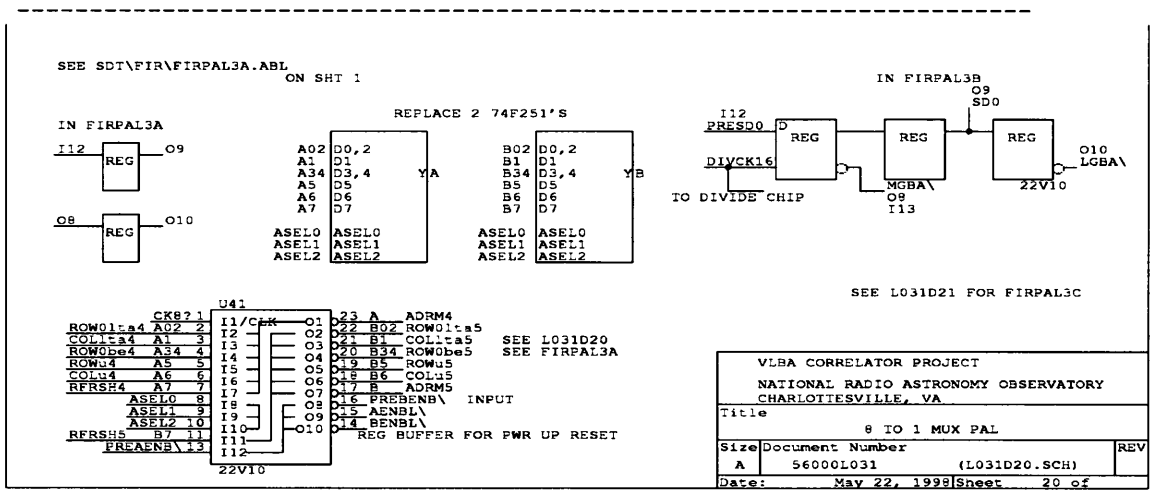


Figure 7 The 8 to 1 MUX PAL

See Figure 7, The 8 to 1 MUX PAL (1031d20) for FIRPAL A & B. Bits 0 through 5 use FIRPAL3A. Note the two extra registers provided as buffers.

FIRPAL3B is used for bits 6 and 8. The extra registers are used in association with divide chip timing.

A discrete 74F251 is used for bit 7.

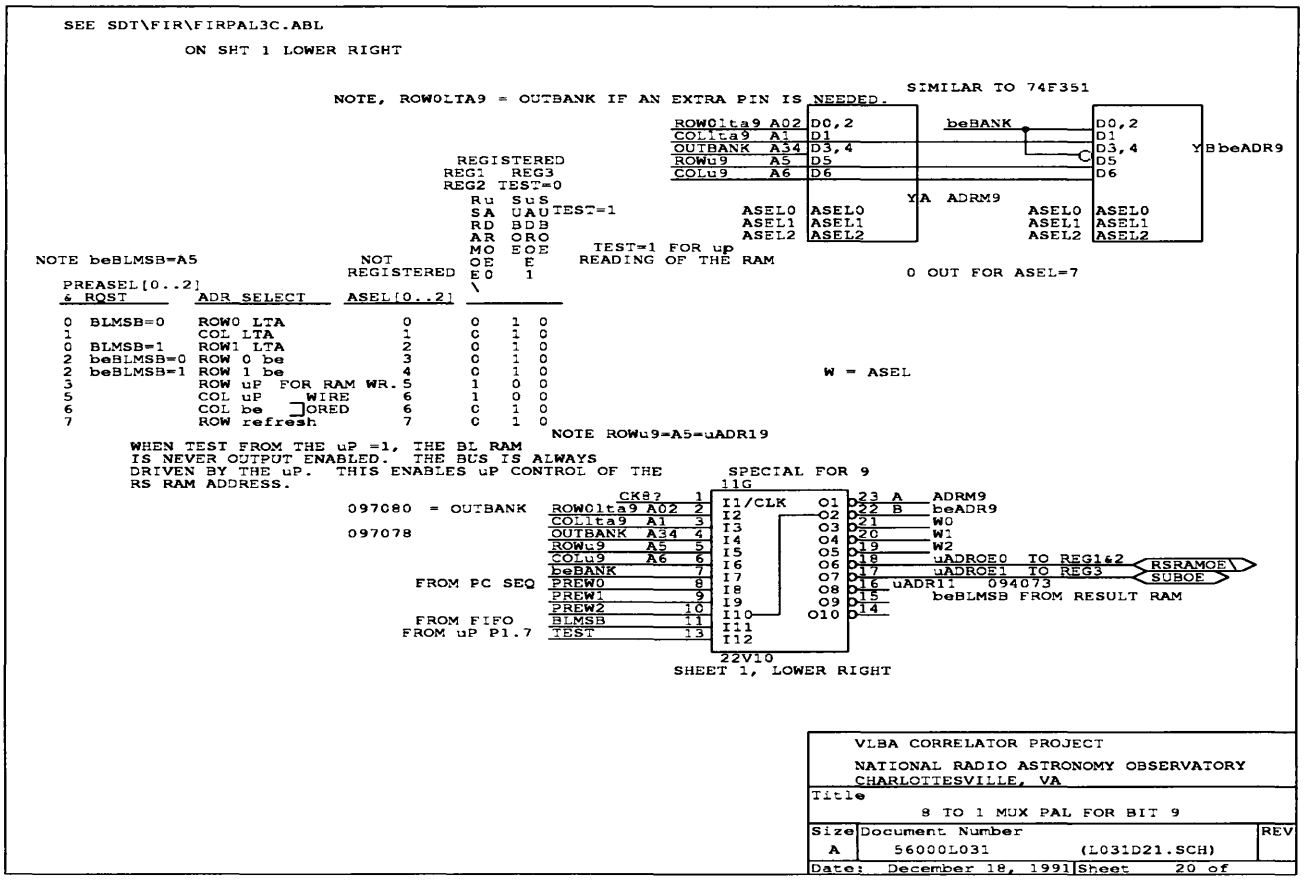


Figure 8 8 to 1 MUX PAL for Bit 9

FIRPALC is used for bit 9. See 1031d21 shown above. Bit 9 is handled differently, in that there is a separate bit 9 for the backend buffer address. The other 8 bits are shared between the backend buffer and the rest of the DRAMs. For the backend, bit 9 toggles between the two backend banks, every 131 ms. For the main DRAMs, bit 9 controls the bank 0 vs. bank 2 selection.

7.5 FIRPAL4A Data Crossbar

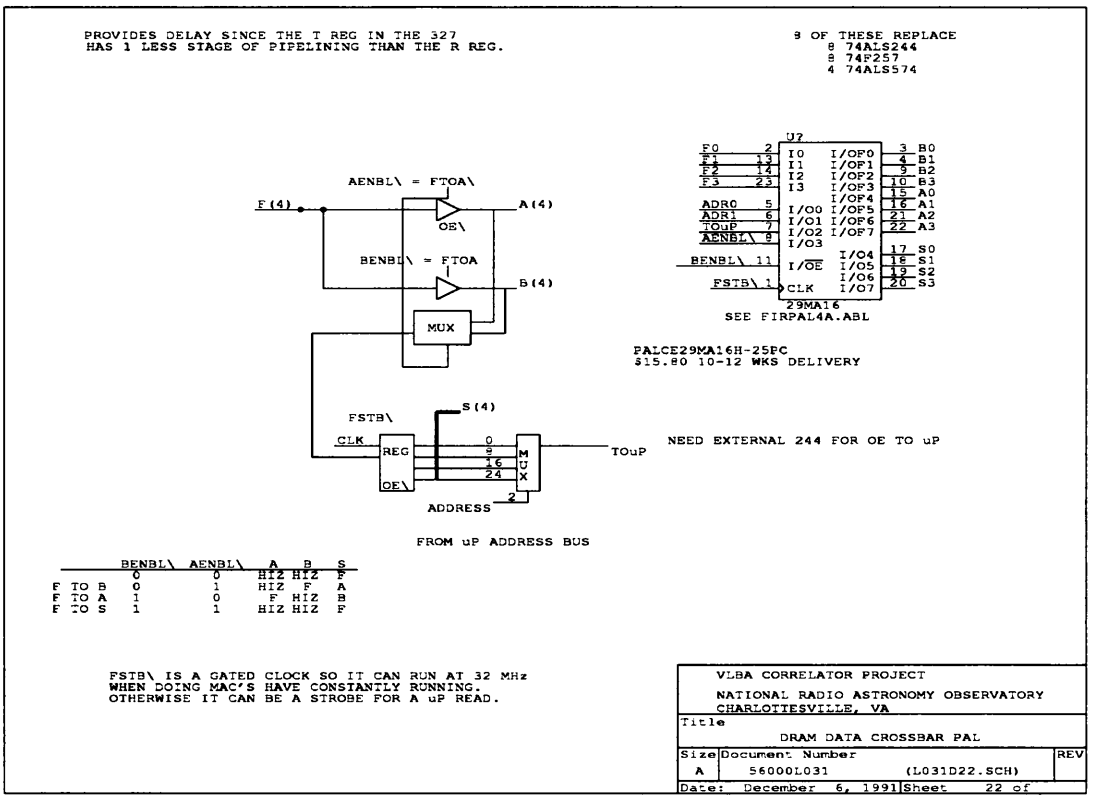


Figure 9 The DRAM Data Crossbar PAL

1031d22.sch presented here, shows the PAL logic. This uses the AMD29MA16 PAL, to take advantage of the 16 outputs available. The block diagram, 1031d25, shows the function of the crossbar. The upper left of 1031d06 shows the schematic. Chip locations are 24E, 25E, 26E, 27E, 28E, 29E, 30E, and 31E. The crossbar routes the 32 bit F bus output from the floating point chip (FPC). The output can go to the A or B busses to the DRAM banks, or to the input of the FPC via the S bus. The A or B busses from the DRAM can also go the S bus. The S bus can be read by the microprocessor via a 2 bit address, to select one of the four bits.

The 32 bit, crossbar is implemented in 4 bit slices, using 8 identical PALs. This makes an 8 bit bus to the microprocessor.

7.6 FIRPAL5 Address Multiplexer

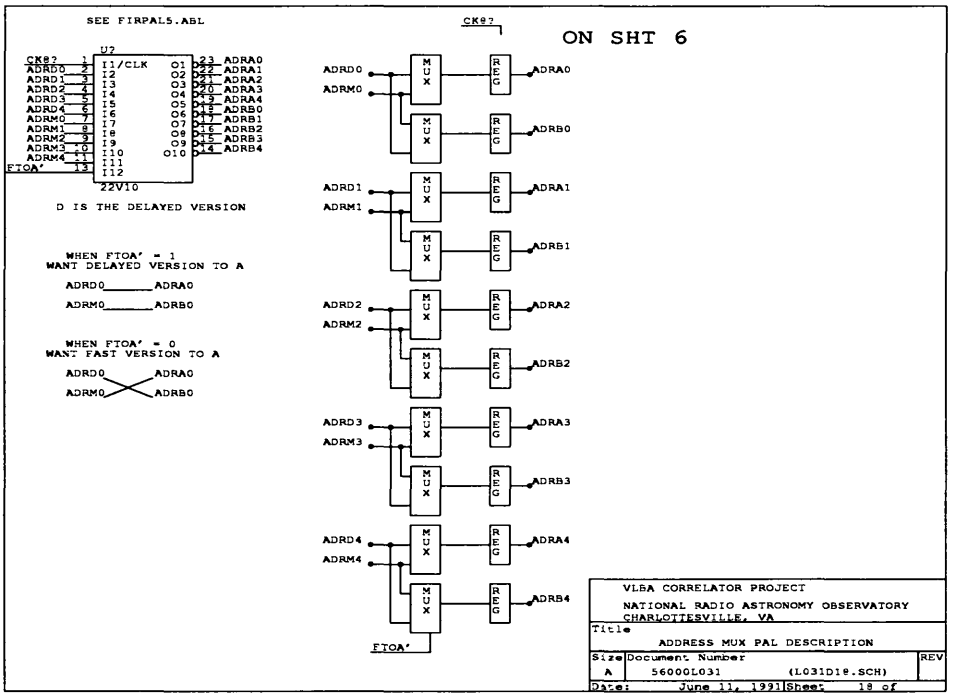


Figure 10 Address Mux PAL Description

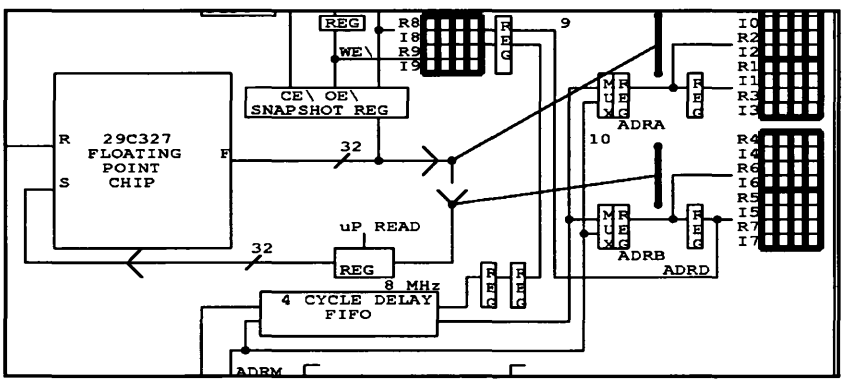


Figure 11 Address MUX Portion of the Block Diagram

See 1031d18.sch above in Figure 10, for the PAL logic. The Address Multiplexer can also be seen above in Figure 11, as the MUXs labeled ADRA and ADRB before the DRAMS.

The schematic 1031d06 shows the PALS in the upper left as chips 13G and 13H. The MUX switches between the address, and the address delayed four clock cycles. The Delay is needed, since writes occur four cycles later than reads. One DRAM bank is being read, while the other is being written to.

7.7 FIRPAL6B Ironics

This PAL handles the handshake for data transfers with the Ironics card. See Figure 12, below, for logic and timing. See the center of schematic 1031d06, chip 17B.

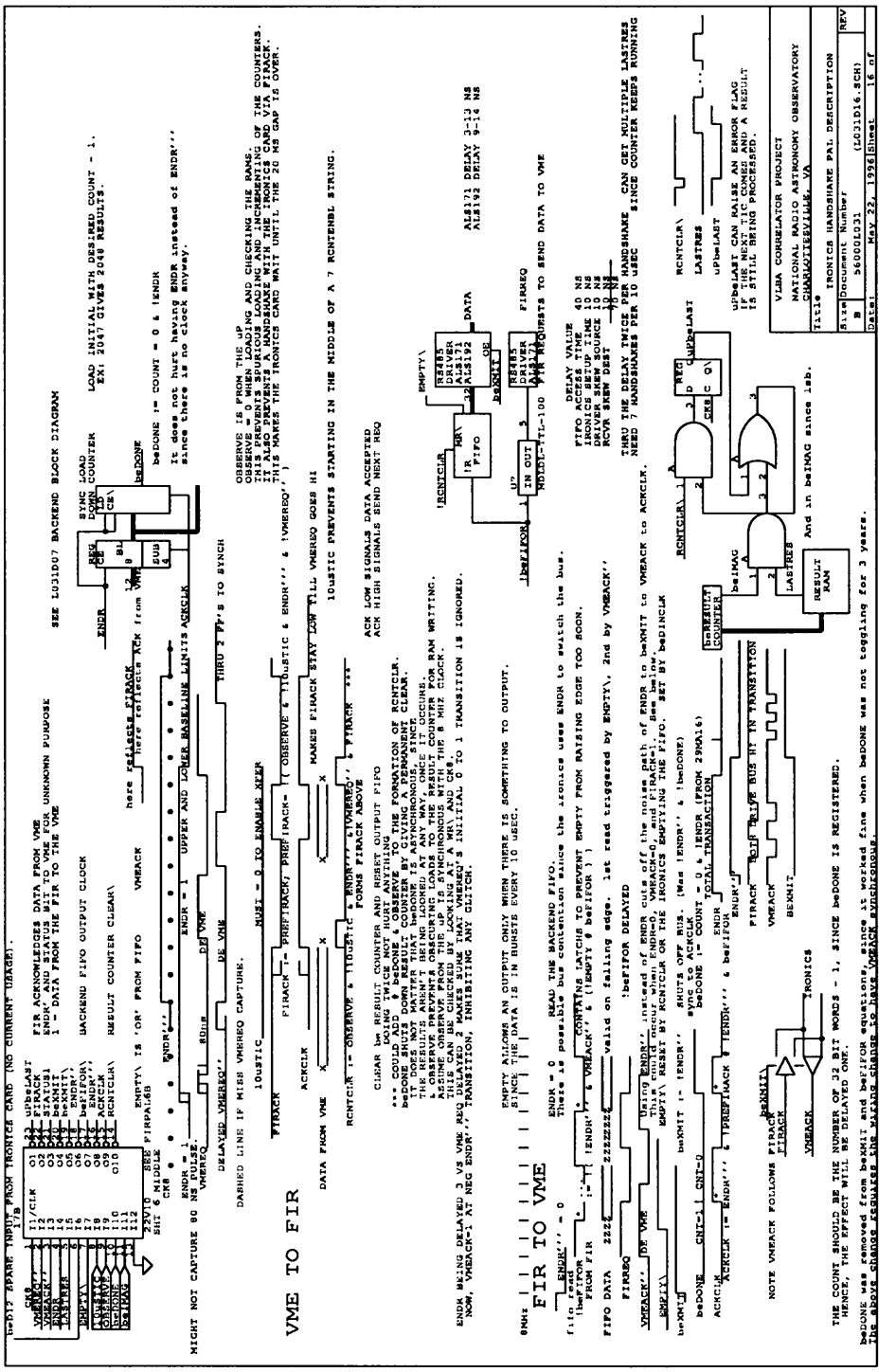


Figure 12 The Ironics Handshake PAL Description

7.8 FIRPAL7 Sequencer

See Figure 13, which is below. This is derived from 1031d19. Also see 1031d09 in Volume II, the Sequencer Block Diagram. The pal is on schematic 1031d02, Chip 11C, right center. ATOB and MEMOE\ are standard microprocessor control signals. They are independent of the rest of the PAL. The Page logic synchronizes the selecting a new Page of sequencer instructions. The Page request initially comes from the microprocessor, so is asynchronous to the sequencer.

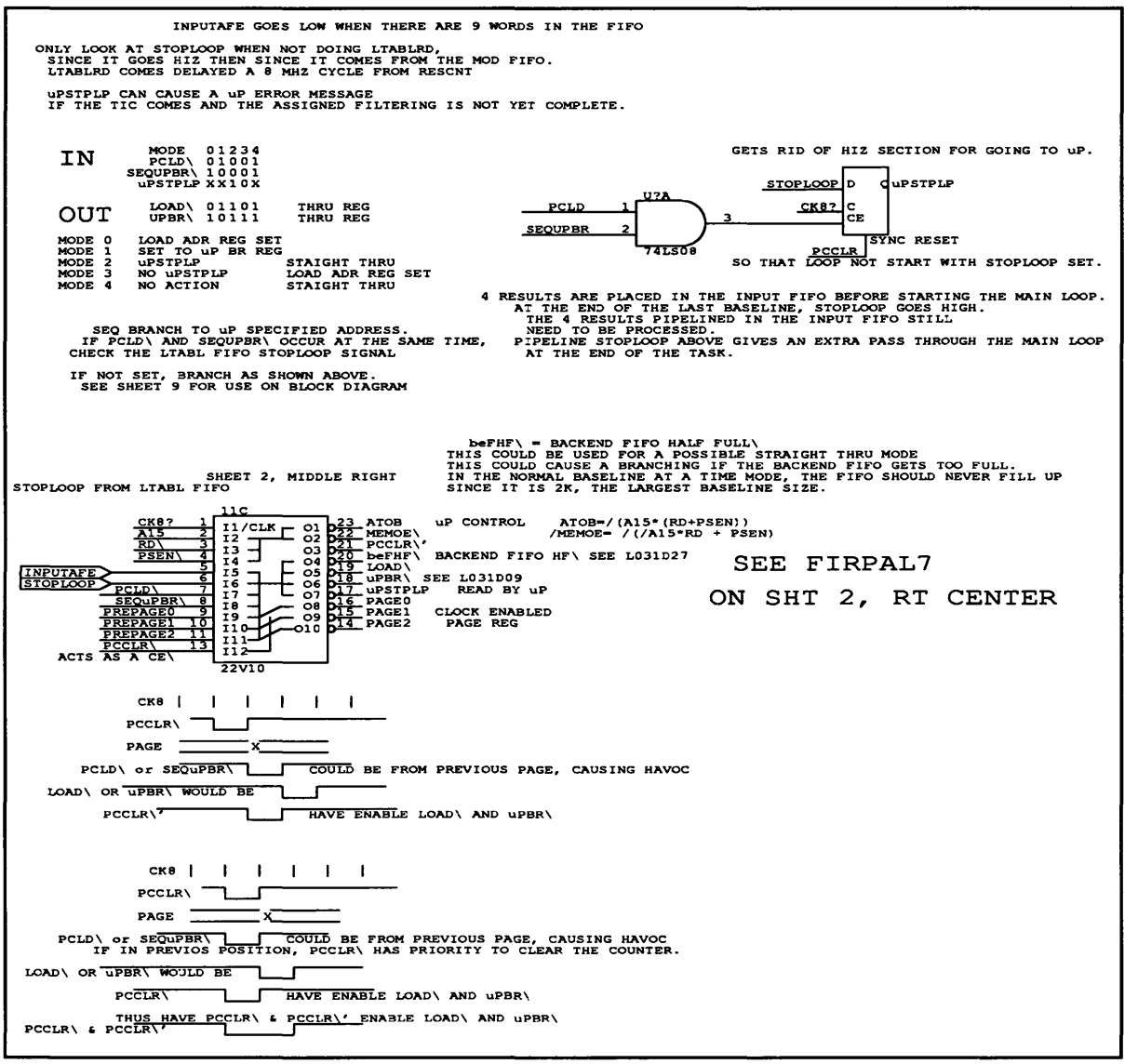


Figure 13 The Sequencer Control PAL Description

7.9 FIRPAL8 CAS

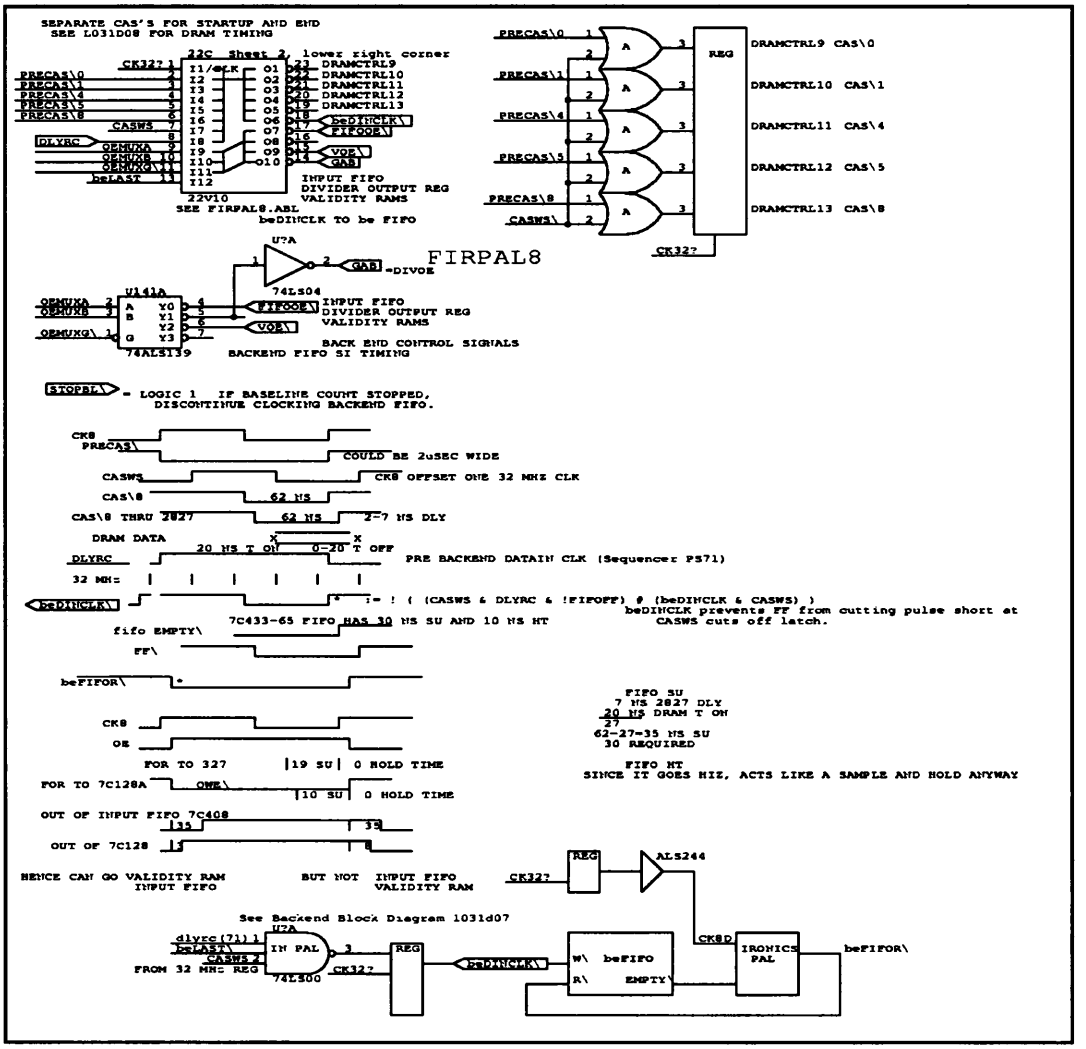


Figure 14 The CAS Handling Pal

The figure above shows the CAS PAL logic. See 1031d02, the lower right corner, Chip 22C, for the schematic. This PAL provides the buffering to generate the asymmetrical CAS pulses to the DRAMs. See 1031d08 for more details of the DRAM timing. The beDINCLK logic provides for writing data into the backend fifo, from the Output DRAM. See the backend block diagram, 1031d07.

The OEMUX logic selects the device to drive the R Input Bus of the FPC. See the block diagram, 1031d25. The choices are the Input Fifo, the Validity/Tap Weight Ram, or the Divide chip. The OEMUX address comes from the sequencer.

7.10 FIRPAL9A & B VME BL & Subarray Register / ACK Counter

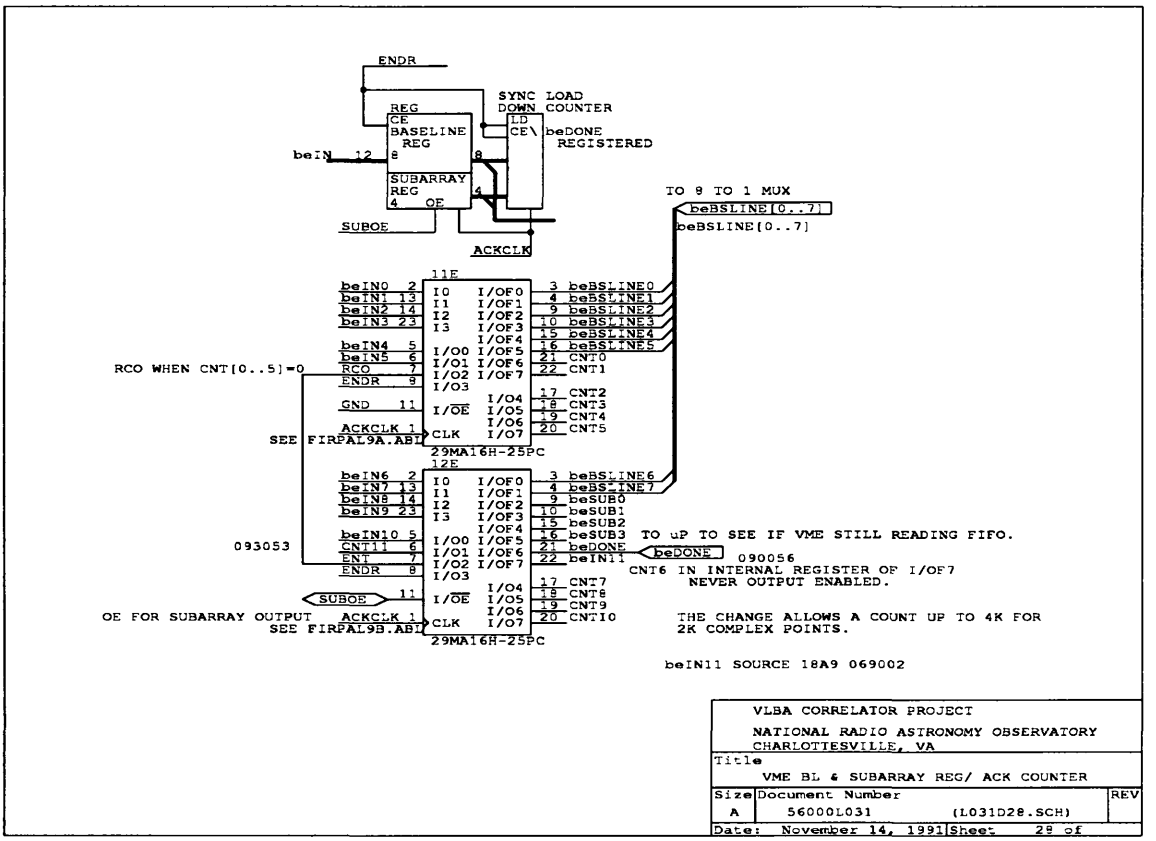


Figure 15 VME Baseline and Subarray Register / ACK Counter PAL

The figure, above, shows the PAL logic. See the Backend Block Diagram 1031d07. The schematic is on 1031d06, Chips 11E and 12E, on the lower center.

The 29MA16 PALs receive the 12 bit commands from the Ironics Card. The first 12 bit word is a result count. The second is an 8 bit baseline, and 4 bit subarray (aka BERRI).

The PALs count down the results, based on ACKCLK. The beDONE signals all the results have been received.

The requested Baseline and Subarray go to the address generation logic.

7.11 FIRPAL10 Clear Accumulator (CLAC)

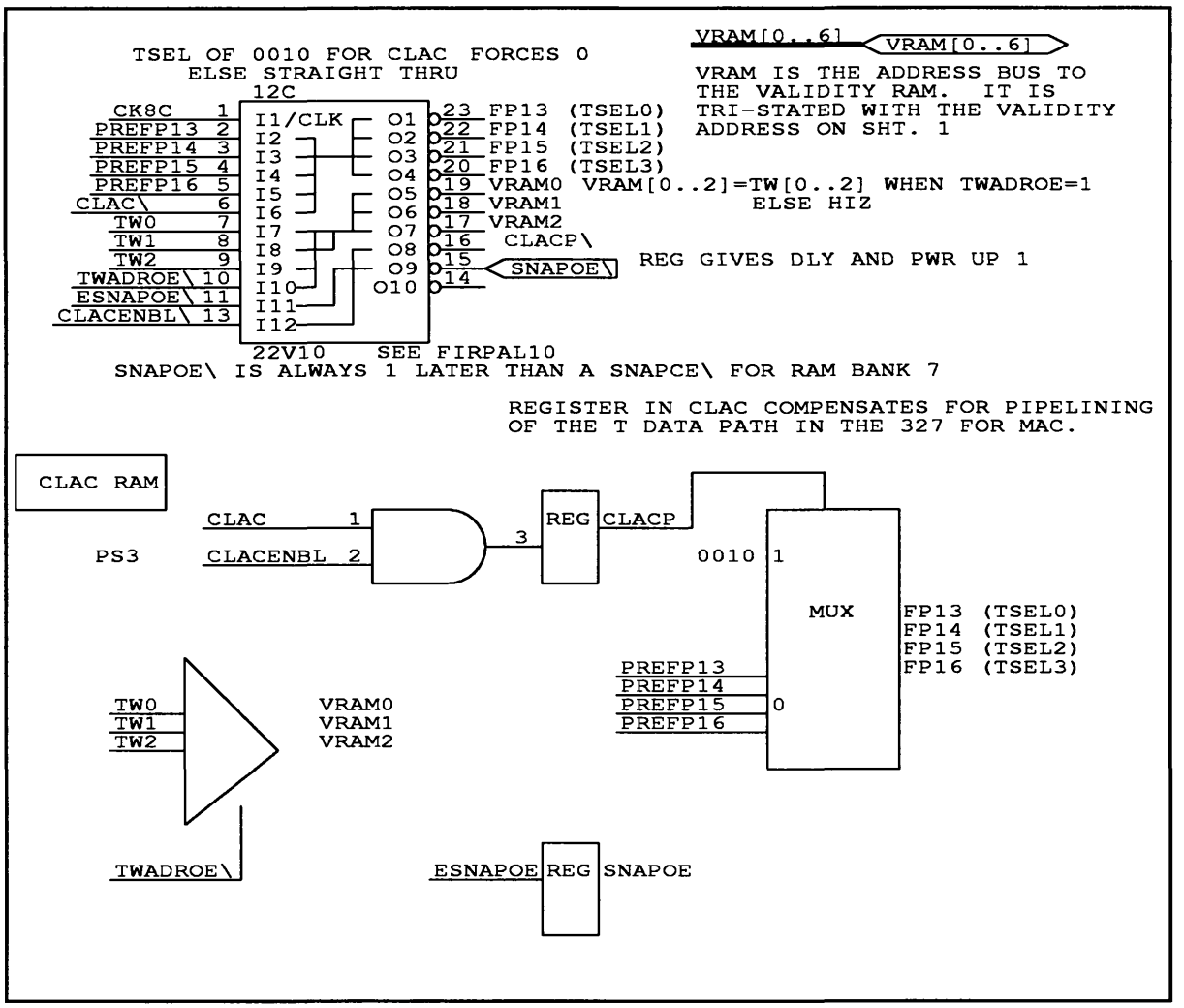


Figure 16 The CLAC PAL Logic

The figure, above, shows the PAL logic. The PAL can be found on the upper right of the schematic 1031d02, Chip 12C.

The CLAC signal functions by manipulating the 4 bit TSEL signal to the floating point chip (FPC). Normally the TSEL command, from the sequencer goes straight through the PAL. If CLAC and CLACENBL are asserted, the TSEL signal is jammed to 0010. This causes the FPC to add in zero, instead of the accumulated value.

For the TW[0..2] function, see the Block Diagram, 1031d25. The PAL provides the tristate buffering for the Tap Weight Address from the Sequencer, to the address bus of the Tap Weight/Validity Ram.

7.12 FIRPAL11 HCB Control

This is the standard 16L8 HCB Control Pal. See schematic on 1031d04, Chip 2C.

7.13 FIRPAL12 Delay beLAST

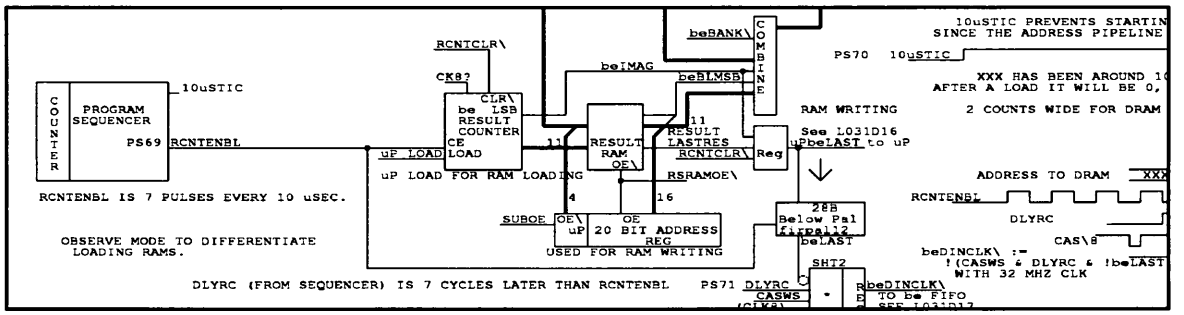


Figure 17 Lower Center of Backend Block Diagram, 1031d07

See the figure, pictured above. The Delay PAL, 28B, is to the right. See schematic 1031d02, chip 28B, in the lower left. This PAL delays the signal to inhibit writing into the backend fifo. The delay is needed to beLAST, since there is a 7 cycle delay between RCNTENBL and DLYRC.

8 File Locations

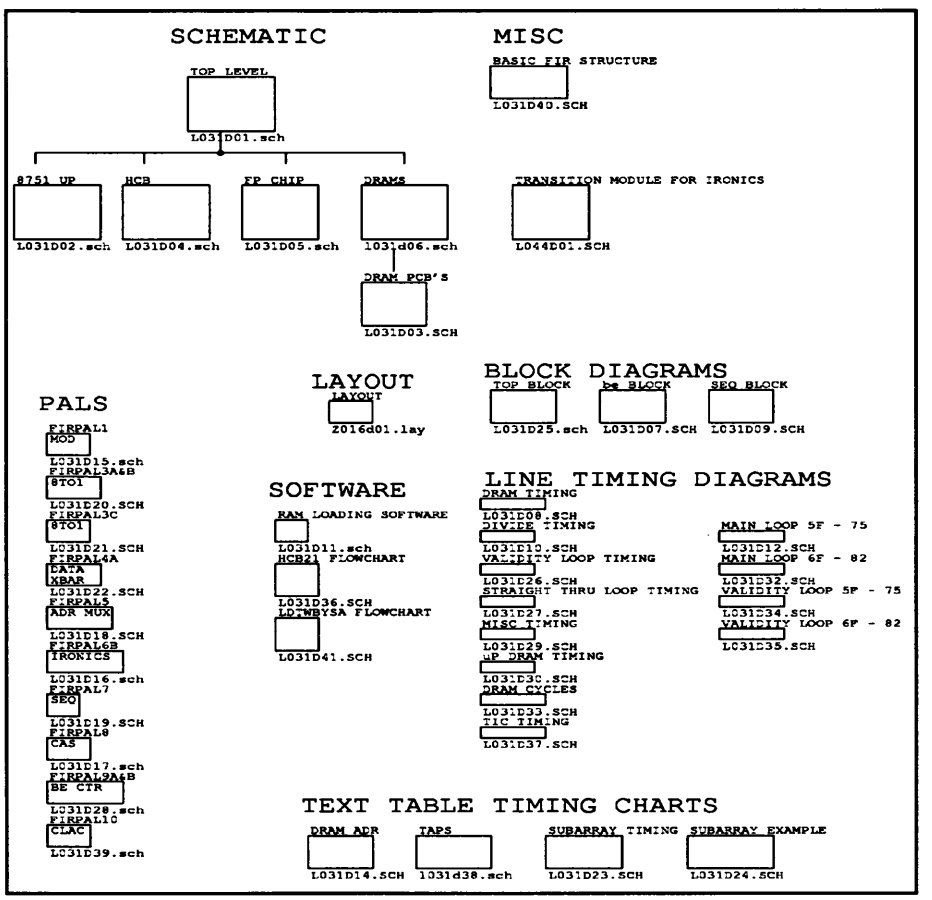


Figure 18 FIR Orcad Drawing Menu

The figure, on the previous page, shows the organization of the Orcad drawings. The Orcad drawings are in /home/azalea/corrdwgs/fir/sch/SCCS.

The file /vlbsoft/firasm/d023fir.hcb contains the hcb protocol at the microprocessor level. Also in Appendix I.

The file vlbsoft/hcb/hcbFir.c contains a prototype program for the operation of the filter card.

The file vlbsoft/hcb/hcbFirCore.c contains c functions for interfacing with the above hcb routines.

The file /vlbsoft/ironics/irFir.c contains irFirRead() for reading the ironics card.

Copies of the fir 8751 microprocessor assembly language code are in vlbsoft/firasm/*.asm.

The fir card sequencer is programmed via the file /vlbsoft/pal_prom/proms/firfpc/pg0.wk3, a Lotus worksheet. On the worksheet, page 0 is miscellaneous functions, page 1 is the main loop, page 2 is the normalized validity loop, and page 3 is the non-normalized validity loop. The macro alt-p generates the print file fir.prn. Executing a c program as "fircode fir.prn", causes the 13 binary files to be generated for the roms. Running "seq" in E:\allpro.v22\jg sets up the prom programmer for these files.

The abel files for the pals are firpal1.abl thru firpal11.abl in vlbsoft/pal_prom/pals/fir . Running "allpro 29ma16, 122v10, or ti22v10" in E:\allpro.v22\jg sets up the prom programmer for these files.

9 Software for Exercising the FIR

9.1 *firSST*

firSST(), in hcbtest/hcbFirTest.c, runs a multiple subarray, multiple tap weight, filter. The different tap weight indexes are made to represent different filter sizes. The output of 1024 tics are stored. Random data was written into the LTA each tic. An fft, done on the output 1024 tics, reveals the filter characteristics. Summing the ffts from multiple spectral points together gives less noisy results. The file firout in the delivery area is generated. firout is the file of 5 graphs intended for xvgr. In the joeenv/firasm, run diff /deliver/jgreenbe/firout firoutstd, to check for correctness. Note, the filter shapes plotted need to be shifted by the decimation factor, for 16, 32 and 64 tap weight filters.

9.2 *firValSST*

Also in hcbtest/hcbFirTest.c is firValSST(outputdata). If outputdata is 1, output data, otherwise output validities. firValSST runs a multiple subarray, multiple tap weight, filter. The file firout in the delivery area is generated. firout contains the outputs which are the fixed data or validities multiplied by incrementing tap weights. This gives the output of, validity or data value times the summation of the tap weights. In the joeenv/firasm, run diff /deliver/jgreenbe/firout firoutval.

9.3 *bldbsln*

In the startup script, /home/ccc/vlb/vwlog/jgreenbe/hcbtst.both, call the routine bldbsln(), to define the array bsln[][] which gives the baseline based on two stations. I call the routines defstatlwr() and defstatupr() which are in hcbFir.c, to initialize the array statno[20] with the stations of interest.

9.4 firRun

firRun runs a typical filter sequence and prints out results.

firRun(task,numstn,channel,tap weight index) in hcb/hcbFir.c

task - If task<2, this does functions that have to be done at least once, and writes the output.dat files. if task<1, this writes a sequence of number's in the LTA via the hcb. Thus, in the correlator a value might be 1, so as not to alter the LTA.

numstn - The number of stations to generate pairs of baselines from. These are the first numstn values in the array statno[20], which was defined by defstatlwr() & defstatupr().

In firRun, channel and tap weight index will be the same for all baselines.

The ext variable incorr should be set to 1 for use in the correlator. It specifies, use the semaphore generated tic's, as opposed to the test fixture waiting for firena to change. The test fixture method wastes cpu time waiting in loops. Note incorr affects ticfc going to HCB21. The external variables, busnr and targetnr, should be set to the appropriate bus number and target numbers.

The external variable firrptcnt set equal to 1, caused endtic, 131 ms tics to be performed then the loop exited. If firrptcnt=0, blocks of endtic tics are performed indefinitely. Spawning firRun with firrptcnt=0 can have the job terminated by setting firrptcnt=1.

The function RRXlate in hcbFir.c loads the result ram so as to translate the fir output into spectral point order. The results are read from the LTA with the channel as the lsbs.

The function firdump in hcbFir.c reads the ironics card, and writes a file for a pair of stations, in Chuck's backend processor format. The function syncdump in hcbFir.c is meant to call firdump from command level, while firRun is spawned looping with firrptcnt=0.

Now, I'll talk about the features of firRun, the main routine for operating the fir. This document is meant as an overview instead of supplying all the details.

Tap weights are included for testing 8,16,32, and 64 tap filters. Only the lower halves of the tap weight arrays are initialized. The function mirror, defined in hcbFirTest.c, fills in the upper tap weights with the reflection of the lower array half. firInitOnStart defined in hcbFirCore.c initializes the microprocessor. Dividend is the dividend value to be used in normalizing validities.

All the tap weights are downloaded to the microprocessor memory, and have their checksums calculated, by the function DwldCkTaps, which is defined in hcbFirTest.c. The different size filters go in different tap weight indexes. Notice in tap weight index 0, saying nt, the number of taps, = 0 gives straight thru mode. li = LTA integration time of 1,2, or 4 tics. li does not matter for straight thru mode.

Each tic, the fir control fifos need to be downloaded, for both processing the validities, and the main baselines, to be filtered. Information in the fifos is the LTA baseline, the modulus, the tap weight index, the dram baseline, and stoploop. Since the LTA has 210, 2048 result baselines, and the fir has 128, 2048 result baselines, dynamic baseline allocation is necessary.

The validities are always processed first. In processing the validities, LOCK=1 sets the baseline to the LTA to 210, and routes the baseline from the LTA baseline fifo to the result field. See 1031d25, the FIR Block Diagram. The dram baseline is set to 127. In firRun, the fifobufs are defined initially, then identically downloaded every tic by firLdFifos() in hcbFirCore.c. This assumes the amount of work to be done is small enough, to all be done each tic. More complex schemes could download different baselines each tic.

The routine firStartMainLp(ticfc,busnr,targetnr) is defined in d023fir.hcb as HCB21. It starts the processing for each tic's worth of baselines, having been loaded into the fifos. One function of the argument ticfc, is to say if the fifo list contains all normalized or non-normalized validities baselines, or lists of both. Baselines with validities to be normalized have the validity divided into the dividend, and stored in the validity ram for later use. If a baseline is to be non-normalized, unity is stored in the validity ram. valfifobuf contains all 210 baselines and can be used to make all the baselines normalized or non-normalized, determined by ticfc. If a mixture of normalized and non-normalized baselines is desired, place the baselines to be normalized in val2fifobuf and the non-normalized baselines in val3fifobuf. ((ticfc & 6) == 6) says to then expect and process the two fifobufs. In firRun, the even baselines went in val2fifobuf and the odd in val3fifobuf. For validity fifo buffers, the first byte is the dram baseline to hold validities of 127. Then are the pairs of LTA baselines, and the byte with tap weight index, mod, and stoploop. mod is always 0 for validities.

fifobuf is generated to give the instructions for processing the main baselines, that is baselines other than the validities. Each baseline has a set of 3 bytes. The first byte is the LTA baseline. Second is the memory mapped dram baseline number. Third is the byte with tap weight index, mod and stoploop. A mod of 1,2,3,4,5,6, or 7 specifies 32,64,128,256,512,1024, or 2048 results per baseline, respectively. Remember that the 0,1,2 bits (or 3 lsb's) of the result number are the channel number. The assignment of the lsb's is determined in the LTA ram tables. See vlbsoft/ltaasm/d024lta.hcb. The stoploop bit is always automatically assigned by the microprocessor software in HCB16, firLdFifos.

The loop with the index of looping is the main filter loop. The inner loop processes endtic tics at a time. When filtering, it is important to end after a number of tics equal to an integral multiple of the number of tap weights, so that the correct value is read out. Since the filter reduces the data output rate, erroneous values will be present for tics that are not supposed to generate an output.

Doing a semTake (tickExtern.bigTick, wdog_cnt), when incorr==1, makes the software wait until the beginning of a tic, without wasting cpu time. After the 2nd semTake, firuPStatus uses hcb17 to read uPINT\0. uPINT\0 equals 0 while the fir sequencer is processing the baselines loaded into the fifos. When done, the sequencer goes into an idle loop, and it sets uPINT\0=1. If the next tic comes around and uPINT\0 (a.k.a. statusptr[1]) != 1, then too much work was given to the fir to be done in a single tic. This condition could also set bit 1 of errorcode, returned by hcb21.

firStartMainLp is defined more fully in d023fir.hcb, in Appendix I, as HCB21. It returns an errorcode which should be 0 for no error. The appropriate 8 tap weights, for each tap weight, index are downloaded from the microprocessor memory. Stored up sections of result ram data could be downloaded.

Once firStartMainLp has started the fir sequencer working on the baselines in the fifo's, other tasks can be done by the real time system, while waiting for the next tic. An essential task is loading the fifos with the baselines for the next tic. Since the fifo is only reset in hcb20, this does not effect the current tic's usage of the fifos. Optionally, hcb10 and hcb11 can be used to download new tap weights into the microprocessor memory, on the fly. Also, optionally hcb11 and hcb12 can be used to download a section of the result ram into the microprocessor memory, on the fly. HCB21 transfers the result ram data automatically from the microprocessor memory to the result ram. The transfer takes place in the pause at the beginning of a tic. Be sure to complete these tasks before the next tic starts, or the next call to efcPend could cause a whole extra tic to be waited.

9.5 Backend Access

firRun, or a similar task, should be spawned to keep the fir busy every tic.

Reading the backend via the ironics card can be run fairly independently. The function syncdump() can be run while firRun is spawned in an infinite loop with firrptcnt=0. semTake (tickExtern.bigTick, wdog_cnt) syncs to

the beginning of a tic. firdump() calls irFirRead() which gives a pointer to the data from the ironics card. The backend has been described in detail in Section 3.12.

10 Transition Module for the Ironics Card

The transition module provides the drivers to drive the long cable between the Ironics card and the FIR. The transition module is located in the back of the VME chassis, in back of the Ironics card. See the schematics in Volume II.

J1 and J2 connect to cables to the FIR Card. J3 connects to the Ironics card. There is a connector for power. There is a ground strap provided.

The spare card is used as a hot spare in the back of the v47a VME chassis.

11 The Ironics Card

11.1 Ironics Hardware Description

See the description in APPENDIX IV Ironics IV-3272-PIO Parallel Interface Daughter Board. This is a daughter board on the Ironics board. This document provides a detailed description of the data exchange handshake.

11.2 Ironics Software Description

The ironics software is under the vlbsoft/ironics directory

11.2.1 *irFirRead(firstbsln,nrbslines,nrresults,subarray)

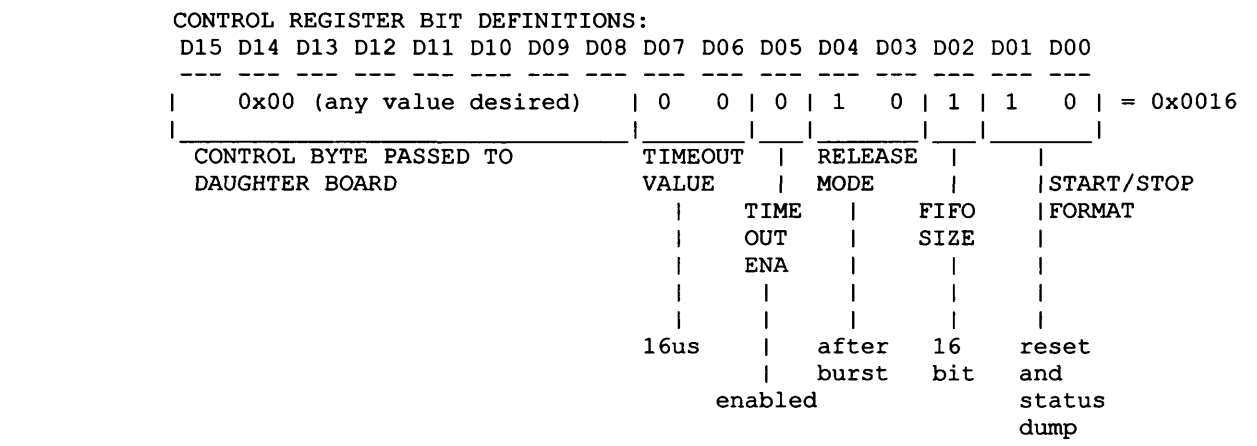
in hcb/hcbFir.c

```
int firstbsln;  /* first baseline              */
int nrbslines;  /* nr of sequential baselines */
int nrresults;  /* nr of complex results      */
int subarray;   /* sub-array affiliation for these baselines */
{
```

This Function reads a section of the backend buffer, and returns a pointer. The calling function must free the pointer. A fir task must have previously written the data into the backend buffer.

11.2.2 irReset()

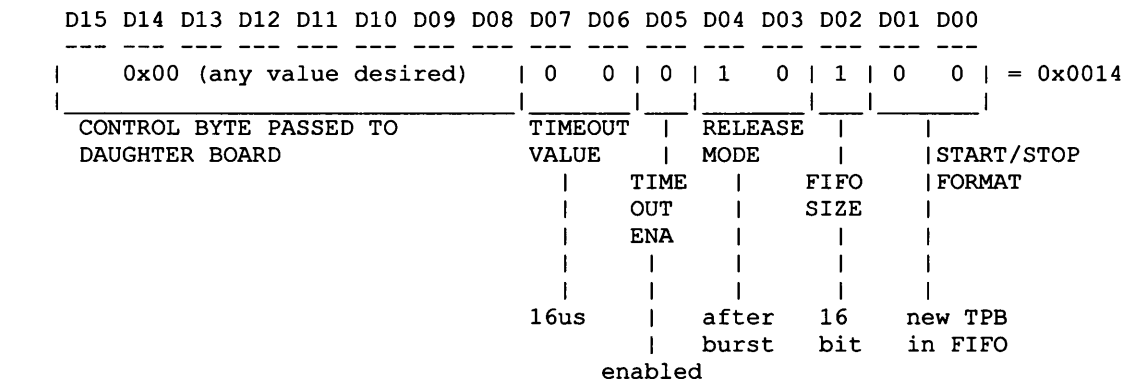
In irReset(), a control word with the RESET stop format will be written to the Ironics control register, followed by a STOP command to the Ironics start/stop register.



11.2.3 irStartDma()

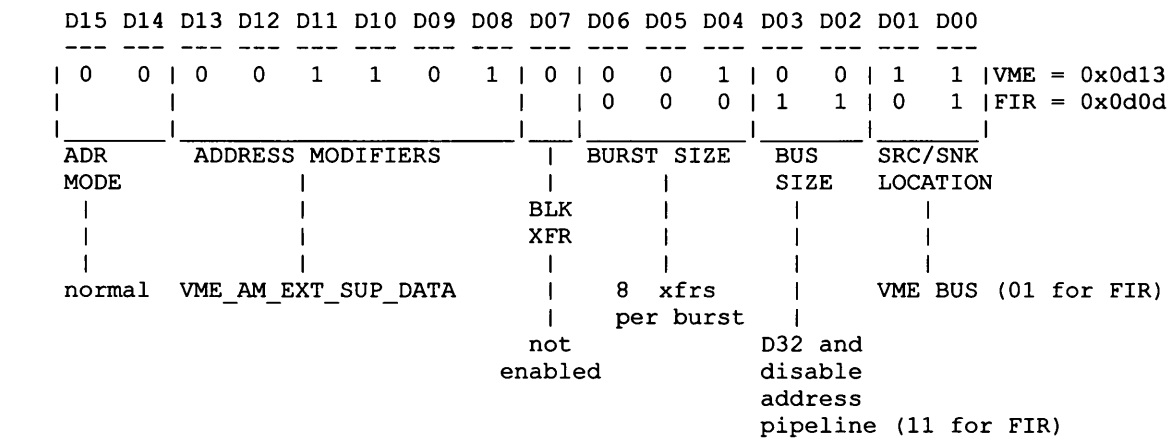
In irStartDma(), a control word with the proper start format will be written, then the START/STOP register will be given a start.

CONTROL WORD BIT DEFINITIONS



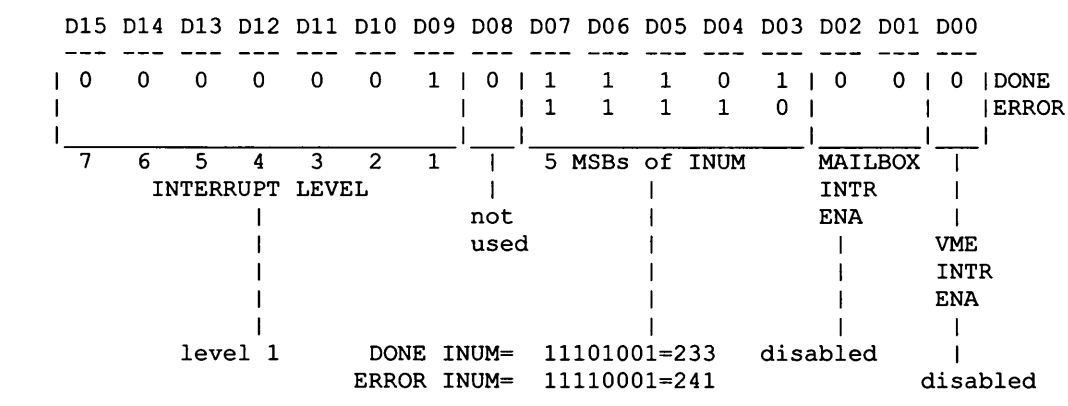
11.2.4 Other Ironics Control Words

SOURCE/SINK ATTRIBUTE WORD BIT DEFINITIONS



Thus, TOFIR = 0x0d130d0d
FROMFIR = 0x0d0d0d13

INTERRUPT VECTOR WORD BIT DEFINITIONS



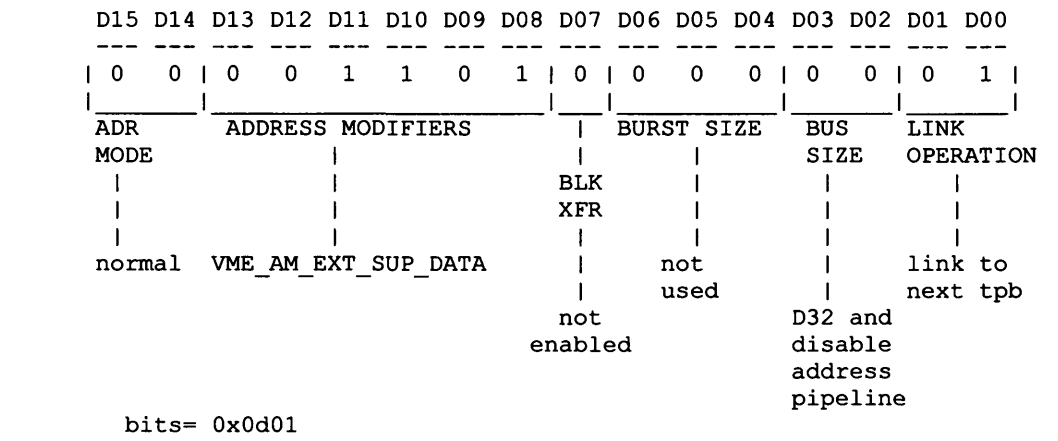
The interrupt number released by the Ironics card to the VME bus in

response to an interrupt acknowledge, is composed of the 5 bits from the interrupt vector word in the 5 MS bits, and the three bit interrupt level code in the 3 LS bits. Thus the two interrupt numbers 233 and 241 above are what this test plans to use when VME interrupts are enabled. We will start without enabling the interrupts.

DONE = 0x02e8 ERROR = 0x02f0 for interrupts not enabled
DONE = 0x02e9 ERROR = 0x02f1 for interrupts enabled

Thus, NODONEINTR = 0x02e802f1
DONEINTR = 0x02e902f1

LINK ATTRIBUTE WORD



bits= 0x0d01

I/O CONTROL WORD

These bits are for use with the interface daughter board and are defined by the daughter board's specifications. The exception is bit 1. This bit set (equal to 1) defines this TPB as tagged. A tagged TPB indicates that data is to be processed by a downloaded user program.

No reference to this field has been found in the PIO manual, so set all = 0.

Thus the following assignments are used:

```
testtpb.law_iocw = 0x0d010000;
testtpb.reserved = NULL;
```

12 The FIR Test Fixture

12.1 Introduction

The FIR Test Fixture provides an environment for testing an LTA and FIR wired together. The Test Fixture has its own power supply, and a single control board. Connectors are provided for interfacing with the HCB and the Ironics Board. Serial port connectors are provided to allow terminals to talk to the LTA and FIR microprocessors.

Note the newer Testbed allows testing the LTA and FIR together also. The only advantage the FIR Test Fixture offers is it allows easier probing of the cards.

12.2 FIR Test Fixture Control Logic

See 1045d01.sch in Volume II. The Control Logic Card has a 32 MHz Oscillator to provide a clock. There is a sequencer for providing control signals. A jumper cable connects to the HCB connection at the top of the LTA. There is a MAX232 to drive the RS232 lines to the terminals for the LTA and FIR.

12.3 FIR Test Fixture Wiring

See 1045d02.sch in Volume II. The LTA and FIR are interconnected as in the system, as are the Ironics and HCB Cables. In the correlator, the interface is identical for the FIR, since the FIR only talks to the LTA, HCB, terminal, and Ironics Card. The Test Fixture Control Logic Card supplies timing and the RS232 interface.

A jumper plug is provided to allow Test Fixture use without an LTA being plugged in. The plug goes on the 10P1 slot. This routes signals, which are normally supplied by the LTA, from the Test Fixture Control Logic.

12.4 Using the FIR Test Fixture

Connect the Test Fixture to a v47a HCB. Connect the Ironics cable from the transition module to the front of the Test Fixture. On v47a, initialize with the <load.tstfix script. Then initialize the test fixture with the <ldlf.cmd script. The standard RTS software for the FIR can then be run.

13 Troubleshooting the FIR

This section describes tests to determine if the FIR is working properly.

13.1 Microprocessor Based Tests

13.1.1 Rom Based Tests

These tests are in the routine test.asm. They are stored in the microprocessor onboard rom. They are accessed from a terminal hooked to the microprocessor.

TM - test ram memory. Display 00 RAM ERRORS in a successful test.

T0 - Display a selected portion of DRAM. To use, store the desired 20 bit DRAM address in microprocessor addresses 7001, 7002, and 7003.

T2 - Test result ram. Loops and should give messages RESULT RAM WRITE COMPLETE and RESULT RAM ERROR CHECK COMPLETE.

T4 - Do division on the value in 7000,1,2,3, and display the results. Results are the inverse, and the product of the inverse times the original which should give 1 (3f800000)

T5 - Test validity ram straight thru the fifo. Reads VALIDITY RAM TEST COMPLETE.

T6 - Test validity ram using a queue in the fifo. Reads FIFOED VRAM TEST COMPLETE.

T8 - Test tap weight ram. Reads TAP WEIGHTS COMPLETE.

TA - Full test of 8 DRAM banks. Says DRAM TEST COMPLETE, after about 4 minutes, then loops again.

TC - Display LTA contents for the baseline in 7000. The first 0x16 results are displayed.

13.1.2 Microprocessor RAM based tests

These tests are in the routine ramtest.asm. They are stored in the microprocessor external ram. (Need to have the ram downloaded by <ldlf.cmd) They are accessed from a terminal hooked to the microprocessor.

TF 0 - Run page 1 main loop in a tight loop.

TF 1 - Run page 1 and look at results. See corrdwgs/fir/doc/product.txt for a version of the output expected. Needs some translation.

TF 7 - Straight thru mode. Displays some results.

TF 9 - Validity filter test. Display 46000000's.

TF A - Display validity/tap weight ram contents.

13.2 Tests via the Real Time System

The following Real Time System routines can aid in troubleshooting the FIR.

13.2.1 firTestTaps

```
void firTestTaps(sa,nt,li,busnr,targetnr)
int sa; /* Subarray number 0-15 */
int nt; /* Number of tap weights, allowable values:
            0 for straight thru mode 8, 16, 32, or 64 */
int li; /* LTA Integration time.  1,2, or 4 tics. */
```

This downloads test tap weights to the microprocessor memory, then reads it back and calculates a checksum.

13.2.2 firTestRRLd

```
firTestRRLd (fc, busnr, targetnr) /* fc=0 for immediate compares. */
```

This downloads random numbers into the result ram, then reads back a checksum.

13.2.3 firSST

See Section 9.1 for more information.
firSST() in hcbtest/hcbFirTest.c

This runs a multiple subarray, multiple tap weight, filter. The file firout in the delivery area is generated. firout is the file of 5 graphs intended for xvgr. In the joeenv/firasm, run diff /deliver/jgreenbe/firout firoutstd. Note, the output waveforms need to be shifted by the decimation factor for 16, 32 and 64 tap weight filters.

13.2.4 firValSST

See Section 9.2 for more information.

```
firValSST(outputdata)

int outputdata; /* if 1, output data, otherwise output validities */
```

in hcbtest/hcbFirTest.c

This runs a multiple subarray, multiple tap weight, filter. The file firout in the delivery area is generated. firout contains the outputs which are the fixed data, or validities, multiplied by incrementing tap weights. This gives the output of, validity or data value, times the summation of the tap weights.

In the joeenv/firasm, run diff /deliver/jgreenbe/firout firoutval.

13.2.5 firRun

See Section 9.4 for more information.

```
int firRun(task,numstn,channel,subarray)
            int task; /* 0 for first call.  4 for later */
            int numstn; /* number of stations */
```

in hcb/hcbFir.c

This is the prototype filter package for in the system.

To test, command firRun 0,4. Then in the delivery area:
cmp s1012.dat stat10_12.dat.

This looks at a binary file in the format for Chuck's backend processor.

13.2.6 hcbTestLtaFir

```
hcbTestLtaFir(nrbls,nowrlta)

      int nrbls /* Number of lta baselines, max 210 */

      int nowrlta /* If !=0, don't write the lta again. */
```

This provides a good comprehensive test of the lta and fir. The lta and fir are tested by writing a sequence of numbers into the lta, and reading them out via the fir and ironics. It works in either the test fixture, the testbed, or system. The function without arguments performs the complete test. However, it takes a long time to write all the baselines into the LTA. Specify a smaller number of baselines, such as 15, to give a good abbreviated test

13.3 How to Read an LTA Baseline Via the FIR

13.3.1 How to Read an Fir Baseline

The backend, via the Ironics card, only reads FIR baselines. It is important to remember that the FIR baselines are assigned independently of the LTA baselines. If we want to read an LTA baseline, it must be first transferred to an FIR baselines, then read out.

To read out an FIR baseline use the function:

```
int irFirTest(firstbsln,nrbslines,nrresults,subarray,dsplyflag,rsltflag)
int firstbsln;  /* first baseline                */
int nrbslines;  /* nr of sequential baselines */
int nrresults;  /* nr of complex results       */
int subarray;   /* sub-array affiliation for these baselines */
int dsplyflag;  /* 0= Normal output.  1 = outputs additional information */
int rsltflag;   /* 0= dsply no results, !=0 dsply this many result.  Note
                   normally this number is nrbslines * nrresults. */
```

Example: irFirTest(0,3,4,0,0,12) will display 4 results from each of baselines 0, 1, and 2.

13.3.2 How to Read LTA Baselines via the FIR

In this example it is assumed the LTA and FIR have been loaded by using the ldlf.cmd script. Ldlf.cmd is in the vlbsoft/diagnostics directory under SCCS, and is delivered to the delivery area. If the testbed is being used, the scripts load.tstbed, followed by ldtstbed.cmd, are used to initialize the testbed, before the ldlf.cmd script.

To specify the LTA baseline to be read, use the function
defstatlwr(s0,s1,s2,s3,s4,s5,s6,s7,s8,s9)
This function specifies station numbers 0 through 19.

See the figure v0304b.blk for a table of Stations vs. LTA Baselines.

The Function: firRun(task,numstn,channel,tap weight index), in hcb/hcbFir.c, will be run to transfer the LTA results. For the arguments, task will be 0, numstn, number of stations. numstn reflects how many stations, defined in defstatlwr, starting with s0, should be looked at. The program looks at the baselines, reflecting all combinations of the stations considered, including self products.

Example:

defstatlwr 5,6,7 This says consider stations 5, 6 , and 7.

firrun 0,3 This says consider the first 3 stations from defstatlwr.

```
v47a-> firRun 0,3
The Real Time System outputs:
subarray=0
val2fc 14,mainfc 4,mod 7,task 0,stbl 0,endbl 70,
 actticfc 4,
2048 results
Result ram loaded in spectral point order
drambl 0, ltabl 90, stations 5 & 5
drambl 1, ltabl 91, stations 5 & 6
drambl 2, ltabl 92, stations 5 & 7
drambl 3, ltabl 105, stations 6 & 6
drambl 4, ltabl 106, stations 6 & 7
drambl 5, ltabl 119, stations 7 & 7
beBANK=FF, uPINT0=01, Reg8001=F9

Baselines 0-2, 4 results each
    9.000000e+01  42  B4  00  00          9.000000e+02  44  61  00  00
    9.800000e+01  42  C4  00  00          9.800000e+02  44  75  00  00
    1.060000e+02  42  D4  00  00          1.060000e+03  44  84  80  00
    1.140000e+02  42  E4  00  00          1.140000e+03  44  8E  80  00
    9.100000e+01  42  B6  00  00          9.100000e+02  44  63  80  00
    9.900000e+01  42  C6  00  00          9.900000e+02  44  77  80  00
    1.070000e+02  42  D6  00  00          1.070000e+03  44  85  C0  00
    1.150000e+02  42  E6  00  00          1.150000e+03  44  8F  C0  00
    9.200000e+01  42  B8  00  00          9.200000e+02  44  66  00  00
    1.000000e+02  42  C8  00  00          1.000000e+03  44  7A  00  00
    1.080000e+02  42  D8  00  00          1.080000e+03  44  87  00  00
    1.160000e+02  42  E8  00  00          1.160000e+03  44  91  00  00
```

Note the dram baselines are assigned consecutively, while the lta baselines are based on the pairs of stations.

firRun performs a irFirTest(0,3,4,0,0,12) command at the end, to give the first four results of dram baselines 0, 1, and 2. Each line represents the real and imaginary part of the number, along with a hex representation.

The values in the baseline were written by firRun, using the routine wrfirtst, defined in vlbsoft/hcbtest/hctLtaTestFix.c:

```
int wrfirtst(bsline,nrresults,offset)
int bsline;  /* LTA baseline at which to write some values */
int nrresults; /* starting at result nr zero, write this many values */
double offset; /* convert this to a float and write a sequence as follows:
*/
                /*    *(result).real= resultnr + offset       */
                /*    *(result).imag= resultnr + offset * 10 */
```

A value of offset equal to the LTA baseline number is used.

Having the task argument, in firRun, equal to 1, will prevent writing to the LTA from occurring. This could be useful for seeing what was already in the LTA. Example: firRun 1,3.

Subsequent irFirTest commands can be issued for further results.

Example:

```
v47a-> irFirTest 1,1,6,0,0,6
    9.100000e+01  42  B6  00  00          9.100000e+02  44  63  80  00
    9.900000e+01  42  C6  00  00          9.900000e+02  44  77  80  00
    1.070000e+02  42  D6  00  00          1.070000e+03  44  85  C0  00
    1.150000e+02  42  E6  00  00          1.150000e+03  44  8F  C0  00
    1.230000e+02  42  F6  00  00          1.230000e+03  44  99  C0  00
    1.310000e+02  43  03  00  00          1.310000e+03  44  A3  C0  00
```

APPENDIX I FIR HCB Protocol

FIR HCB PROTOCOL NRAO DRAWING #A56000D023
TEXT FILE In component firasm: file d023fir.hcb

VLBA CORRELATOR PROJECT
NATIONAL RADIO ASTRONOMY OBSERVATORY
CHARLOTTESVILLE, VA 22903

I) SUMMARY OF ALL FUNCTION CODES
All function codes below are in HEX:

TRANSFER TYPE	PROTOCOL	FREQ OF OCCURRENCE
NOP:	00 ZZ (ZZ = RQRD DUMMY BYTE)	TEST
WRITE MEMORY:	01 00 00 AA AA CC CC DD----DD	INIT
READ MEMORY:	02 00 00 AA AA CC CC	TEST
CALCULATE CHECKSUM:	03 00 00 AA AA CC CC	INIT
DWNLDTAPS	10 TWI NT LI AA AA AA AA BB BB BB BB ---	OBS
CALCTWCHECKSUM	11 TWI (2 bytes returned, msbyte 1st)	OBS
LDRESULTRAM	12 NN BERRI FC AA AA BB BB ---	OBS
CALCRRAMCS (FC=0)	13 NN BERRI FC (Two byte CS CS returned)	TEST
CALCRRAMCS (FC=1)	13 NN BERRI FC CS CS	OBS
LDALLRRAM	14 ZZ	TEST
LDVRAMONE	15 ZZ	TEST
LDFIFOS (FC, bit3=1)	16 FC NN VD LA MA LB MB ---	OBS
LDFIFOS (FC, bit3=0)	16 FC NN LA DA MA LB DB MB ---	OBS
UPSTATUS	17 (3 bytes returned)	OBS
INITONSTART	20 AA AA AA AA	OBS
STARTMAINLP	21 FC (1 byte error code returned)	OBS

****
Checksum is of the form:

```
   AAAAAAAA AAAAAAAA     ACCUMULATED SUM
 + 00000000 BBBBBBBB     CURRENT BYTE BEING SUMMED INTO CHECKSUM
 -------------------
   AAAAAAAA AAAAAAAA     NEW ACCUMULATED SUM
```

****
FUNCTION DURATIONS
The below table shows the name for the function code in the include file hcbFuncCodes.h. Also shown are the times in milliseconds from a survey of how long the functions take. Approximate data rates in KBytes/second are shown. There are potential problems for functions that have an equivalent transfer rate of less than 50 KBytes/sec, and take more than 10 ms. For a detailed explanation of the measurements taken see /home/azalea1/staff/jgreenbe/notes/survey.txt. Functions labled INIT below are only used during initialization, where timing is not critical. The timing of functions labeled TEST is not of concern when observing. Data rates are not give where minimal data is transferred.

```
INCLUDE FILE   FC      TIME    RATE
  NAME                 (ms)    (KBytes/sec)

Straight Through Mode
DWNLDTAPS    0x10      0.170
CALCTWCKSUM 0x11       0.128

8 Tap Filter - Times vary with the number of tap weights downloaded
DWNLDTAPS    0x10      0.598   53.5
CALCTWCKSUM 0x11       0.285

64 Tap Filter
DWNLDTAPS    0x10      2.988   85.6
CALCTWCKSUM 0x11       1.895

For a 512 byte section
LDRESULTRAM 0x12       15.6    32.8
CALCRRAMCS   0x13      7.6

LDALLRRAM    0x14      TEST
LDVRAMONE    0x15      TEST

Loading fifos for 210 Validity Baselines
LDFIFOS      0x16      4.4     96.1
Loading fifos for 254 Main Baselines
LDFIFOS      0x16      9.3     81.9

UPSTATUS     0x17      Fast reading of three bytes.

INITONSTART 0x20       0.338
STARTMAINLP 0x21       This depends on the quantity of instructions loaded into the fifos.
                       The fir sequencer is what is then busy.
```

II) DETAILED DESCRIPTION OF EACH FUNCTION CODE

C functions for calling these routine are mentioned and are in VxWorks/hcbus/hcbnew/hcbFirCore.c All function codes below are in HEX:

1) NOP and MAIN MEMORY OPERATIONS, FUNCTION CODES 00, 01, 02 and 03:

```
       NOP:                    00 ZZ  (ZZ = REQUIRED DUMMY BYTE)
       WRITE MEMORY:           01 00 00 AA AA CC CC DD----DD
       READ MEMORY:            02 00 00 AA AA CC CC
       CALCULATE CHECKSUM:     03 00 00 AA AA CC CC (2 bytes returned, msbyte 1st)
```

WHERE
00 00 AA AA is a 32 bit start address and CC CC is a transfer count with MS byte first.

2) LOAD FIR ON BOARD RAMS AND FIFOS

```
*******************************************************************************
 Func 10 = Send Tap Weights (single tap weight index)
       DWNLDTAPS       10 TWI NT LI AA AA AA AA BB BB BB BB ---
```

The tap weight table in the microprocessor has room for sixteen sets of up to 64 tap weights. The tap weight index identifies which of the sixteen sets is referred to. It is assumed that the section referred to by the tap weight index, being loaded is not currently being used for observing. This table of tap weights will be stored in the uP RAM. The uP will select the correct tap weights for each tic. Tap weights can be sent for 8, 16, 32, and 64 tap filters. The tap weight index can be later switched to by referring to the tap weight index field in the command fifos (See function 16, LDFIFOS).

TWI = Tap Weight Index number being sent (0-15)
NT = Number of tap weights
- 0 - straight thru mode. If NT=0, unity tap weights are automatically sent to the fir, independent of the values sent by this function.
- 8 - 8 tap filter
- 16 - 16 tap filter
- 32 - 32 tap filter

```
              64 - 64 tap filter
              FF - DO NOT LOAD ANY TAP WEIGHTS.  COULD BE USED FOR UNUSED TAP WEIGHT INDEX
       LI = LTA integration time.  1,2, or 4 tics.  Not matter for straight thru mode
       AA AA AA AA = First of NT, 32 bit, IEEE Single Precision tap weight
              (ie. T0).  MSByte first.
 STORE THE TAP WEIGHTS IN 0-FFFH, AS 16 TAP WEIGHT INDEXS OF 64 TAPWEIGHTS.  16*64*4=4K
7100-710F      NT - NUMBER OF TAP WEIGHTS FOR EACH TAP WEIGHT INDEX
7110-711F      LI - NUMBER OF 131 MS PERIODS PER DUMP TIME FOR EACH TAP WEIGHT INDEX
7120-712F      ID - INDEX TELLING PLACE IN TAP WEIGHT CYCLE AS A FUNCTION OF TAP
WEIGHT INDEX.   ZEROED WHEN THE VALUES FOR THAT TAP WEIGHT INDEX  INITIALLY LOADED.
```

The tap weight sequencing is initially zeroed when the tap weights are loaded by HCB Function 10 DWNLDTAPS. There is the assumption that DWNLDTAPS will not be called for a tap weight index (TWI) currently used for observing. There is no double buffering. HCB Function 21 STARTMAINLP is called each tic. STARTMAINLP downloads the eight tap weights for each active TWI, then increments the counter which keeps track of the tap weight cycling for that TWI. Thus it is critical for the RTS to keep track of during which tic DWNLDTAPS is executed, to know where the FIR is in the tap weight cycle for that TWI. Since the two hcb functions DWNLDTAPS and STARTMAINLP cannot be called at the same time, there will be no conflict. The first call to STARTMAINLP after DWNLDTAPS will be the beginning of the tap weight cycle for that TWI.

```
Called by c function:
void firDwnldTap (twi,nt,li,buf,busnr,targetnr)
int twi; /* Tap Weight Index number 0-15 */
int nt; /* Number of tap weights, allowable values: 0 for straight thru mode
          8, 16, 32, or 64 */
int li; /* LTA Integration time. 1,2,or 4 tics. Not looked at in straight thru mode.*/
unsigned char *buf; /* the actual incoming tap weights */
int busnr, targetnr;

*******************************************************************************
Func 11 = Calculate Tap Weight Checksum (single tap weight index)
       CALCTWCHECKSUM         11 TWI (2 bytes returned, msbyte 1st)
       TWI = Tap Weight Index number (0-15)
```

The number of tap weights for that tap weight index is looked up in the uP RAM. The check sum is done on the data from the uP RAM. The TW ram is not checked, since only 8 TW's/TWI go into the TW RAM each tic. A 2 byte check sum of the tap weight bytes is calculated and returned. This does not have to be done during the 20 ms gap.

If this tap weight index has 0 tap weights for straight thru mode, 0 is returned as the checksum. However once filtering has begun, straight thru mode might return an erronious checksum, due to ff being stored as the number of taps. ff signals that the unity tap weights have already been downloaded and do not need to be redownloaded each tic. The checksum does not have much meaning for straight through mode anyway, so it is not advised to call this functions for a straight through mode tap weight index. If you do, it won't hurt anything as long as filtering has not begun. firTwCksumErr below will immediately return if the number of taps (nt) equals zero, indicating straight through mode.

Called by c function, which compares the uP supplied checksum to the checksum of the buffer data:

```
int firTwCksumErr (twi,nt,buf,busnr,targetnr)  /* return 1 if checksum error, else 0 */
int twi; /* Tap Weight Index number 0-15 */
int nt; /* Number of tap weights, allowable values: 0 for straight thru mode
          8, 16, 32, or 64 */
unsigned char *buf; /* the actual incoming tap weights */
int busnr, targetnr;

*******************************************************************************
 Func 12 = Load Result Ram (Single Section for a single backend result ram index)
       LDRESULTRAM            12 NN BERRI FC AA AA BB BB ---
 The result ram is loaded in sections during the 20 ms pause each tic.  The sections
 are stored in the uP RAM, to be transferred to the result ram during the 20
 ms pause in the next 131 ms interval.  This call just affects the uP RAM.  The
 total ram consists of 16 backend result ram index sections.
       NN = Section 0-7 of 256, 16 bit results.  8 * 256 = 2048 results.  Store in 7134.
       BERRI = Backend Result Ram Index is 0-15.  FF Means no data to download at the
              next tic. After downloading the data, substitute FF here.  Store in 7135.
              Note, each BERRI consists of 2048 independent results.
```

```
       FC = 0 load data now - Immediately transfer data into the result ram
            1 load data later -  wait until 20 ms gap in tic to download the data.
       AA AA = First of 256, 16 bit values, MSByte first (512 bytes).
              Store in 7200-73FF as 16 bit words, MSbyte 1st.
 Called by c function:
firLdResultRam (nn, berri, fc, buf, busnr, targetnr) in hcbFirCore.c
       int nn, berri, fc, busnr, targetnr;
       unsigned char *buf;

*******************************************************************************
 Func 13 = Calculate Result RAM Checksum
       CALCRRAMCS (FC=0) 13 NN BERRI FC (Two byte checksum CS CS returned immediately)
       CALCRRAMCS (FC=1) 13 NN BERRI FC CS CS
       NN = Section 0-7 of 256, 16 bit results. 8*256=2048 results.  Store in 7136
       BERRI = Backend Result Ram Index is 0-15.  Store in 7137.  After comparing
              the checksum, CALCRRCS stores FF here.
       FC = 0 means return a 2 byte check sum CS CS immediately
            1 means wait until 20 ms gap in tic to down load the data.

During the next 20 ms gap.  Read the Backend Result Ram Index BERRI, Section NN of
the result ram and calculate a 16 bit checksum of the 512 bytes in Section NN.
Compare that checksum with CS CS stored in 7139-713A.  For a checksum error, set
error bit 2 sent in function 21. CS CS = 16 bit checksum, msbyte first.  This is the
summation of both the ms & ls bytes.

Called by c function, which compares the uP supplied checksum to the checksum of the
buffer data:
int firCalcRRamCS (nn, berri, fc, buf, busnr, targetnr)
                /* return 1 if checksum error, for fc=0 */
       int     nn,berri,fc,busnr,targetnr;
       unsigned char  *buf;   /* 512 bytes of result ram data */

*******************************************************************************
 Func 14 = Load entire result ram.  ZZ = dummy value.
       LDALLRRAM              14 ZZ

This provides a default, result ram loading.  The result output is set equal to the
count address in. Bits 14-11 output are set equal to the backend result ram index
address input.  Since the value stored equals the address, the lsb of the BERRI (bit
11)= beBLMSB. The 2nd bit of the BERRI (bit 12) = LASTRES.  LASTRES indicates the
last result of section indicated by BERRI. LASTRES is used to sense an error
condition 1 sent by Func 20.

Called by c function:
void    firLdAllRRam(busnr, targetnr)
       int busnr, targetnr;

*******************************************************************************
 Func 15 = Load validity ram with IEEE Floating Point 1.0 in all locations.
       LDVRAMONE              15 ZZ     ZZ = dummy value
 This provides a default loading of the validity/tap weight ram.
Called by c function:
void    firLdVRamOne(busnr, targetnr)
       int busnr, targetnr;

*******************************************************************************
 Func 16 = Load Mod, LTABL, and DRAMBL Fifo's
       LDFIFOS (FC, bit3=1)          16 FC NN VD LA MA LB MB ---
       LDFIFOS (FC, bit3=0)          16 FC NN LA DA MA LB DB MB ---
       FC = Function Code.
              bit 0 - If this bit is a 1,
                       reset the fifos initially, plus provide an initial write.
              bit 1 - If this bit is a 1,
                      place an initial header of 16 fifo values for the tap weights.
                      This header is used in loading the correct values in the tap
                      weight and clac rams.
              bit 2 - If this bit is a 1,
                      add an extra baseline at the end of the fifo with stoploop = 1.
              bit 3 - If this bit is a 1, we are doing validities, so the format is:
                      LDFIFOS               16 FC NN VD LA MA LB MB ---
                      Note we do not load multiple DRAM fifo values, but a single
```

```
                value of VD.  If this bit is a 0, we are doing the main
                loop, so the format is:
                LDFIFOS              16 FC NN LA DA MA LB DB MB ---
          Typical usage might be FC=14 for processing the validities,
                the FC=4 for doing the main loop.
     NN = Number of baselines not including the extra one at the beginning or end
          for stoploop.  Each baseline consists of the 2 or 3 bytes Lx Mx or
          Lx Mx Dx defined as below.
          0 gives the max # of 256.  There can be multiple calls for >256.
     VD = Single validity DRAM baseline, typically 127 as a 2048 result baseline,
          loaded if bit3=1.
     Lx = 8 bit baseline to be sent to the LTA.
     Dx = 8 bit baseline sent to the DRAM modules.

          The 9th bit of Dx, BLMSB will be automatically set to 0 by the uP
          when MOD=7 OR 0, so no action is required by the RTS.
     Mx = STOPLOOP(MSB), TWI3, TWI2, TWI1, TWI0, MOD2, MOD1, MOD0
          STOPLOOP - Signals the end of the list of baselines for a given tic.
                It requires an extra pad baseline value at the end as
                signaled by FC bit 2.
          TWI[0..3] Tap Weight Index affiliation of that baseline.  This
                corresponds to a section of the tap weight table loaded by
                function 10, DWNLDTAPS.
          MOD[0..2] Modulus to tell number of results per baseline, defined:
                      MOD[0..2] |   *0    1   2    3    4    5    6    7
                # RESULT/BASELINE | 2048   32   64  128  256  512 1024 2048
                * MOD 0 is used only for loading the validity baseline, #210.

Called by the c function:
     firLdFifos (fc, nn, buf, busnr, targetnr) /* nn = 1 to 256 */
/* buf consists of VD LA MA LB MB ... if bit 3=1 or LA DA MA LB DB MB if bit 3=0 */
     int fc, nn, busnr, targetnr;
     unsigned char *buf;

*******************************************************************************
 Func 17 Read uP Status bytes
      UPSTATUS      17  (AA BB CC returned)
     AA = beBANK 0 or FF to indicate backend bank selected.
     BB = uPINT\0  0 when processing data.  1 when idling.
     CC = Reg 8001 contents where
          bit 0 = FIRENA from the LTA.  0 during the pause for not observing.
          bit 1 = uPSTPLP for determining if the main loop reached stoploop.
          bit 2 = beDONE requested backed count completed.
          bit 3 = uPbeLAST tells last result of baseline.
          bits 4-7 are spares
 Use uPINT\0 to know if FIR is reading LTA or in idle.
Called by c function:
firuPStatus(statusptr,busnr, targetnr)
int busnr, targetnr;
unsigned char *statusptr; /* points to 3 bytes returned */

*******************************************************************************
 Func 20       Initialization on Startup
     INITONSTART                  20 AA AA AA AA
 Reset the baseline and mod fifos, since in tic mode hcb16 will not do the reset.
Initially set NT=FF for all 16 Tap Weight Index locations.  This says no action will
be required for that TWI.
 7100-710F     NT -  0,8,16,32, OR 64.  NUMBER OF TAP WEIGHTS FOR EACH TAP WEIGHT INDEX
          FF CAUSES NO ACTION .
     HCB10   Downloads actual tapweights for use later.
 Put FF in 7135 and 7137 to say there is no result ram section there to store.
 Put FF in the error byte in 713B.
 Initially set beBANK backend bankswitch bit in 7138.
 Note the tic count ID was zeroed by hcb10, when the tap weights were downloaded.
 Load the divide chip control word.
 Load AA AA AA AA as the divide chip dividend.  AA AA AA AA is a IEEE floating point
 number, MSBYTE first.  8192 (AA AA AA AA = 46 00 00 00) would be a typical value.
Called by c function:
     firInitOnStart (dividend, busnr, targetnr)
     int busnr, targetnr;
     float dividend;
```

```
*******************************************************************************
 Func 21 Start main filtering loop
       STARTMAINLP                    21 FC (1 byte error code returned)
 Before any calls here:
       Call Func 20 at least once for initialization.
       Multiple calls to Func 16, LDFIFOS, to store the baselines, moduli, TWIs, and
       stoploops in the fifo's for normalized validity loop, non-normalized
       validity loop, and main loop filtering.
       Call Func 10 & 11 to download the tap weights for the current Tap Weight Indices.
       Call Func 12 & 13 to load the result ram for backend result mapping.
              Alternately Func 14 could give a default result ram loading.

 FC is the tic function code.
       bit 0 = 1 for non-tic, asynchronous test mode, 0 for normal tic mode.
       bit 1 = 1 for validities to be normalized having been loaded into the fifo.
       bit 2 = 1 for validities to not be normalized having been loaded into the fifo.
Note:  Both bits 1 & 2 being 0 is an error state.

If both bits 1 & 2 are 1, have the 'validities to be normalized' loaded first, by
calling HCB16 with a fc of 1110B.  Then load the validity baselines, which are the
'validities not to be normalized', by calling HCB16 with a fc of 1100B.  If there
are only normalized or non-normalized validities, have a single call to HCB16 with a
fc of 1110B.  The main loop baselines are then loaded by HCB16 with a fc of 0100B.

       bit 3 = 1 for the correlator using semaphores to synchronize tics.
       Otherwise use FIRENA to synchronize tics
              as in the test fixture.  See the flowchart of HCB21 following.
```

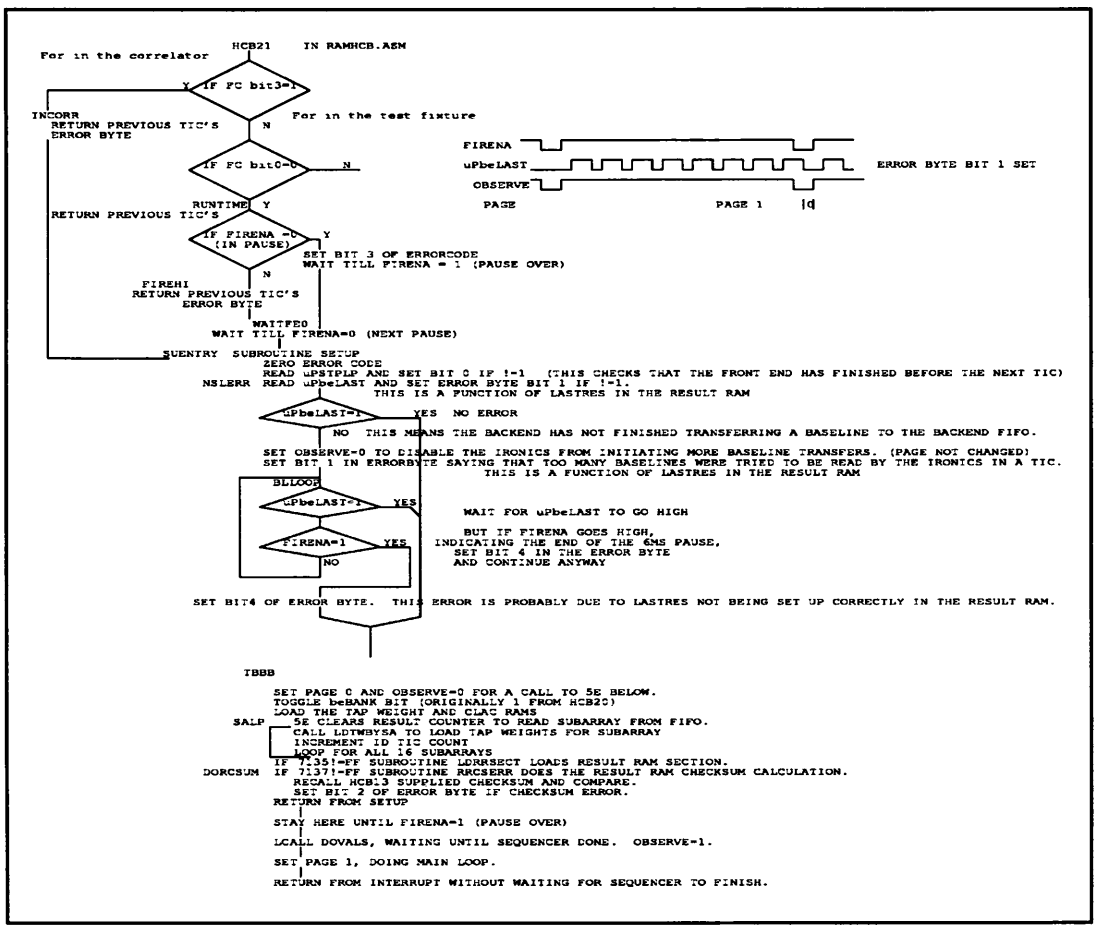


Figure 19 HCB21 Flowchart

When FC = 0: (Tic mode for use with the correlator)

Func 21 will be called once per tic, sometime before the 20 ms pause. The 20 ms pause may actually be more or less than 20 ms. The pause allows the LTA to do maintenance functions without being accessed. If Func 21 is called during the pause, bit 3 of the errorcode will be set, and the uP will wait for the next 20 ms pause. The 20 ms pause is defined by FIRENA from the LTA going low. Note, the microprocessor waits for FIRENA without returning from the interrupt. Thus, the hcb bus is not available for other functions to use. In a given tic, STARTMAINLP should be the last function called, since it will keep the bus until after FIRENA goes low.

First, return the stored error byte from the previous tic. The error byte is stored in uP memory 713B. The uP outputs OBSERVE, equal to FIRENA, going low in the 20 ms pause to inhibit backend activity. In the pause, the uP routine SETUP is called. When FIRENA goes high, have the sequencer do the validity loop or loops, then the main loop. The main loop returns from interrupt while the sequencer is still working. The RTS, while the sequencer is doing the main loop, can call HCB10, HCB11, HCB12 or HCB13, to download tap weights and result ram values to the uP RAM, and calculate checksums. Do not exceed the time remaining in the tic. Note the next call to an HCB function is not acknowledged until the previous call is done. This provides a way to keep the RTS synchronized with tics. That is the next tic's call to HCB21 cannot start until the previous tic's call is done. Since HCB21 waits for the pause, it synchronizes to the tics. The first call to Func 21 (after HCB20 did an initialization) returns the error byte of FF stored by HCB20. This first call stores a garbage errorcode from the nonexistent previous tic. The second call to Func 21 returns this garbage errorcode, and stores the error byte from the first tic. The third call returns the errorcode for the first tic.

On calls to HCB21 when FC = 1: (Non-tic asynchronous test mode)
Initiate the main filter loop based on no tics from the LTA. That is: Call SETUP.
Initiate the validity and main filter loops immediately, without any waiting for the 20 ms pause. Return the error byte, to indicate the processing is done.

The uP routine SETUP performs the following in this order:
Zero the stored error byte from the previous tic. The error byte is stored in 713B.

The bits in the error byte are as follows: (a set bit indicates an error)

bit 0 = The list of baselines did not finish processing by the end of the tic.
This is sensed by STOPLOOP from the DRAM fifo not occurring.
See Figure 13, The Sequencer Control PAL Description.
uPSTPLP is read at the beginning of SETUP. This bit is 1 if fc bit0=1, since STOPLOOP has not yet been set, since SETUP is called at the beginning.

bit 1 = The backend did not finish reading all the data by the end of the tic.
This is sensed when uPbeLAST is not set by LASTRES from the result ram. (see l031d16) uPbeLAST is read at the beginning of SETUP.
Specifically, the backend did not finishe writing the baseline into the backend fifo.

bit 2 = a checksum error generated when the checksum supplied from the RTS via HCB13, does not compare to that obtained by reading the section of the result ram specified by HCB13. The reading occurs in SETUP.

bit 3 = The call to HCB21 (FC bit0=0) occurred with FIRENA already low.
This bit will be returned immediately.

bit 4 = After the bit 1 error condition above occurred, where the backend did not finish writing a baseline into the backend fifo, the software waits for uPbeLAST to go high. uPbeLAST going high indicates the backend finished writing that baseline in the backend fifo. If FIRENA returns high, indicating the end of the LTA pause, and uPbeLAST still hasn't returned high, bit 4 is set. The writing of the baseline into the fifo should take less than the time of the FIRENA pause. The most likely cause of uPbeLAST not ever returning high, is that LASTRES was not set correctly in the result ram, for the size fft used.

The software will wait for FIRENA = 1, before proceeding.
Read uPSTPLP (bit1) and set error byte bit 0 if uPSTPLP != 1.
uPSTPLP signals the end of the main loop.
That means the next tic started before the baselines were all processed.
See Figure 13, The Sequencer Control PAL Description.
Read uPbeLAST(bit3) and set error byte bit 1 if uPbeLAST !=1.
That means the backend had not finished reading a baseline when the next tic started. See L031D16. Note beDONE could be read by HCB17 to see if the IRONICs supplied count has been met with.

Toggle the backend bank switch bit.

Load the tap weight and CLAC rams by calling LDTWBYSA then, increment the Tap Weight Index tic counts (ID). See the following flowchart of LDTWBYSA. HCB10 has previously downloaded all the tap weights for each active Tap Weight Index to the uP RAM. The uP routine LDTWBYSA downloads the 8 appropriate tap weights for each Tap Weight Index to the Tap Weight Ram. This happens once per tic. The tic count (ID) keys which tap weights to download each tic. The tap weight ram only holds 8 tap weights per Tap Weight Index at a time. Transfer the result ram section data previously stored in the uP ram by HCB12, to the result ram. Read the specified result ram section, calculate a checksum, compare to value stored from HCB13, set error bit 2 if error.

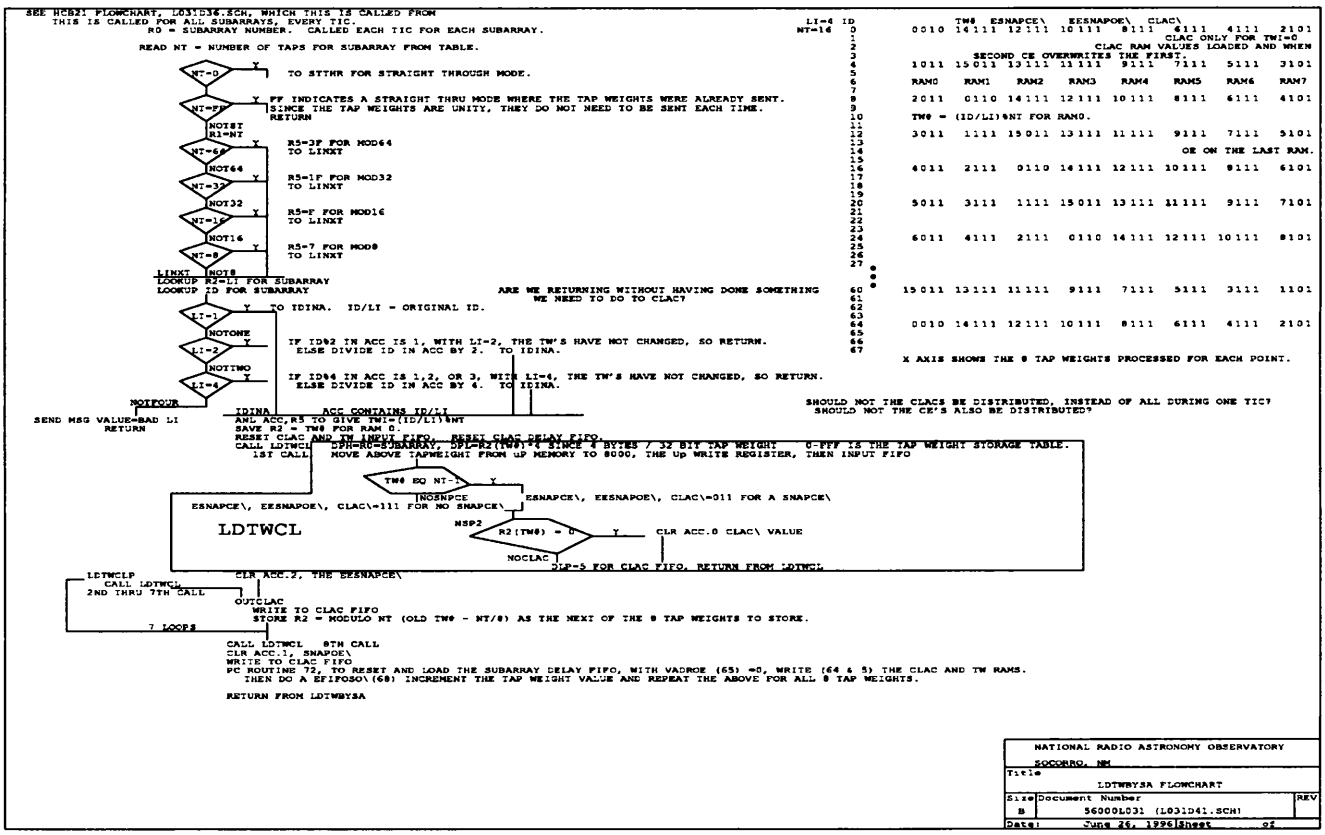


Figure 20 LDTWBYSA Flowchart

When FIRENA returns high, or if FIRENA is already high, do the following:
Using the baselines in the fifos, load the validity ram while filtering validities,
Initiate the main filter loop to process the baselines in the fifo until STOPLOOP is encountered.

Called by c function:

```
unsigned char firStartMainLp (fc, busnr, targetnr) /* returns errorcode */
        int     fc,busnr,targetnr;
```

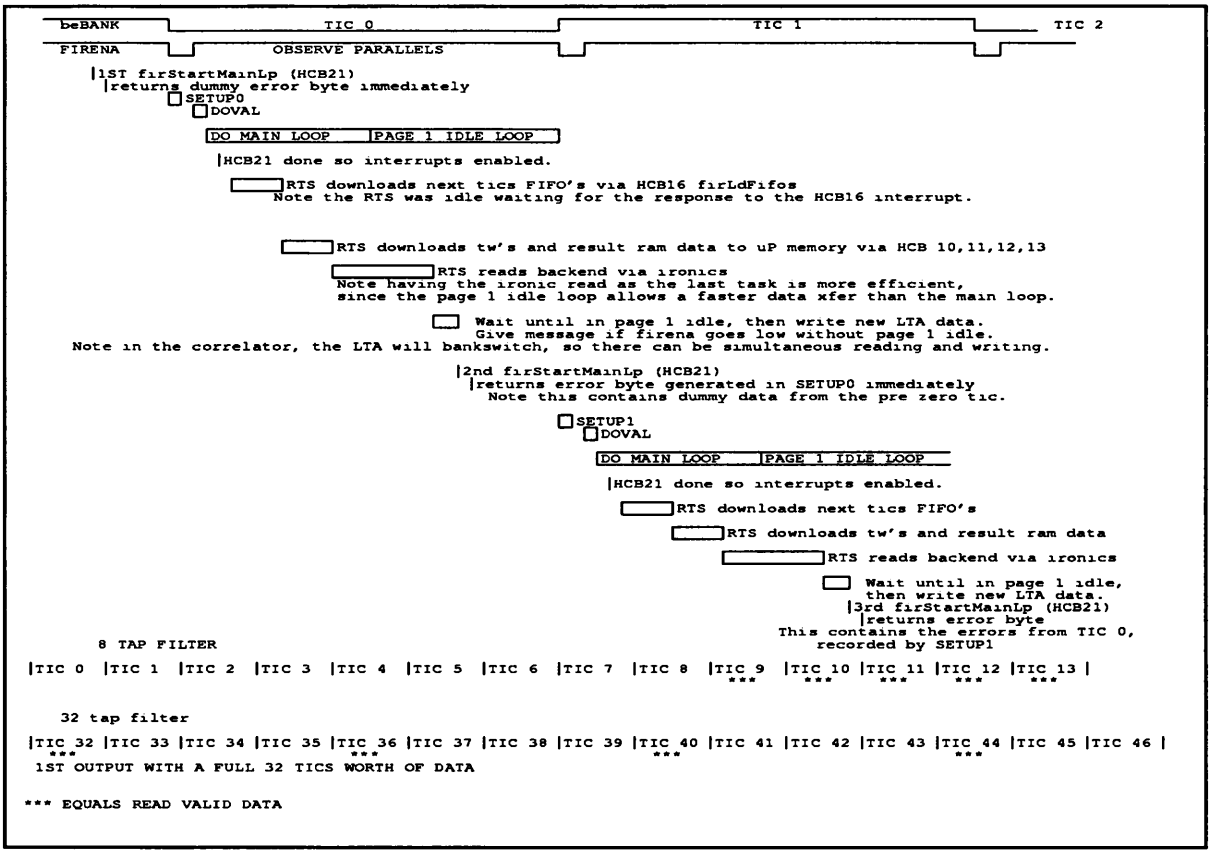


Figure 21 Tic Timing

The above figure shows the timing of the first two tics.

APPENDIX II FIR Memory Allocations

THE FOLLOWING MEMORY LOCATION ASSIGNMENTS ARE FOR 87C51 MICROPROCESSOR.

LOC	FUNCTION	
00	REGISTER 0, BANK 0	
01	REGISTER 1, BANK 0	
02	REGISTER 2, BANK 0	
03	REGISTER 3, BANK 0	
04	REGISTER 4, BANK 0	
05	REGISTER 5, BANK 0	
06	REGISTER 6, BANK 0	
07	REGISTER 7, BANK 0	
08	REGISTER 0, BANK 1	
09	REGISTER 1, BANK 1	
0A	REGISTER 2, BANK 1	
0B	REGISTER 3, BANK 1	
0C	REGISTER 4, BANK 1	
0D	REGISTER 5, BANK 1	
0E	REGISTER 6, BANK 1	
0F	REGISTER 7, BANK 1	
10	REGISTER 0, BANK 2	
11	REGISTER 1, BANK 2	
12	REGISTER 2, BANK 2	
13	REGISTER 3, BANK 2	
14	REGISTER 4, BANK 2	
15	REGISTER 5, BANK 2	
16	REGISTER 6, BANK 2	
17	REGISTER 7, BANK 2	
18	REGISTER 0, BANK 3	USED IN HCB INTERRUPT
19	REGISTER 1, BANK 3	USED IN HCB INTERRUPT
1A	REGISTER 2, BANK 3	USED IN HCB INTERRUPT
1B	REGISTER 3, BANK 3	USED IN HCB INTERRUPT
1C	REGISTER 4, BANK 3	USED IN HCB INTERRUPT
1D	REGISTER 5, BANK 3	USED IN HCB INTERRUPT
1E	REGISTER 6, BANK 3	USED IN HCB INTERRUPT
1F	REGISTER 7, BANK 3	USED IN HCB INTERRUPT
20	BIT ADDRESSABLE	(BITS 00 THRU 07)
21	BIT ADDRESSABLE	(BITS 08 THRU 0F)
22	BIT ADDRESSABLE	(BITS 10 THRU 17)
23	BIT ADDRESSABLE	(BITS 18 THRU 1F)
24	BIT ADDRESSABLE	(BITS 20 THRU 27)
25	BIT ADDRESSABLE	(BITS 28 THRU 2F)
26	BIT ADDRESSABLE	(BITS 30 THRU 37)
27	BIT ADDRESSABLE	(BITS 38 THRU 3F)
28	BIT ADDRESSABLE	(BITS 40 THRU 47)
29	BIT ADDRESSABLE	(BITS 48 THRU 4F)
2A	BIT ADDRESSABLE	(BITS 50 THRU 57)
2B	BIT ADDRESSABLE	(BITS 58 THRU 5F)
2C	BIT ADDRESSABLE	(BITS 60 THRU 67)
2D	BIT ADDRESSABLE	(BITS 68 THRU 6F)
2E	BIT ADDRESSABLE	(BITS 70 THRU 77)
2F	BIT ADDRESSABLE	(BITS 78 THRU 7F)
30	START OF STACK	
31		
32		
33		
34		
35		
36		
37		
38		
39		
3A		
3B		
3C		
3D		
3E		
3F		

LOC	FUNCTION
40	
41	
42	
43	
44	
45	
46	
47	
48	
49	
4A	
4B	
4C	
4D	
4E	
4F	
50	
51	
52	
53	
54	
55	
56	
57	
58	
59	
5A	
5B	
5C	
5D	
5E	
5F	
60	
61	
62	
63	
64	
65	
66	INT0 INTERRUPT VECTOR
67	INT0 INTERRUPT VECTOR
68	TIM0 INTERRUPT VECTOR
69	TIM0 INTERRUPT VECTOR
6A	INT1 INTERRUPT VECTOR
6B	INT1 INTERRUPT VECTOR
6C	TIM1 INTERRUPT VECTOR
6D	TIM1 INTERRUPT VECTOR
6E	UART INTERRUPT VECTOR
6F	UART INTERRUPT VECTOR
70	
71	
72	
73	TEMPORARY STORAGE, LDTWCL
74	TEMPORARY STORAGE, LDTWCL
75	TEMPORARY STORAGE, CALCRRCS ALSO IN LDTWCL
76	TEMPORARY STORAGE, TEST 2, CALCRRCS, LDTWCL
77	LAST TERMINAL KEY
78	MONITOR ADDRESS SHIFT IN
79	MONITOR ADDRESS SHIFT IN
7A	MONITOR ADDRESS SHIFT IN
7B	MONITOR ADDRESS SHIFT IN
7C	MONITOR SHIFT IN INDEX
7D	LAST MONITOR FUNCTION
7E	MONITOR TEMP STORAGE
7F	MONITOR TEMP STORAGE

THE FOLLOWING BIT LOCATION ASSIGNMENTS ARE FOR THE FIR 87C51 MICROPROCESSOR.

BIT	FUNCTION	
00	MONITOR FUNCTIONS	DISPLAY,PAGE, TEST FLAG 1
01	MONITOR FUNCTIONS	NIBBLE FLAG
02	MONITOR FUNCTIONS	DISPLAY,PAGE, TEST FLAG 2
03	MONITOR FUNCTIONS	DISPLAY/PAGE INTERNAL,EXTERNAL MEM FLAG
04	MONITOR FUNCTIONS	D ONLY AND P ONLY FLAG
05	MONITOR FUNCTIONS	MODIFY FLAG
06		
07	MONITOR FUNCTIONS	HELP FLAG
08	C/R ONLY FLAG	clear= send only c/r when c/r hit; set= continue last dsply
09		
0A		
0B		
0C		
0D		
0E		
0F		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
1A		
1B		
1C		
1D		
1E		
1F		
20		
21		
22		
23		
24		
25		
26		
27		
28		
29		
2A		
2B		
2C		
2D		
2E		
2F		
30		
31		
32		
33		
34		
35		
.		
.		
7F		

FIR MICROPROCESSOR RAM STORAGE ALLOCATION IN HEX

0-FFF TAP WEIGHT TABLE STORAGE
1000 3FFF PROGRAMS
7000-70FF MISCELANEOUS TEMP LOCATIONS
7100-710F NT - NUMBER OF TAP WEIGHTS FOR EACH SUBARRAY 0,8,16,32, OR 64. FF CAUSES NO ACTION .
7110-711F LI - NUMBER OF 131 MS PERIODS PER DUMP TIME FOR EACH SUBARRAY. 1,2, OR 4
7120-712F ID - INDEX TELLING PLACE IN TAP WEIGHT CYCLE BY SUBARAY. ZEROED WHEN SUBARRY INITIALLY LOADED.
INCREMENTED EACH TIC.
7130-7133 DIVIDEND FOR DIVIDE CHIP, IEEE 32 BIT #. MSBYTE 1ST.
7134-7135 NN AND SA FOR RESULT RAM DOWN LOADING
7136-7137 NN AND SA FOR RESULT RAM CHECKSUM CALCULATION
7138 LSB IS BANK SWITCH BIT beBANK
7139-713A CHECKSUM STORAGE FOR RESULT RAM CHECKSUM COMPARISON
713B HCB21 ERROR BYTE IMAGE
713C HCB21 FUNCTION CODE IMAGE
7200-73FF RESULT RAM TEMPORARY BUFFER

APPENDIX III TRW TMC3210 CMOS Floating Point Divider

CMOS Floating-Point Divider
32-Bit, 2.5MFLOPS

The TMC3210 is a CMOS monolithic device which is capable of performing a full 32-bit floating-point division in 400 nanoseconds. The floating-point device divides normalized numbers expressed in IEEE 32-bit single-precision format and can also accommodate denormalized operands if they are first "wrapped" by a companion TMC3033 arithmetic unit. The user can select either FAST mode (output zero) or IEEE mode (output a wrapped quotient) to handle underflows. With wrapping and unwrapping externally provided, the TMC3210 is fully compliant with the number format and single-precision division operation described in Version 10.0 of IEEE Standard 754. The TMC3210 is built using TRW's OMICRON-C™ one-micron CMOS process.

All data and instruction inputs are registered. The two input operands (divisor and dividend) are each loaded in two 16-bit words through the dedicated half-width bus and the output is produced in two 16-bit words through the dedicated output port. With a clock rate of 20MHz, the divider has a 2.5 Megaflop pipelined throughput rate with a latency on any given operation of 6 internal clock cycles (600ns). Renormalizing, rounding and limiting functions are all generated per IEEE specification. The output quotient and status flag ports are driven by three-state buffers.

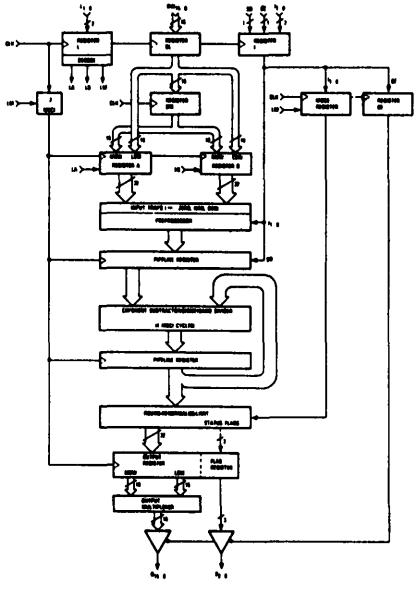
Features

- IEEE Standard 754 Version 10.0 32-Bit Floating-Point Data Format
- 20MHz Bus Clock Rate; 2.5 Megaflop Pipelined Throughput Rate
- IEEE Unbiased Round To Nearest, Round Toward Zero, Round Toward Positive Infinity And Round Toward Negative Infinity Modes
- Supports Denormalized Operands/Results Through "Wrapping/Unwrapping" By External TMC3033 Arithmetic Unit
- Two-Bus Architecture (Dedicated Input And Output) Works With Single Bus Or Data Flow Systems
- IEEE Exception Flags Including Inexact Result, Overflow, Underflow, Divide By Zero, Invalid Operation And Denormalized Operands
- Automatic Limiting For Overflow Or Underflow
- Input Traps For Infinity, Zero, Not-A-Number And Denormalized Operand
- All Inputs And Outputs Registered And TTL Compatible
- Low Power CMOS Construction
- Available In A 48 Lead DIP

Applications

- Graphics And Image Processors
- Solids Modeling
- Matrix Operations And Geometric Transforms
- Microcomputers/Minicomputers

Functional Block Diagram

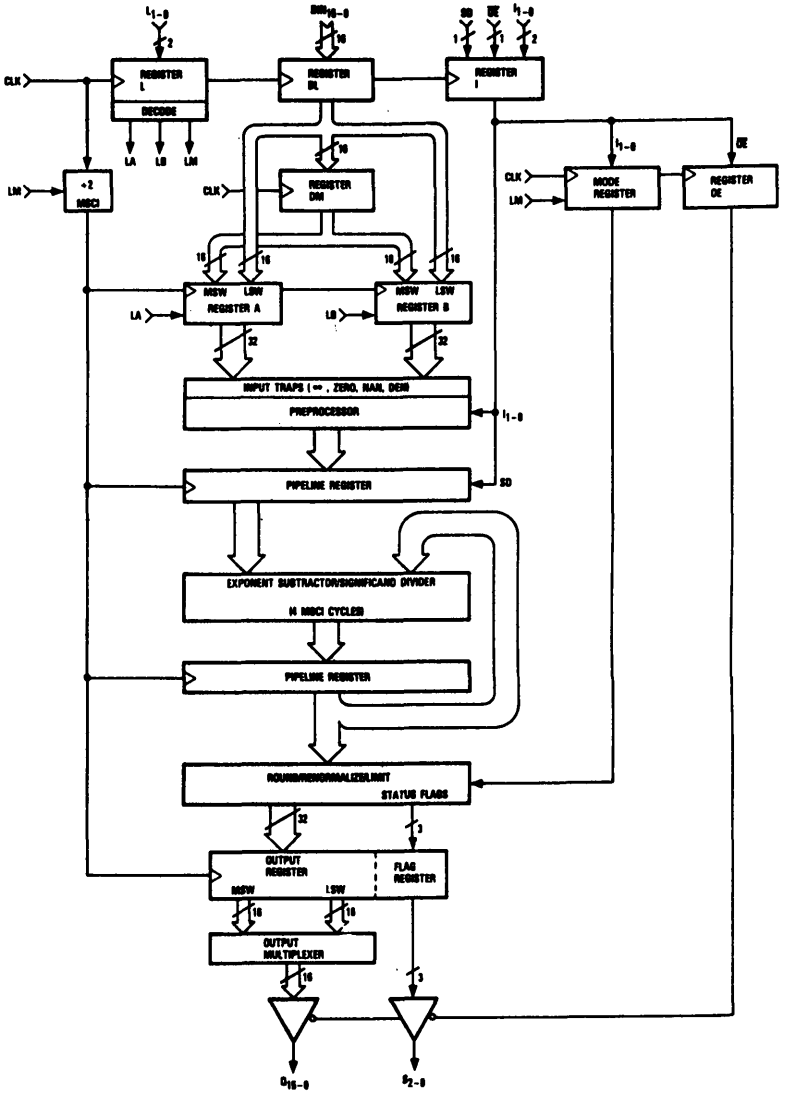


TRW LSI Products Inc.
P.O. Box 2472
La Jolla, CA 92038

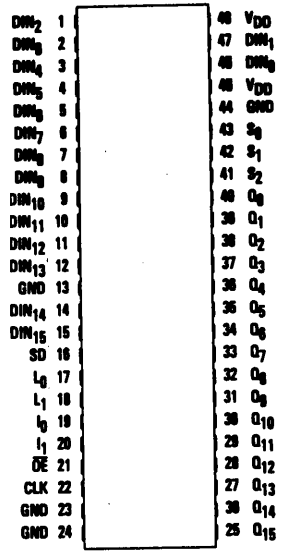
Phone: (619) 457-1000
Telex: 697-957
TWX: 910-335-1571

40G05346 Rev. A-4/88
Printed in the U.S.A.

Functional Block Diagram



Pin Assignments



48 Lead DIP - J4 Package

Functional Description

General Information

The TMC3210 consists of five sections: the input registers, the input preprocessor, the exponent subtractor/significand divider, the round/renormalize/limit block, and the output multiplexer, registers and drivers.

Input Registers

The input section accepts the data input (DIN) operand for the divisor (B) or dividend (A) along with an instruction which sets the mode (rounding) or format (wrapped or normalized number) depending on the load instructions. The external clock (CLK) strobes the DL and DM input preload registers, as well as the Load (L), Instruction (I) and Mode registers. CLK is internally divided by two to support an internal pipeline rate which is half the external bus clock rate.

The Most Significant Word (MSW) and the Least Significant Word (LSW) of both operands enter through the single 16-bit half-width input bus. The input preload register DL latches in the contents on the bus on the rising edge of CLK. The load instruction L_{1-0} enables the A, B or Mode register and must be input at the same time as the A operand (dividend), B operand (divisor) or selected rounding mode instruction I_{1-0} respectively. L_{1-0} must be held for two clock cycles while the MSW and then the LSW of the dividend or divider is loaded. The two operands may be loaded in either order, but each always enters on two consecutive rising edges of CLK with the MSW first. If either operand is not updated, the next division will use the respective value from the previous operation, facilitating repeated divisions by or into a constant.

Table 1. Load Instructions

L_{1-0}	Mnemonic	Operation
00	NOP	No loading of A, B, or Mode registers
01	LA	Load register A from DL and DM preload registers
10	LB	Load register B from DL and DM preload registers
11	LM	Load Mode register from I register

One of the four IEEE rounding modes is selected by I_{1-0} when the Mode register is enabled through the LM load instruction. During a Load Mode, the Start Divide (SD) control selects either FAST or IEEE mode for the handling of underflowing results.

Table 2. Mode Instructions

I_{1-0}[1]	Mnemonic	Operation
00	RN	Round to nearest number, or nearest even number if distances are equal (IEEE Standard 754 default)
01	RZ	Round toward zero (truncate product significand)
10	RP	Round toward positive infinity
11	RM	Round toward negative infinity

Note:
1. I_{1-0} selects the rounding mode during a Load Mode (LMI) instruction.

Table 3. Mode Control

SD[1]	Mnemonic	Operation
0	IEEE	Gradual Underflow (wrap exponent underflow values)
1	FAST	Flush-to-zero (replace exponent underflow numbers with zero)

Note:
1. SD selects IEEE or FAST mode during a Load Mode (LMI) instruction.

The registered Start Divide control initiates a division. SD must remain HIGH for two CLK cycles and may be asserted during the loading of the second operand. After SD is exercised, the user may load the next set of operands without interfering with the operation in progress. Another SD may occur every four internal MSCI clock cycles (eight external CLK cycles).

The format instructions I_{1-0} select the dividend and divisor format and must be input with the loading of the second operand. If only one operand needs to be loaded for a division, I_{1-0} is registered at the same time as the operand. Wrapped operands are too small to be expressed as standard IEEE normalized values, therefore instead of being denormalized with an exponent and hidden bit of 0, they are represented with a nonpositive two's complement exponent and a hidden bit of 1. A wrapped number is normalized, but has a special exponent. This special format allows the divider to handle denormalized numbers without large on-board normalizing shifters.

Table 4. Format Instructions

I_{1-0}	Mnemonic	Operation
00	A/B	Divide normalized A by normalized B
01	WA/B	Divide wrapped A by normalized B
10	A/WB	Divide normalized A by wrapped B
11	WA/WB	Divide wrapped A by wrapped B

Input Preprocessor

This section includes the input traps which detect infinity, zero, not-a-number and denormalized operand to generate the appropriate status flag.

Main Section (Exponent Subtractor/Significand Divider)

The difference of the exponents and the quotient of the significands is computed including the IEEE guard, round and sticky bits. This operation requires eight CLK cycles from the initial rising edge of SD. To avoid disruption, the next SD must not begin for eight CLK cycles. After the unrounded, unnormalized intermediate result leaves this section, the user may exercise SD to bring in the next set of operands from the input block.

Round/Renormalize/Limit Section

The significand of the quotient is rounded and readjusted so that the Most Significant Bit (MSB) occupies the nominal hidden bit position. If necessary, the exponent is adjusted to compensate for the renormalization shift. The final exponent is compared to the IEEE limits of 0 and 255 to generate the appropriate output condition and exception flag S_{2-0}.

Table 5. Status Outputs

S_{2-0}	Mnemonic	Exceptions
00X	OK	No exceptions
01X	UNF	Exponent underflow
10X	OVF	Exponent overflow or divide by zero
110	INV	Invalid operands or invalid operation
111	DIN	Denormalized operand

Table 6. Divider Exception Flags and Outputs

A Operand (Dividend)	B Operand (Divisor)				
	ZERO	DNRM	NRM/WNRM	INF	NaN
ZERO	INV, NaN	OK, ZERO	OK, ZERO	OK, ZERO	INV, NaN
DNRM	OVF, INF	INV[1], NaN	OK[1], ZERO	OK, ZERO	INV, NaN
NRM/WNRM	OVF, INF	OVF[1], INF	See Note 2	OK, ZERO	INV, NaN
INF	OK, INF	OK, INF	OK, INF	INV, NaN	INV, NaN
NaN	INV, NaN	INV, NaN	INV, NaN	INV, NaN	INV, NaN

Notes:

1. In IEEE mode, DIN (S_{2-0} = 111) is the status flag output.
2. In the case of NRM/WNRM divided by NRM/WNRM.
   OVF: Output is OVF, +NRM.MAX if (RM,RZ) and TRESULT > NRM.MAX.
   OVF, -NRM.MAX if (RP,RZ) and TRESULT < -NRM.MAX.
   OVF, +INF if (RN,RP) and TRESULT > NRM.MAX.
   OVF, -INF if (RN,RM) and TRESULT < -NRM.MAX.
   UNF: Output is zero with UNF if | TRESULT | < NRM.MIN (FAST mode).
   Output is WNRM with UNF if | TRESULT | < NRM.MIN (IEEE mode).
   ELSE: Output is OK with normalized value.
   NRM.MIN ≤ | TRESULT | ≤ NRM.MAX.
3. Terms used in this table include:
   OK = No exceptions raised.
   NRM = Normalized number.
   DNRM = Denormalized number.
   WNRM = Wrapped number.
   INF = Infinity (±, Exponent = FF_H, Significand = 000000_H).
   NaN = Not-A-Number (±, Exponent = FF_H, Significand = 600000_H).
   TRESULT = Normalized, rounded, true result before limiting.
   NRM.MAX = Maximum allowable positive normalized number ($(2-2^{-23}) \times 2^{+104}$ or Sign = 0, Exponent = FE_H, Significand = $7FFFFF_H$).
   NRM.MIN = Minimum allowable positive normalized number (2^{-126} or Sign = 0, Exponent = 01_H, Significand = 000000_H).

In FAST mode, all underflows are forced to zero and the underflow flag is generated. In IEEE mode, underflowing values are wrapped and the underflow flag is generated. Overflows are limited to the infinities for round toward nearest and to maximum magnitude normalized values for round toward zero. Round toward positive infinity limits the output to a positive infinity or a negative limit of maximum magnitude, negative normalized number. Round toward negative infinity limits the output to a negative infinity or a positive limit of maximum magnitude, positive normalized number.

Output Multiplexer, Registers and Drivers

The 32-bit output register and 3-bit flag register are clocked by MSCI. The quotient is output through the 16-bit output port via the output multiplexer which selects either the MSW or LSW. The synchronization of MSW or LSW with CLK is set by the LM load instruction. After the SD control is HIGH for two CLK cycles to begin a division, the MSW of the quotient is output after the 12th rising edge of CLK. The output will toggle MSW and LSW with CLK until the quotient from the next division is available. The state of the status flags will remain set until new exception conditions occur. The output drivers are enabled and disabled by the Output Enable ($\overline{OE}$) control.

Signal Definitions

Power

V_{DD}, GND — The TMC3210 operates from a single +5 Volt supply. All power and ground lines must be connected.

Data Inputs

DIN_{15-0} — DIN is the 16-bit input to the preload register DL which is loaded on the rising edge of CLK. All data operands (dividends and divisors) are loaded through the DIN port, MSW followed by the LSW.

Data Outputs

Q_{15-0} — Q is the 16-bit output from the output register which is clocked by MSCI. The output multiplexer is internally synchronized to select MSW then LSW of the quotient which is output through three-state output drivers.

Clock

CLK — The CLK frequency is twice the internal clock rate to allow for input/output data multiplexing. All operations are with respect to the rising edge CLK. The A and B input registers, pipeline registers and output registers are clocked by internal MSCI which is generated by dividing CLK by two.

Controls

I_{1-0} — The Mode/Format Instructions determine the rounding mode during a Load Mode, and select the input data formats when the operands are loaded. The rounding mode controls must be held for both CLK cycles during the loading of the Mode register. The format controls must be held for two CLK cycles during the loading of the last operand for a division.

L_{1-0} — The Load Instructions generate LA, LB and LM which enable the A, B and Mode input registers respectively. The load controls are read on every rising edge of CLK. All data transfers into these input registers take place on the rising edge of CLK following the load controls commanding the data transfer. L_{1-0} must be valid for two CLK cycles since the MSW and LSW must be loaded in two consecutive cycles. The LM instruction establishes the internal synchronization of CLK with MSCI and should not be asserted during a division.

$\overline{OE}$ — Output Enable is a registered control which enables the quotient and status outputs when LOW. When $\overline{OE}$ is HIGH, the outputs are in the high-impedance state. $\overline{OE}$ is read on the rising edge of CLK. The state of the output drivers will change after the next rising edge of CLK. Therefore, two CLK cycles are required to enable or disable the three-state drivers.

SD — Start Divide is an active HIGH control which begins the four MSCI clock cycle division. SD must remain HIGH for two CLK cycles and be asserted during or after the loading of the last operand of the divide. Subsequent SD may begin eight CLK cycles after the SD of the previous division. During the loading of the Mode register, SD selects whether FAST or IEEE mode is used in handling underflows.

Status Outputs

S_{2-0} — The status flags indicate the presence of exception conditions with the input operands or output quotient. The flags are valid while both the MSW and the LSW are output as long as the output buffer is enabled.

Package Interconnections

Signal Type	Signal Name	Function	J4 Package
Power	V_{DD}	Supply Voltage	45, 46
	GND	Ground	13, 23, 24, 44
Data Input	DIN_{15-0}	Input Data Word	15, 14, 12–1, 47, 46
Data Output	Q_{15-0}	Output Quotient Word	25–40
Clock	CLK	Clock	22
Controls	I_{1-0}	Mode/Format Instructions	20, 19
	L_{1-0}	Load Instructions	18, 17
	$\overline{OE}$	Output Enable	21
	SD	Start Divide	16
Flags	S_{2-0}	Status Outputs	41–43

Data Format

The TMC3210 conforms to IEEE Standard 754, Version 10.0 data format for 32-bit arithmetic. The divider requires two clock cycles to transfer a data word since the input and output buses are 16-bit wide.

Standard IEEE 32-Bit Floating-Point Format

The IEEE Standard 754, Version 10.0 specifies a 32-bit data format for floating-point arithmetic. In this format the MSB (bit 31) is the sign bit, the next eight bits (bits 30–23) are the exponent field and the 23 LSBs are the fractional significand field (bits 22–0). The "hidden bit" completes the 24-bit significand.

Sign Bit

The MSB carries the sign information. A HIGH for a sign bit indicates a negative number and a LOW indicates a positive number.

Exponent Field

The 8-bit exponent field determines whether the floating-point number is a signed infinity, a NaN, a zero, a denormalized number or a normalized floating-point number.

The exponent values 0 and 255 are special. If the exponent field is all ones (1111 1111, 255_{10}) and the fraction (bits 22–0) is zero, the number is evaluated as infinity x $(-1)^S$ with S being the sign bit. Any exponent of 255 with a nonzero fraction is a NaN. A NaN is generally used to communicate error information such as invalid operation or uninitialized memory and has no numerical value.

When the exponent field is all zeros (0000 0000) and the fraction is also zero, the number is a true floating-point zero. Note that this data format allows both positive and negative zeros which are computationally treated identically. When the exponent is zero and the fraction is nonzero, the number is a denormalized floating-point number evaluated as:

$$\text{Number} = (-1)^S \times 2^{E-126} \times (0.F)$$

where S is the sign bit, E is the value of the exponent field (base 10) and F is the value of the fractional field.

If the exponent field is neither all zeros nor all ones, the floating-point number is normalized and evaluated as:

$$\text{Number} = (-1)^S \times 2^{E-127} \times (1.F)$$

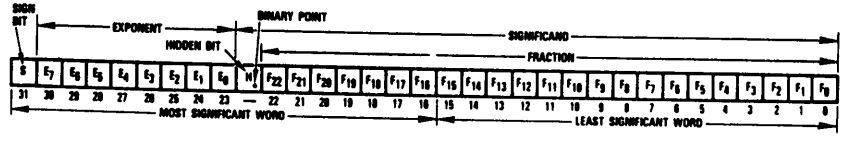
Note that the exponent bias has changed from 126 to 127 and that 1.0 has been added to the fractional field. The exponent can assume values which run from −126 to +127 (0 to 254 biased by 127). Note that both exponent fields of zero and one map onto the exponent value of −126. These provisions ensure a smooth transition from normalized numbers through gradual underflow into the denormalized numbers.

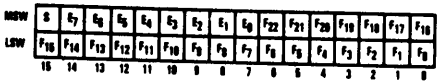
Fractional Field

Bits 22–0 comprise the fractional field (mentissa). There is a binary point assumed between bit 22 and the implied "hidden" bit 23. For a nonzero exponent, the hidden bit assumes a value of "1." For a zero exponent, the hidden bit has a value of "0." Bit 22 carries a binary weighting of 2^{-1}. The following bits carry decreasing binary weights down to the LSB (bit 0) which carries the weight of 2^{-23}. This is identical to treating the fractional part (bits 22–0) like an integer F multiplied by 2^{-23}. The fractional part of the floating-point number is either 0 + F (in the case of a zero exponent), or 1 + F (in the case of a nonzero exponent).

The difference between the smallest normalized number (exponent = 1, fractional part = 0) and the largest denormalized number (exponent = 0, fractional part = all ones) is one LSB. The smallest normalized number is: exponent = −126, significand = 1.00...00 written as exponent = 01_H, significand = 000000_H. The largest denormalized number is: exponent = −126, significand = 0.11...11 written as exponent = 00_H, significand = $7FFFFF_H$.

Figure 1. IEEE 32-Bit Floating-Point Format





Exponent	Fraction	Value	Name	Mnemonic
255	Not all zeros	--	Not-A-Number	NaN
255	All zeros	$(-1)^S \times \infty$	Signed Infinity	INF
1 through 254	Any	$(-1)^S \times (1.F) \times 2^{E-127}$	Normalized Number	NRM
0	Not all zeros	$(-1)^S \times (0.F) \times 2^{E-126}$	Denormalized Number	DNRM
0	All zeros	$(-1)^S \times 0.0$	Zero	ZERO

Note:

1. H, the hidden bit, is one except for zero and denormalized numbers when it is zero.

Figure 2. Timing Diagram

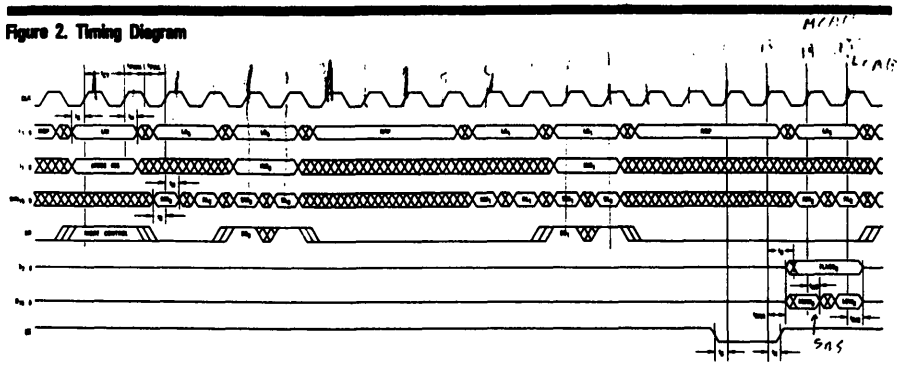


Figure 3. Equivalent Input Circuit

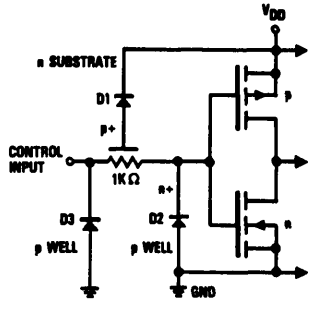


Figure 4. Equivalent Output Circuit

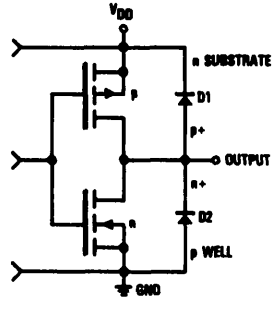
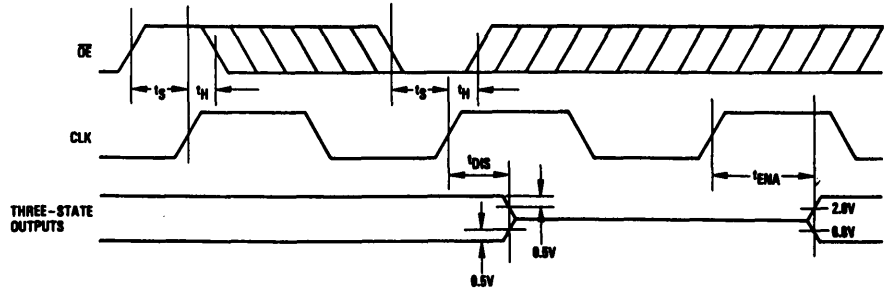


Figure 5. Threshold Levels For Three-State Measurement



Absolute maximum ratings (beyond which the device may be damaged)[1]

Supply Voltage	-0.5 to +7.0V
Input Voltage	-0.5 to (V_{DD} +0.5V
Output	
Applied voltage[2]	-0.5 to (V_{DD} +0.5V
Forced current[3,4]	-1.0 to +8.0mA
Short-circuit duration (single output in HIGH state to ground)	1 sec
Temperature	
Operating, case	-60 to +130°C
junction	175°C
Lead, soldering (10 seconds)	300°C
Storage	-65 to +150°C

Notes:

1. Absolute maximum ratings are limiting values applied individually while all other parameters are within specified operating conditions. Functional operation under any of these conditions is NOT implied.
2. Applied voltage must be current limited to specified range, and measured with respect to GND.
3. Forcing voltage must be limited to specified range.
4. Current is specified as conventional current flowing into the device.

Operating conditions

Parameter		Temperature Range						Units
		Standard			Extended			
		Min	Nom	Max	Min	Nom	Max	
V_{DD}	Supply Voltage	4.75	5.0	5.25	4.5	5.0	5.0	V
T_A	Ambient Temperature, Still Air	0		70				°C
T_C	Case Temperature				-55		125	°C

DC characteristics within specified operating conditions[1]

Parameter		Test Conditions	Temperature Range: Standard Min	Standard Max	Extended Min	Extended Max	Units
I_{DDQ}	Supply Current, Quiescent	V_{DD} = Max, V_{IN} = 0V		10		10	mA
I_{DDU}	Supply Current, Unloaded	V_{DD} = Max, $\overline{OE}$ = 0V					
		f = 20MHz		50		70	mA
		f = 10MHz		25		35	mA
I_{IL}	Input Current, Logic LOW	V_{DD} = Max, V_{IN} = 0V		-10		-40	µA
I_{IH}	Input Current, Logic HIGH	V_{DD} = Max, V_{IN} = V_{DD}		10		40	µA
V_{IL}	Input Voltage, Logic LOW			0.8		0.8	V
V_{IH}	Input Voltage, Logic HIGH		2.0		2.0		V
V_{OL}	Output Voltage, Logic LOW	V_{DD} = Min, I_{OL} = 4mA		0.4		0.4	V
V_{OH}	Output Voltage, Logic HIGH	V_{DD} = Min, I_{OH} = -2mA	2.4		2.4		V
I_{OZL}	Hi-Z Output Leakage Current, Output LOW	V_{DD} = Max, V_{IN} = 0V		-40		-40	µA
I_{OZH}	Hi-Z Output Leakage Current, Output HIGH	V_{DD} = Max, V_{IN} = V_{DD}		40		40	µA
I_{OS}	Short-Circuit Output Current	V_{DD} = Max, Output HIGH, one pin to ground, one second duration max.		-100		-120	mA
C_I	Input Capacitance	T_A = 25°C, f = 1MHz		10		10	pF
C_O	Output Capacitance	T_A = 25°C, f = 1MHz		10		10	pF

Note:

1. Actual test conditions may vary from those shown, but guarantee operation as specified.

AC characteristics within specified operating conditions

Parameter		Test Conditions	Temperature Range: Standard Min	Standard Max	Extended Min	Extended Max	Units
t_{CY}	Cycle Time	V_{DD} = Min		50		55	ns
t_{PWL}	Clock Pulse Width, LOW	V_{DD} = Min	30		35		ns
t_{PWH}	Clock Pulse Width, HIGH	V_{DD} = Min	15		15		ns
t_S	Input Setup Time		15		15		ns
t_H	Input Hold Time		0		3		ns
t_D	Output Delay	V_{DD} = Min, C_{LOAD} = 40pF		20		25	ns
t_{HO}	Output Hold Time	V_{DD} = Max, C_{LOAD} = 40pF	5		5		ns
t_{ENA}	Three-State Output Enable Delay[1]	V_{DD} = Min, C_{LOAD} = 40pF		20		25	ns
t_{DIS}	Three-State Output Disable Delay[1]	V_{DD} = Min, C_{LOAD} = 40pF		25		30	ns

Note

1. All transitions are measured at a 1.5V level except for t_{DIS} and t_{ENA}.

Ordering Information

Product Number	Temperature Range	Screening	Package	Package Marking
TMC3210JAC	STD = 0°C to 70°C	Commercial	48 Lead Ceramic DIP	3210JAC
TMC3210JAG	STD = 0°C to 70°C	Commercial With Burn-In	48 Lead Ceramic DIP	3210JAG
TMC3210JAV	EXT = -55°C to 125°C	MIL-STD-883	48 Lead Ceramic DIP	3210JAV

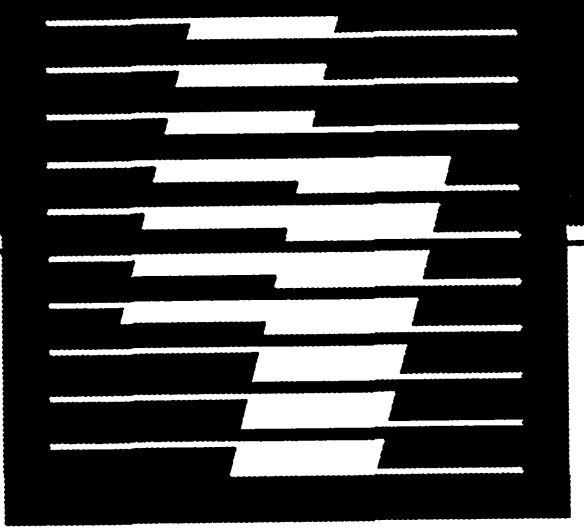
All parameters in this specification are guaranteed by design, characterization, sample testing or 100% testing as appropriate. TRW reserves the right to change products and specifications without notice. This information does not convey any license under patent rights of TRW Inc. or others.

APPENDIX IV Ironics IV-3272-PIO Parallel Interface Daughter Board

IRONICS
Incorporated

IV-3272-PIO
VMEbus Parallel I/O
Daughter Board

USER'S MANUAL



Board Etch:	1.10
Manual Version:	1.400
Manual Part Number:	700061
Release Date:	Dec-31-91

Revision History

Manual	IV-3272-PIO 1.200	15-Jul-87
Manual	IV-3272-PIO 1.300	29-Apr-88
Manual	IV-3272-PIO 1.400	31-Dec-91

Ironics' Quality Assurance, Publications, and Technical Support Groups exercise a multi-staged program in an effort to achieve the highest quality possible in all our products and accompanying documentation. All board products are burned in for a minimum of 48 hours (RAM for 96 hours) at 50 degrees Celsius. All system-level products are run for a minimum of 48 hours in addition to the board-level burn-in. All board, software, and system documentation is reviewed and approved prior to printing.

Please read and abide by the following paragraphs. Questions and comments should be directed to your Ironics sales representative, or to Ironics directly at the address below.

Ironics, Inc. does not recommend the use of its components in life support applications where failure or malfunction of the component may result in injury or death. In accordance with Ironics, Inc.'s terms and conditions of sale, the user of Ironics, Inc. components in any and all life support applications assumes all risks arising out of such use and further agrees to indemnify and hold Ironics, Inc. harmless against any and all claims of whatsoever kind or nature (including claims of culpable conduct [strict liability, negligence or breach of warranty] on the part of Ironics, Inc.) for all costs of defending any such claims.

Ironics, Inc. makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Ironics, Inc. assumes no responsibility for any errors that may appear in this document. The information in this document is subject to change without notice.

IV-3272 Full Speed Data Transporter is a trademark of Ironics, Inc.

Ironics Incorporated
Technical Support Services
798 Cascadilla Street
Ithaca, New York 14850

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 General Description

The IV-3272-PIO Parallel Interface Daughter Board provides a 32-bit-wide data connection between the VMEbus P2 user I/O interface and the IV-3272 Full Speed Data Transporter (FSDT). The interface is general enough to allow a wide variety of devices to be connected with little or no modification. Data transfers can occur at a rate of up to 10 million transfers per second.

The IV-3272-PIO daughter board provides the following features:

- Attaches to IV-3272 Full Speed Data Transporter
- Provides I/O capability at up to 40 Mbytes per second via P2 interface
- Provides 8/16/32-bit I/O parallel data transfer mechanism
- IV-3272 Digital Signal Processor (DSP) port connects to data lines for programmed I/O
- Data multiplexing between I/O and VMEbus implemented in hardware
- Easy to use REQ*, ACK*, and ENDIR data transfer handshake
- Optional status/control signals

The interface is controlled by circuitry on the IV-3272-PIO and firmware resident on the IV-3272. The standard IV-3272 firmware implements a specific subset of the possible I/O configurations. A user has the option of writing TMS320 DSP code for execution on the IV-3272 that operates the IV-3272-PIO interface in a different manner.

The IV-3272-PIO provides a parallel data interface and three handshake signals. The parallel data interface is composed of the I/O data lines which are used for 8, 16, or 32-bit transfers. The IV-3272 can be programmed to supply a 16 or 32-bit VMEbus data path. The daughter board interface (DBI) port on the IV-3272 connects directly to the data line latches (IOBUS) to facilitate programmed I/O.

The three handshake signals generated are request (REQ*), acknowledge (ACK*), and direction (ENDIR). REQ* is always generated by the source, and indicates that data is ready to be received. ACK* is generated by the sink, or destination, and indicates that data has been received. REQ* and ACK* are driven when the transfer actually begins. ENDIR indicates the direction of the data transfer. ENDIR in the low state indicates a transfer from I/O to VME; ENDIR goes high if the transfer is from VME to I/O.

Data multiplexing is provided in the circuitry of the IV-3272-PIO. This feature allows 16 or 32-bit data to be transferred from the VMEbus to the P2 I/O as either 8, 16, or 32-bit data. Conversely, 8, 16, or 32-bit data from the P2 I/O can be transferred to the VMEbus as 16 or 32-bit data. The data sizes of the source and destination are specified in the IV-3272 transfer parameter block (TPB) and are implemented by the standard firmware.

The IV-3272-PIO drives eight control lines and receives eight status lines. The status and control lines are designed to facilitate communication of control information separate from data information. The control lines are driven by either:

- a write to the IV-3272 control register from the VMEbus

- through the DSP by writing to Data Memory at address C00

The status lines are read by either:

- reading the IV-3272 status register over the VMEbus
- through the DSP by reading Data Memory at address C00

Connection of two IV-3272-PIOs status and control lines must be done in the connecting cable. The IV-3272 standard firmware does not utilize the status or control lines.

Table 1-1 shows the IV-3272-PIO daughter board interface signals. Table A-1 and Table A-2 list the IV-3272-PIO signal names associated with the VMEbus P2 connector.

Table 1-1 Daughter Board Interface Signal List

⇒ Out, to interface (daughter board)
⇐ In, to IV-3272 FIFO (VMEbus)
⇔ Bidirectional

Name	Direction	Function
		DMA Data Transfer Signals
DBACK*	⇒	acknowledge transfer between FIFO and DB latches
DBSTB*	⇐	request data transfer between FIFO and DB latches
DBSTB2*	⇐	request data transfer between FIFO and DB latches
DIR	⇒	transfer direction: 1 is VME to P2; 0 is P2 to VME
I/OSTART*	⇒	control signal enabling data transfer (reflects flag status)
IBC2-0	⇒	I/O cycle burst count
IBLT	⇒	I/O block transfer enable
IOBUS31-00	⇔	data lines 31 through 00
ISPARE2	⇐	I/O data port size (from TPB)
ISPARE3	⇐	I/O data port size (from TPB)
VLWORD*	⇒	VMEbus data port size (from TPB)
		Status Line (VMEbus Master) Signals
CRWR*	⇒	control write signal
SRST*	⇒	system reset (VMEbus)
STRD*	⇒	status read signal
SYSCLK*	⇒	VMEbus system clock
VBUS15-8	⇔	VMEbus data lines (15 through 00) - IV-3272 control/status port
		Onboard 32025 DSP Signals
DAUI/O*	⇒	I/O chip select from 32025 to read IOBUS
DAUMEM*	⇒	memory chip select
DCLKIN	⇔	DSP clock in
DCLKOUT	⇒	DSP clock out
DSPA11-0	⇒	address lines
DSPD15-0	⇔	data lines
INT1	⇐	interrupt
IOT0	⇒	toggle successive I/O port accesses between D31-D16 and D15-D00
IS*	⇒	I/O strobe - DEN signal (data enable)
PS*	⇒	program strobe - MEN signal (program memory)
R/W*	⇒	read/write

2 Theory of Operation

The IV-3272-PIO is operated by a combination of the circuitry on the IV-3272 and the control of the DSP software. A user has the option of using either the Ironics standard IV-3272 firmware or of generating DSP software. The following sections discuss each of these options. It is recommended that the user gain a thorough understanding of the information outlined in chapters 7 and 8 of the *IV-3272 VMEbus Full Speed Data Transporter User's Manual.*

2.1 Standard Firmware

Ironics supplies standard firmware in EPROMs that reside on the IV-3272 and provide instructions for the DSP chip. All of the functions of the standard firmware are detailed in the *IV-3272 VMEbus Full Speed Data Transporter User's Manual*. Those features of the firmware which operate the IV-3272-PIO circuitry are discussed in this section.

The operation of the IV-3272 is controlled by software which executes on a CPU board resident on the VMEbus. The controlling software loads a Transfer Parameter Block (TPB) into an idle IV-3272 FIFO and asserts the Start bit in the start/stop register. The firmware moves the TPB from the FIFO to the DSP data RAM, interprets it, and performs the specified transfer.

A TPB contains two attribute words describing the source and destination for the data transfer. These attribute words implicitly indicate the transfer direction. Bits D00 and D01 of the attribute word specify whether the transfer is to use the VMEbus, the DSP onboard RAM, or the IV-3272-PIO (see Table 2-1).

Table 2-1 Attribute Word Location Setting

D01	D00	Mode
0	0	Onboard DRAM
0	1	I/O Type 1
1	0	I/O Type 2
1	1	VMEbus

I/O transfer types indicate IV-3272-PIO transfers. I/O Type 1 directs the DSP to perform transfers between the IV-3272-PIO and the VMEbus in a concurrent and asynchronous fashion. That is, the FIFO is filled and emptied on both the VME and I/O side at the same time. I/O Type 2 directs the DSP to jump to a user program that has been previously loaded into DSP RAM. The user DSP program must perform all data transfer control. This feature is provided to allow data to be collected and processed by a user DSP program as it passes through the board. I/O Type 2 transfers result in an error if the user has not loaded DSP code into the onboard RAM.

2.2 Programmers Model

The programmer accesses the IV-3272-PIO interface by referencing one of four possible access points; two access points exist for the IV-3272 DSP and two exist for the VMEbus. DSP I/O port 6 is designated as the daughter board interface port. The status and control signals are accessed by reading and writing the program memory map, respectively. The status and control signals are also accessed by reading and writing the IV-3272 VMEbus status and control register, respectively. Subsection 2.2.1 through Subsection 2.2.3 discuss each programming reference and the IV-3272 and IV-3272-PIO responses.

2.2.1 Daughter Board Interface Port

A DSP port (PA6) is designated as the daughter board interface (DBI). The DBI allows the software to read or write the 32 I/O data signals on the IV-3272-PIO. The DBI is a 16-bit port which uses the I/O toggle function to access the upper 16-bit word (IOBUS16-31) on the first read and the lower 16-bit word (IOBUS0-15) on the next read. The word alternates with each access. Writing the status latch (STL) with the DSP software resets the toggle to the upper word.

Writing the DBI port causes the data written to be latched by the IV-3272-PIO and asserted on the I/O data lines. This mode operates only if the IV-3272 is not performing a DMA operation to or from the IV-3272-PIO. Reading the DBI turns off all output latches set by the previous write and provides the data that is present on the I/O data lines.

Ironics supplies an optional control PLA which provides a different DBI interface mechanism. This mechanism is intended to provide a dedicated 16-bit output port and 16-bit input port to facilitate users interfacing to D/A controllers, A/D controllers, or other devices where programmed I/O is desirable.

Using this optional PLA, the upper word (IOBUS16-31) is a dedicated output port. Data written to that port remains latched and asserted. The data may be changed by subsequent writes. Subsequent reads to that port do not disable the latch and the data does not represent the incoming data I/O lines. This mechanism stops operating as soon as the IV-3272 begins a DMA transfer involving the IV-3272-PIO.

The lower word is accessed in the same manner as the standard PLA implementation. Data written is latched and remains asserted; a subsequent read turns off only the lower output latch so the data on the I/O lines is read. The upper word latch status does not change.

2.2.2 Status Information

The IV-3272 decodes DSP program memory addresses from C00 to FFF as daughter board memory. Accessing that address range in the DSP software asserts the DAUMEM* and the R/W* signals to the IV-3272-PIO. When the IV-3272-PIO senses DAUMEM*, and R/W* indicates a read, the status byte data is presented on bits D08 - D15. These signals are connected to P2 C32 - C25 (see section 2.4). Also, during this read access, bit D00 indicates the state of the LBG* signal, D01 indicates the state of the ACK* signal, and D03 indicates the state of the LREQ* signal; for each signal, 0 means asserted, 1 means not asserted.

This status port can also be read from the VMEbus via D08 - D15 of the IV-3272 status register.

2.2.3 Control Information

There are two control byte latches, one written to by the DSP and one written to via the IV-3272 VMEbus control register. Either one can be strapped to connect to the P2 lines (P2 C24 - C17) on an

individual basis (see section 2.4). Writing via the DSP, DAUMEM* is asserted and the R/W* signal indicates a write. The data is presented on DSP data lines D08 - D15 and is captured in the DSP control byte latch.

2.3 Data Interface

2.3.1 I/O Data Size Control

The parallel data interface between the IV-3272-PIO and the P2 connector is implemented using four F543 octal latching transceivers. Data width is specified using IV-3272 signals VLWORD*, ISPARE2, and ISPARE3. VLWORD* reflects the size of the VMEbus data; 0 (asserted) indicates a 32-bit VMEbus transfer, 1 indicates a 16-bit VMEbus transfer. VLWORD* is set by the DSP software in bit D03 of the AWL port (attribute word latch). The ISPARE1 and ISPARE2 signals indicate the I/O transfer size, as described in Table 2-2. These signals are set by the DSP software in bits D03 and D02 in the STL port (status latch) and are interpreted by PLA 5A2.

Table 2-2 I/O Data Size Control

ISPARE2	ISPARE3	I/O Size
0	0	--
0	1	8
1	0	16
1	1	32

NOTE:

The transfer of one 32-bit datum will not operate. The 32-bit datum may be multiplexed to 16-bit or 8-bit I/O and transferred, but a single 32-bit transfer is illegal.

2.3.2 I/O Data Multiplexing

Data size multiplexing is performed using FCT245 bidirectional transceivers. PLAs 5A0 and 5A1 provide size control signals. PLA 5A2 clocks transfers and controls the strobe generation. Because of the time required to capture and multiplex data, the use of this feature prevents the maximum transfer rate of 10 MHz from being achieved.

Legal, bidirectional multiplexing modes include:

- I/O 8-bit to VMEbus 16-bit
- I/O 8-bit to VMEbus 32-bit
- I/O 16-bit to VMEbus 32-bit

One multiplexing mode is illegal. The IV-3272 and the IV-3272-PIO cannot perform I/O 32-bit to VMEbus 16-bit multiplexing. Transfers employing this multiplexing parameter will not function properly.

2.3.3 I/O Data Handshaking

Handshake is provided by three signals: ENDIR, REQ*, and ACK*. REQ* and ACK* are bidirectional, open-collector signals. The device supplying data (the source) drives REQ*, and the device receiving data (the sink) drives ACK*. ENDIR signals the direction of the data transfer. ENDIR in the low state indicates a transfer from I/O to VME; ENDIR goes high if the transfer is from VME to I/O. In the idle state, ENDIR is 0 and the data buffers are disabled (see Figure 2-1).

To initiate a transfer, the IV-3272 is programmed via a TPB. To send data to an I/O device, the source attribute word (SRCA) should indicate VMEbus (D01, D00 = 1, 1), and the sink attribute word (SNKA) should indicate I/O type 1 (D01, D00 = 0, 1); see Table 2-1 outlining attribute word location settings.

When the start bit in the VMEbus start/stop register is asserted, the IV-3272 reads the TPB from the FIFO and begins the transfer. The IV-3272 requests the VMEbus and begins filling the onboard FIFO with data. When the FIFO becomes half full (256 transfers), the IOSTART* signal requests use of the daughter board data transfer bus (IOBUS00 - IOBUS31). When the IOBUS is granted (LBG* = 0), ENDIR goes high (1) indicating a VMEbus to I/O transfer. The first data are read from the FIFO using the FIFO handshake signals DBSTB*, DBSTB2* and DBACK*; the data is latched by the IV-3272-PIO. The IV-3272-PIO asserts the REQ* signal indicating data is valid. The destination device asserts ACK* after it has received the data. The receipt of ACK* by the IV-3272-PIO causes REQ* to be released. The FIFO handshake loads the next data and the process continues until the FIFO is empty as signaled by the release of IOSTART*. The rate at which ACK* is returned by the destination device paces the transfer.

If the total transfer length is less than 256 transfers, IOSTART* is asserted after the VMEbus has been released by the IV-3272. If the transfer length is greater than 256, the FIFO behaves elastically, with transfers occurring simultaneously on both sides of the FIFO. If the IV-3272-PIO side is considerably faster than the VME side and the FIFO empties, IOSTART* goes inactive (1) until the FIFO reaches half full or the transfer ends. If the IV-3272-PIO side is considerably slower than the VMEbus and the FIFO fills, the IV-3272 VMEbus request is withheld until the FIFO is emptied to half.

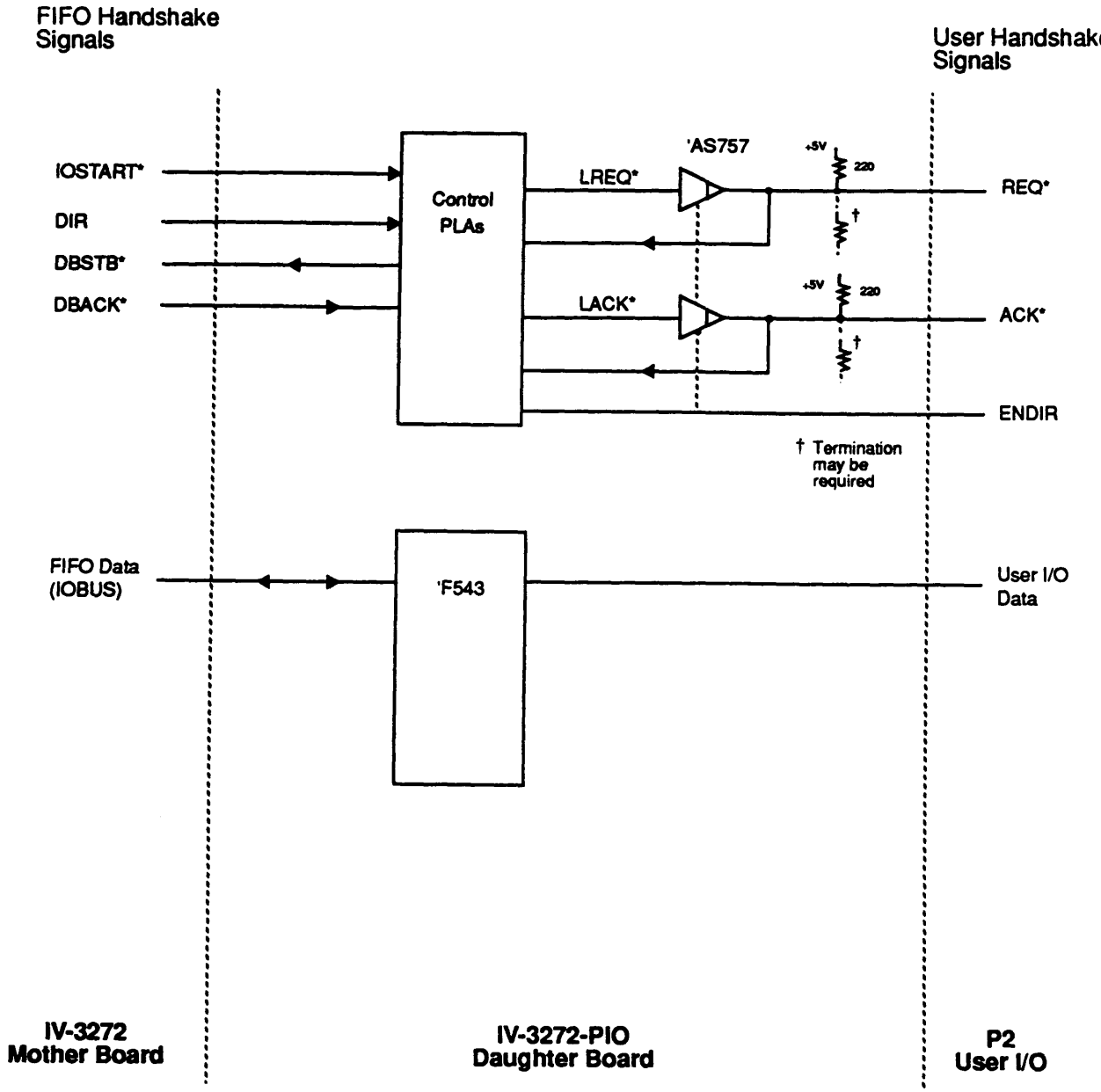
To receive data from an I/O device, the source and sink attribute words are reversed. When the start bit in the VMEbus start/stop register is asserted, the IV-3272 reads the TPB from the FIFO and begins the transfer. The IV-3272 asserts IOSTART* (0), and the IV-3272-PIO waits for the source to assert REQ*. When REQ* is received by the IV-3272-PIO, the data signals information is latched and the IV-3272-PIO asserts ACK*. The FIFO handshake signals are asserted by the IV-3272-PIO to transfer the data from the IV-3272-PIO latches to the FIFO. This process is the reverse of sending data out. After the source releases REQ*, the IV-3272-PIO releases ACK*.

The count programmed in the TPB is the number of VME transfers. Note that when inputting data to the IV-3272, the user must keep count if this information is important to the application. If the actual amount of data coming into the interface is greater than the transfer count programmed in the TPB, the difference is lost (but the expected data is transferred to the destination). If the actual amount of data coming into the interface is less than the programmed transfer count, the IV-3272-PIO must interrupt the DSP chip to indicate that it is finished with the transfer; otherwise the IV-3272 will stay busy, waiting for more data.

Figure 2-2 shows logic analyzer traces of two IV-3272-PIO connected together transferring data between two VME systems.

Figure 2-3 shows the timing requirements of data with respect to the handshake signals.

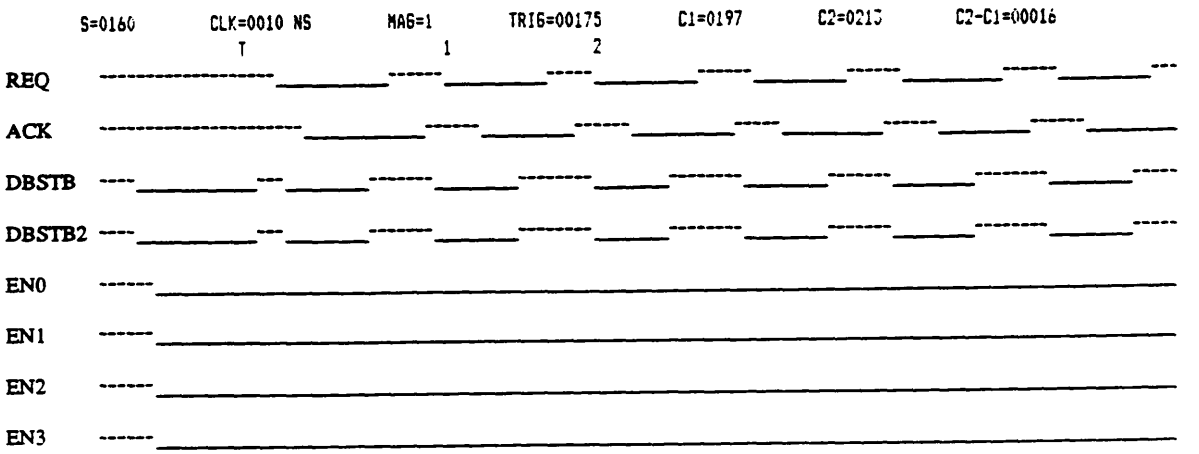
Figure 2-1 Conceptual Block Diagram - Point to Point Connection



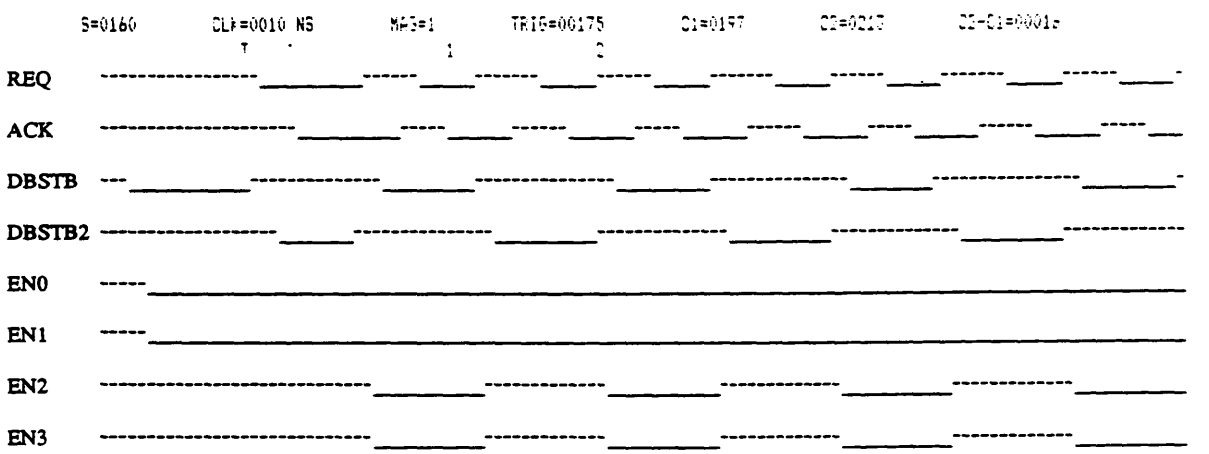
	REQ*	ACK*	ENDIR
DATA OUT:	driven	received	1
DATA IN:	received	driven	0

Idle state is DATA IN

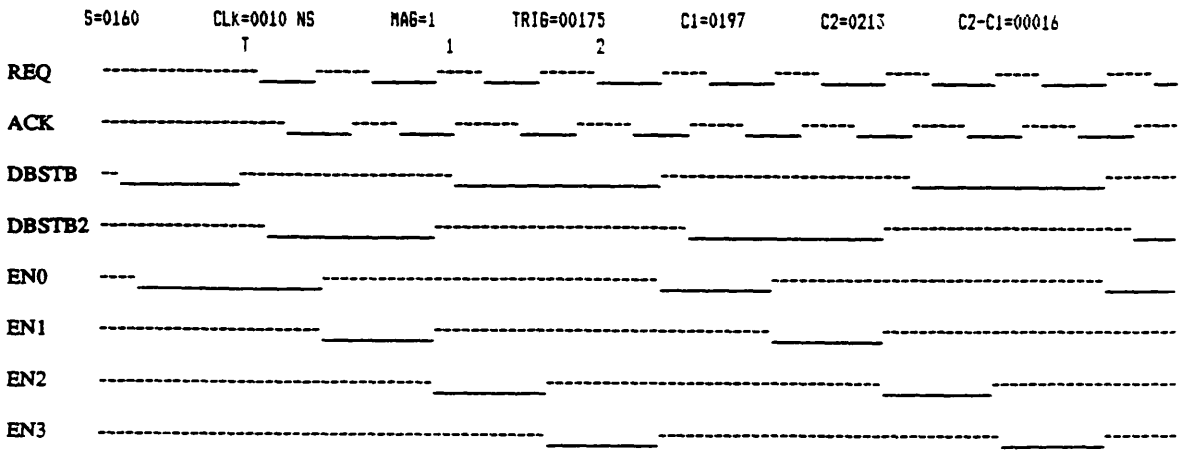
Figure 2-2 Logic Analyzer Traces



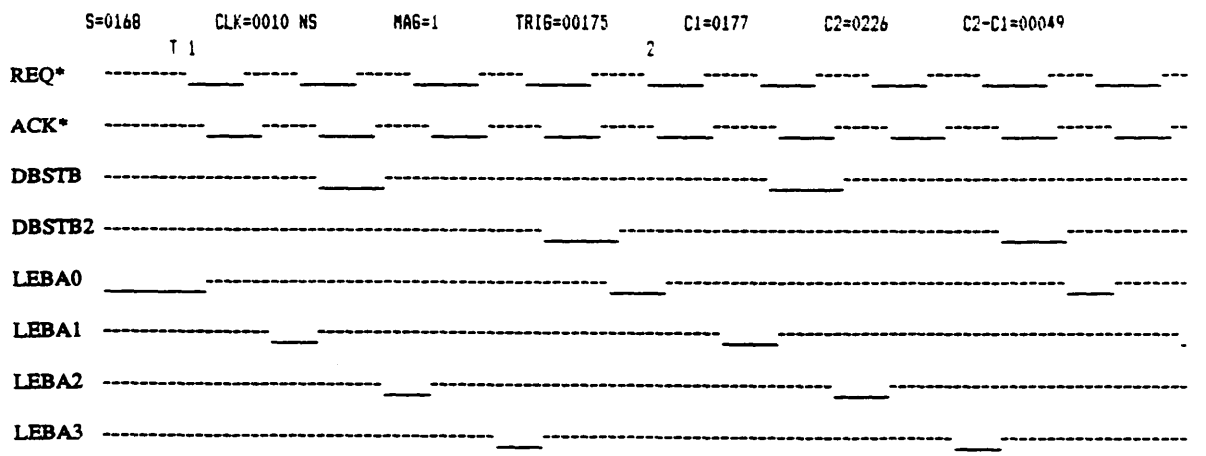
IV-3272 FIFO ⇒ IV-3272-PIO ⇒ P2
32 bits 32 bits



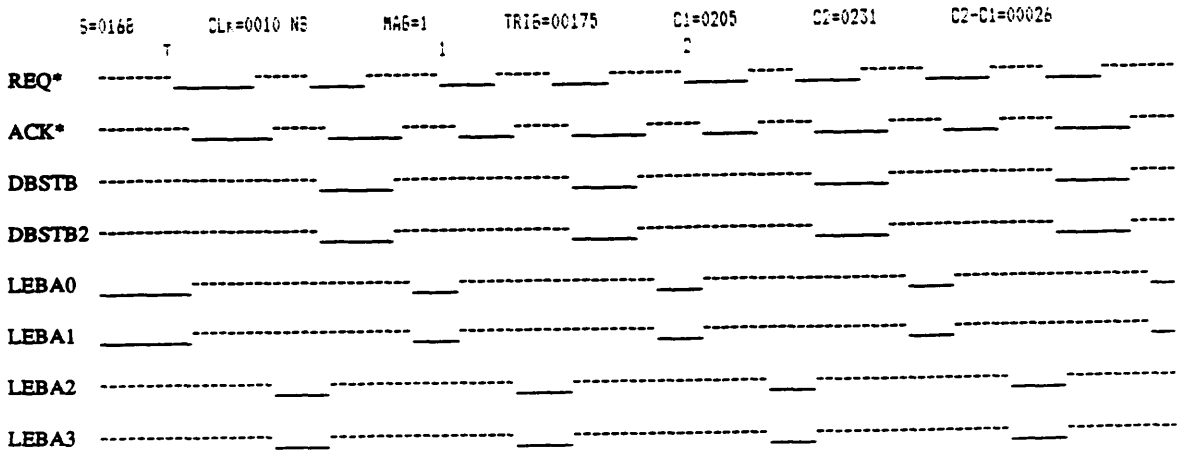
IV-3272 FIFO ⇒ IV-3272-PIO ⇒ P2
32 bits 16 bits



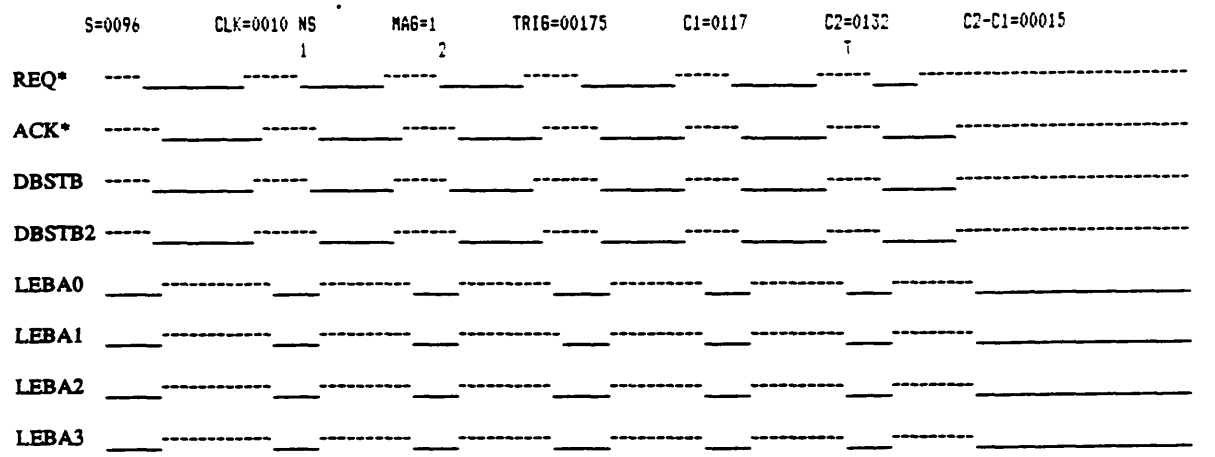
IV-3272 FIFO ⇒ IV-3272-PIO ⇒ P2
32 bits 8 bits



IV-3272 FIFO ⇐ IV-3272-PIO ⇐ P2
32 bits 8 bits



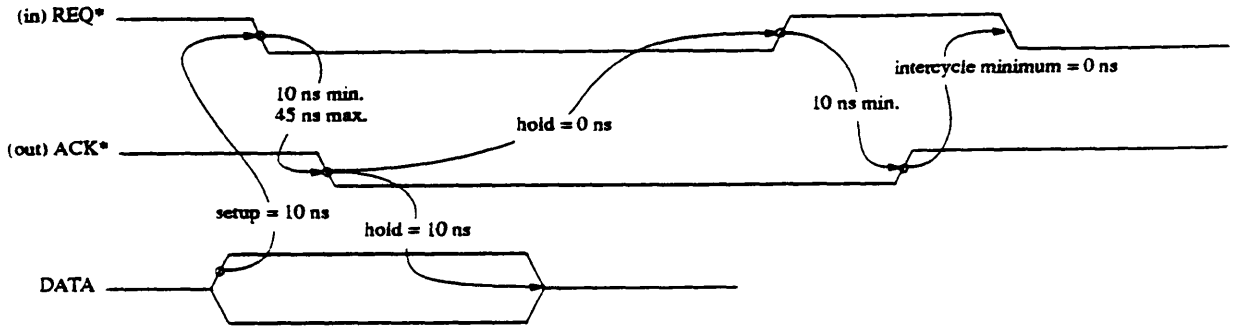
IV-3272 FIFO ⇐ IV-3272-PIO ⇐ P2
32 bits 16 bits



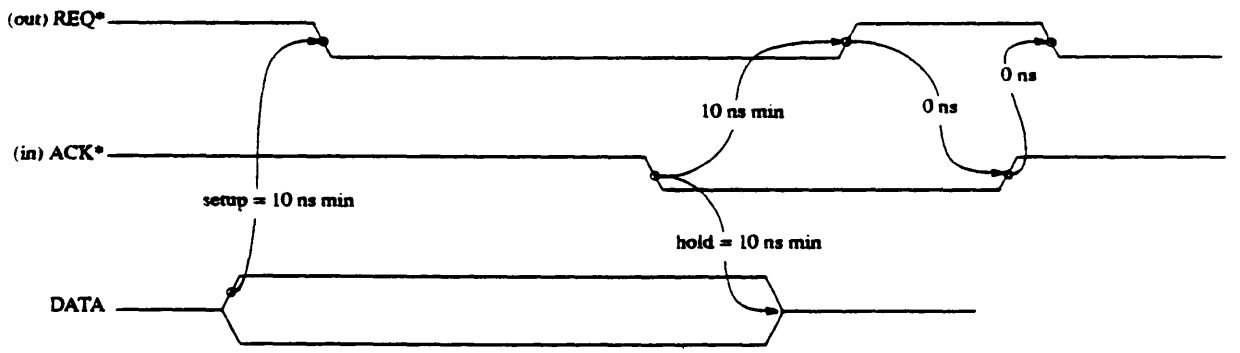
IV-3272 FIFO ⇐ IV-3272-PIO ⇐ P2
32 bits 32 bits

Figure 2-3 External Timing

Data in from P2



Data out to P2



2.4 Status and Control

Eight status and eight control signals may be received and sent by the IV-3272-PIO. These signals are intended to provide a control mechanism without using data lines. These signals may be used in systems where the control signals are routed via the cabling to the status signals of a second interface. Because of the crossover requirement, these signals are not ideal for connection of more than two IV-3272-PIOs. They can be used to facilitate software handshaking between two IV-3272-PIO interfaces or may be adapted to connect to a proprietary interface.

Two 8-bit latches are available for holding a control byte. One latch may be set by writing to the IV-3272 VMEbus control register, bits D08 to D15. The other latch may be set by writing to the DSP program memory at address C00 using bits D08 to D15. The individual outputs of these latches are connected to P2 via a set of three-position shunts, J1 through J8. Is a shunt is in the 1-2 position, the DSP latch is connected to P2. If the shunt is in the 2-3 position, the VMEbus latch bit is connected to P2. The ability to mix and match latch drives which signal provides flexibility as to the source of individual control bits.

The status input signals (P2 C32, lsb to C25, msb) are routed to two buffers. These buffers are read by the DSP at program memory location C00 and by the IV-3272 VMEbus status register. Status input information is presented in bits D08 - D15.

Table 2-3 Control Byte Shunts

Bit	Shunt	P2 Pin
D08	J8	C24
D09	J7	C23
D10	J6	C22
D11	J5	C21
D12	J4	C20
D13	J3	C19
D14	J2	C18
D15	J1	C17

3 Service and Repair

3.1 Unpacking Instructions

All Ironics products are manufactured and tested in a static-controlled environment to ensure minimal degradation of component performance due to electrical discharge. All boards are placed in an electrically conductive plastic wrapping for static protection during shipping. The following precautions should be observed during unpacking and installation:

1. All individuals handling Ironics boards should be properly grounded in static-controlled work areas.
2. Boards must be handled by their edges, avoiding any contact with connector surfaces.
3. A board must *never* be installed in or removed from a powered-on system.

3.2 Inspection

Inspect the board after removing it from the protective wrapping. Remove any loose packing material from the board surface. Examine the board for possible shipping damage; check all chips in sockets (EPROMs, PLAs, etc.) for loose seating. If necessary, reseat loose chips by applying even pressure to the top of the chip. Verify the integrity of any jumper wires.

Immediately report any shipping damage to Ironics Technical Support as outlined in the following section

3.3 IV-3272-PIO Daughter Board Installation

If the IV-3272-PIO daughter board is shipped separate from the IV-3272 mother board, the following procedure must be followed to install the daughter board:

1. On the IV-3272 mother board, remove PLA 594.4 from board location U34 and replace it with PLA 594.3 shipped with the daughter board
2. Mount the IV-3272-PIO daughter board on the mother board so that board locations U13 and U23 are facing the VMEbus P2 connector on the mother board.

3.4 Getting Technical Assistance

Technical support is available to all Ironics users. Telephone support is available every weekday between 9:00 am and 5:30 pm EST (other hours may be arranged in advance). Questions may also be mailed, FAXed, or telexed to Ironics at any time. To contact Ironics Technical Support Services write, call or FAX:

Ironics, Inc.
Technical Support Services
798 Cascadilla Street
Ithaca, New York

Telephone: (607) 277-4060
Telex: 705-742
FAX: 607-272-5787

In case of a problem operating this board, it is recommended that initial contact with Ironics Technical Support Services be made by telephone; the problem might well be solvable in a verbal conversation, especially if Ironics-provided code is installed on the board. For problems involving code not provided by Ironics, an initial telephone call is especially useful, to determine whether it is necessary to send the user's code to Ironics. To get maximum effect from an initial telephone call, follow these guidelines in preparing for the call:

1. Have available the board's model number, purchase date, serial number, and product description. (These items are printed on the product invoice.) System configuration information (i.e., boards in the system; board addresses; bus levels; etc.) may also be necessary. The user's name, telephone number, address, company name, and department will be requested.

2. Make sure that the problem is repeatable. Try the operation several times, if possible, to determine whether the system response is repeatable under identical circumstances. If the problem is not repeatable, or if the responses vary for multiple attempts, carefully note any variations in the input process, and try to discern a correlation between those variations and the differences in system response. In any case, document all parts of the problem thoroughly, and have written notes handy for the telephone call.

3. Be seated at the system console during the telephone call, and be ready to execute the problem operation on the machine. Have all pertinent manuals and documentation immediately available.

If it should become necessary to send to Ironics (e.g., by mail, electronic mail, FAX), the user's code under which the board was operating, be sure to include the documentation discussed in item 2 above, as complete a description as is practicable of the system and the circumstances in which the board was operating, and an explanation of the intended function of the code. Ironics' response time will depend on the complexity of the problem and the quality of the information supplied by the user.

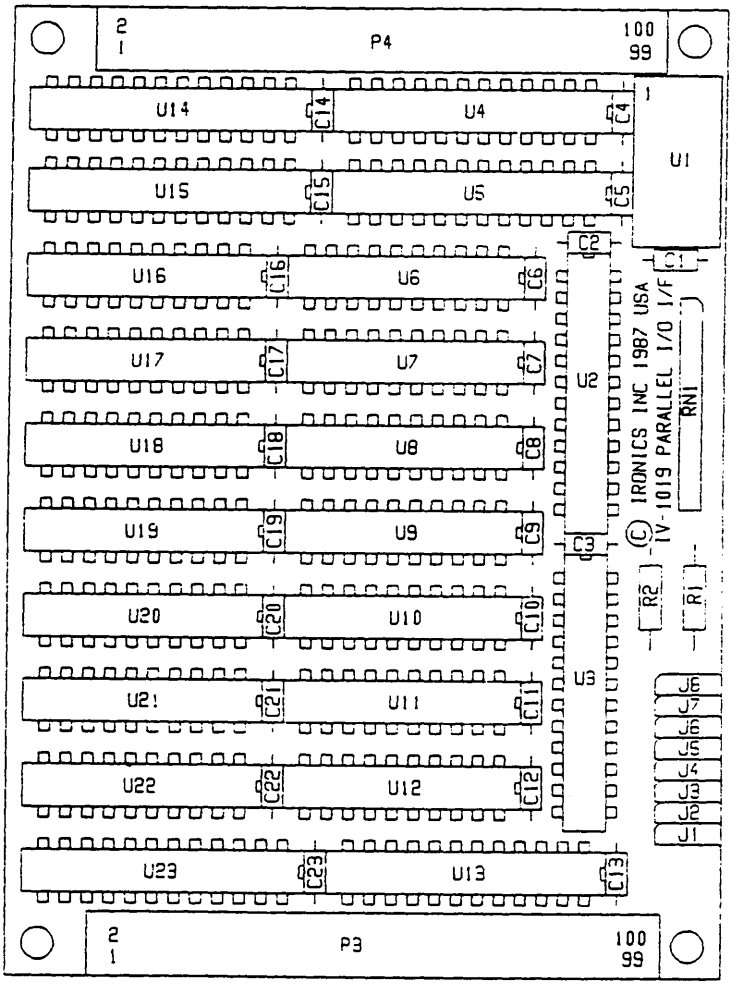
Table A-1 VMEbus J2/P2 Row A Pin Assignments

Pin Number	Schematic Signal	Signal Name	Pin Number	Schematic Signal	Signal Name
A1	P2.1	IOBUS15	A17	P2.17	REQ*
A2	P2.2	IOBUS14	A18	P2.18	ACK*
A3	P2.3	IOBUS13	A19	P2.19	Reserved
A4	P2.4	IOBUS12	A20	P2.20	Reserved
A5	P2.5	IOBUS11	A21	P2.21	Reserved
A6	P2.6	IOBUS10	A22	P2.22	Reserved
A7	P2.7	IOBUS9	A23	P2.23	Reserved
A8	P2.8	IOBUS8	A24	P2.24	ENDIR
A9	P2.9	IOBUS7	A25	P2.25	GND
A10	P2.10	IOBUS6	A26	P2.26	Reserved
A11	P2.11	IOBUS5	A27	P2.27	GND
A12	P2.12	IOBUS4	A28	P2.28	Reserved
A13	P2.13	IOBUS3	A29	P2.29	GND
A14	P2.14	IOBUS2	A30	P2.30	Reserved
A15	P2.15	IOBUS1	A31	P2.31	GND
A16	P2.16	IOBUS0	A32	P2.32	Reserved

Table A-2 VMEbus J2/P2 Row C Pin Assignments

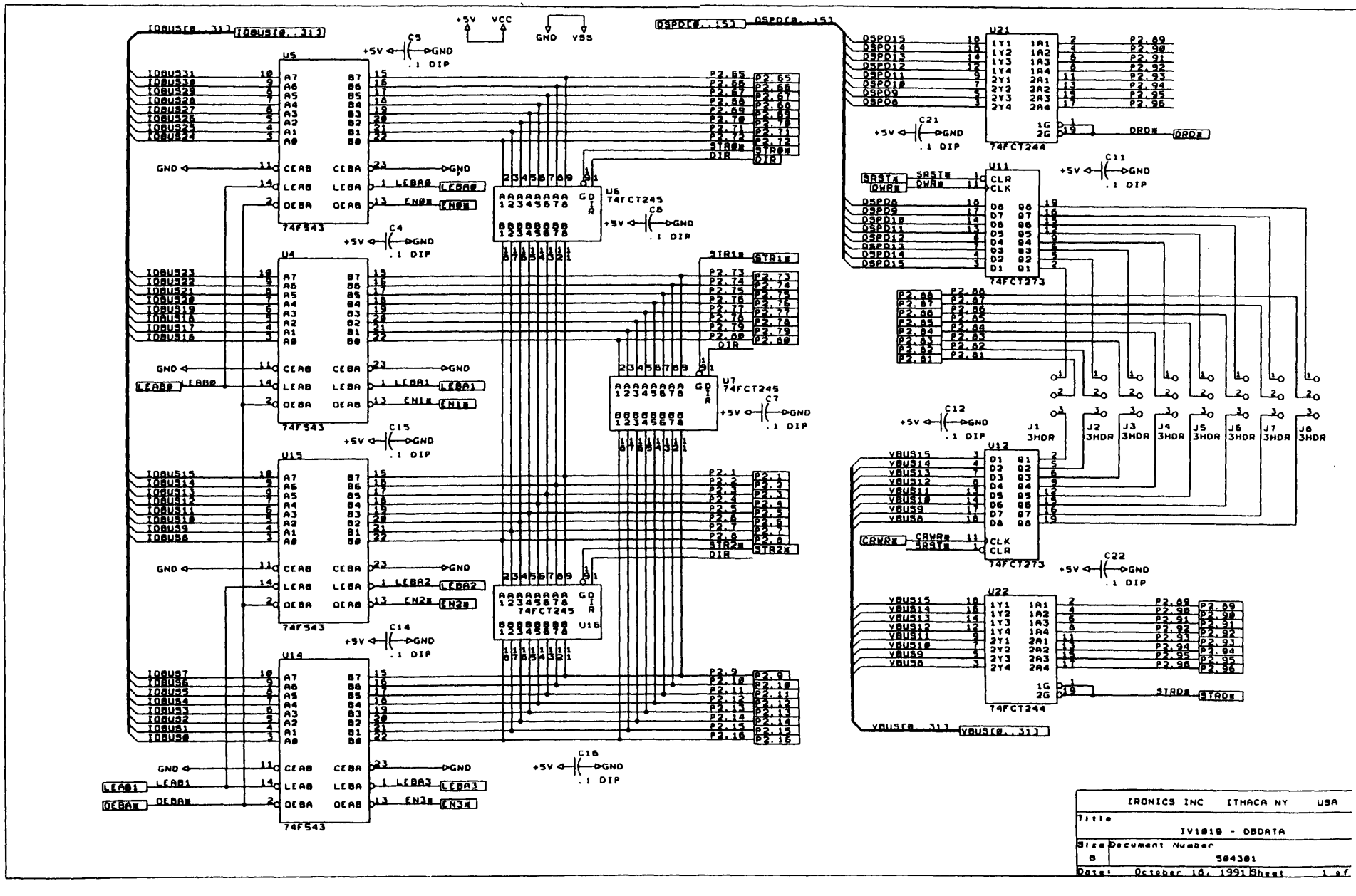
VME P2 Pin Number	Schematic Signal	Signal Name	
C1	P2.65	IOBS31	
C2	P2.66	IOBUS30	
C3	P2.67	IOBUS29	
C4	P2.68	IOBUS28	
C5	P2.69	IOBUS27	
C6	P2.70	IOBUS26	
C7	P2.71	IOBUS25	
C8	P2.72	IOBUS24	
C9	P2.73	IOBUS23	
C10	P2.74	IOBUS22	
C11	P2.75	IOBUS21	
C12	P2.76	IOBUS20	
C13	P2.77	IOBUS19	
C14	P2.78	IOBUS18	
C15	P2.79	IOBUS17	
C16	P2.80	IOBUS16	
C17	P2.81	DSPD15/VBUS15	Always outgoing: driven by DSP or VME as set by shunts J1 - J8. VBUS8 - 15 equals top eight bits of IV-3272 Control Register
C18	P2.82	DSPD14/VBUS14	
C19	P2.83	DSPD13/VBUS13	
C20	P2.84	DSPD12/VBUS12	
C21	P2.85	DSPD11/VBUS11	
C22	P2.86	DSPD10/VBUS10	
C23	P2.87	DSPD9/VBUS9	
C24	P2.88	DSPD8/VBUS8	
C25	P2.89	DSPD15/VBUS15	Always incoming: readable by both VME and DSP. VBUS8 - 15 equals top eight bits of IV-3272 Status Register.
C26	P2.90	DSPD14/VBUS14	
C27	P2.91	DSPS13/VBUS13	
C28	P2.92	DSPD12/VBUS12	
C29	P2.93	DSPD11/VBUS11	
C30	P2.94	DSPD10/VBUS10	
C31	P2.95	DSPD9/VBUS9	
C32	P.96	DSPS8/VBUS8	

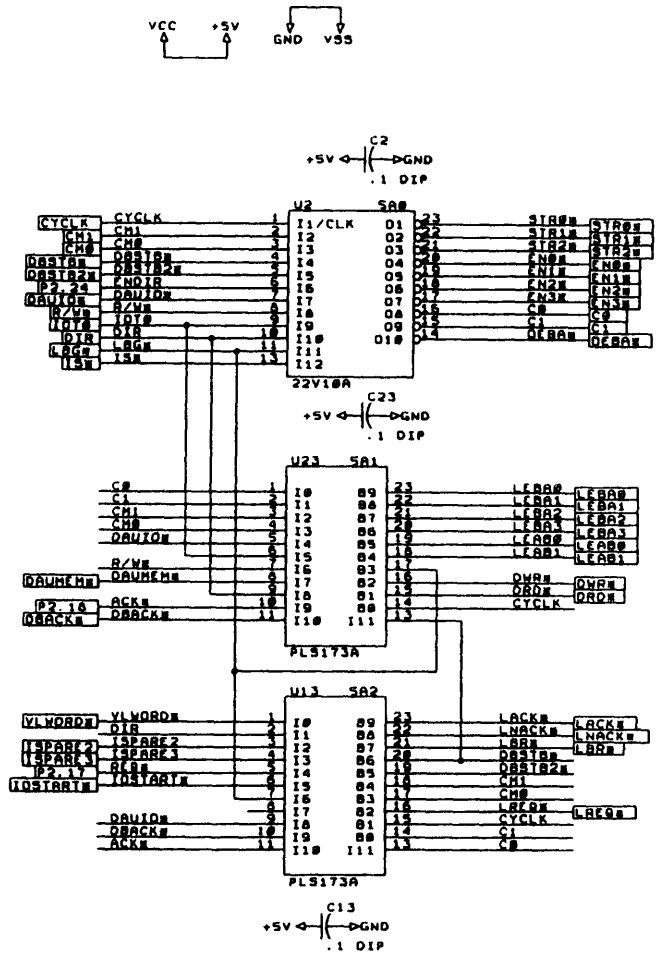
B.1 Component Layout Diagram



B.2 Schematics

The schematics presented here represent board etch 1.10 of the IV-3272-PIO daughter board.


IRONICS INC ITHACA NY USA
Title
IV1819 - DBDATA
Size B
Document Number
504381
Date: October 16, 1991 Sheet 1 of



COUNT MODE DECODING

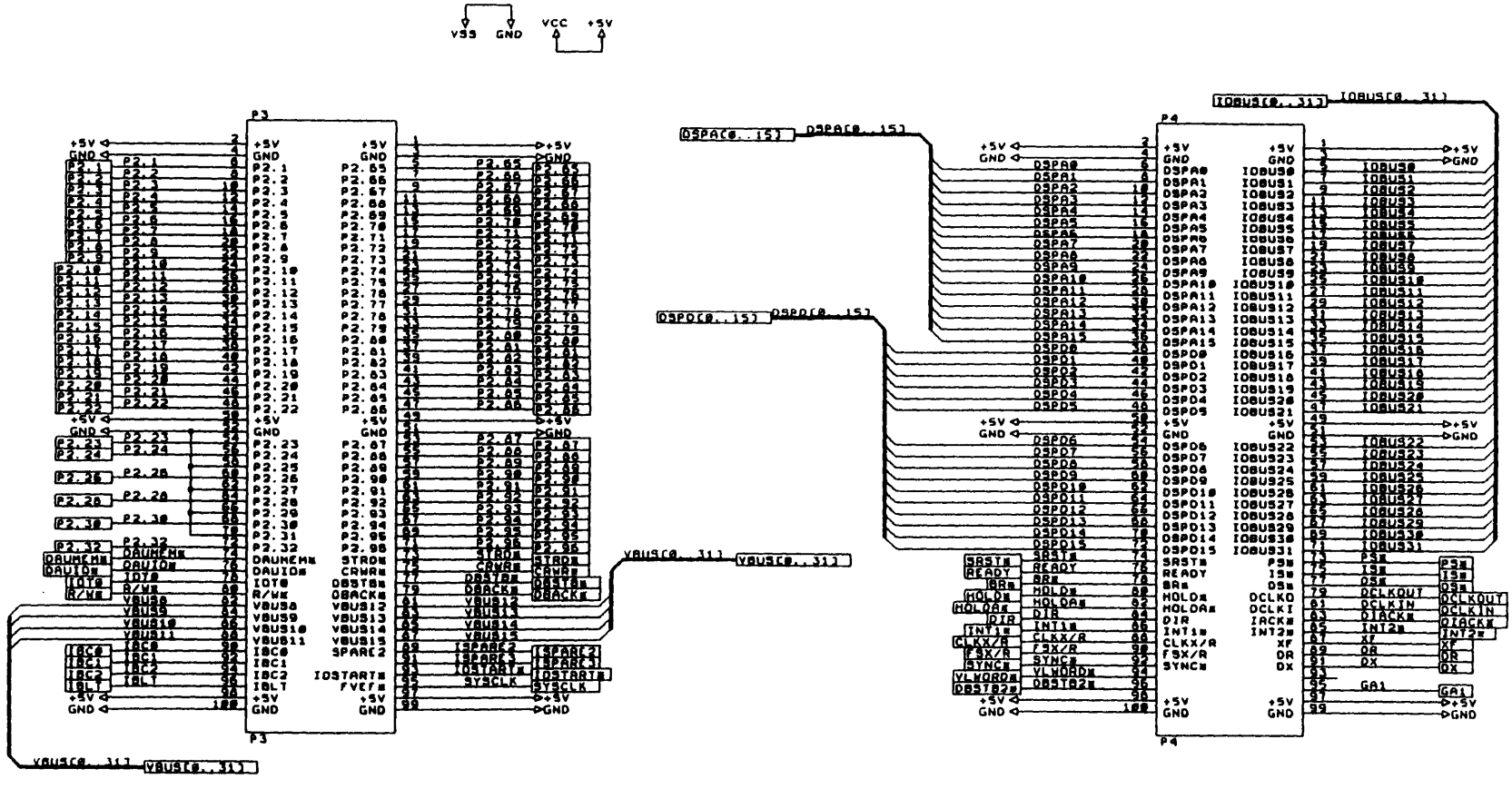
CM1	CM0	FIFO	P2.I/O
0	0	32	8
0	1	32	16
1	0	16	8
1	1	16	16/32 : 32

TRANSFER DIRECTION

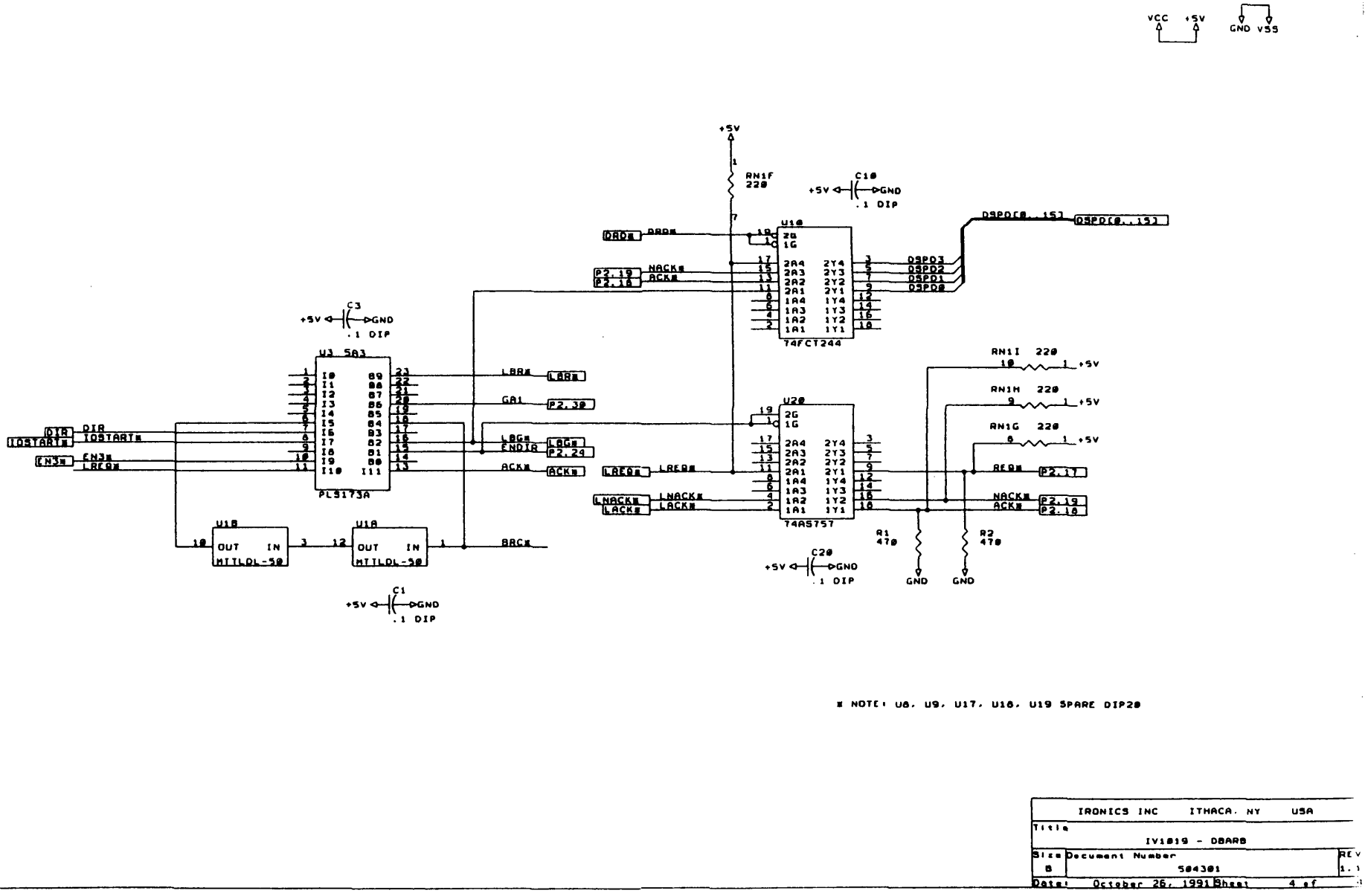
DIR = 0 P2 → VME

DIR = 1 VME → P2

IRONICS INC ITHAC NY USA		
Title: IV1019 - DBCNTL		
Size: B	Document Number: 504301	REV: 1.1
Date: October 21, 1991	Sheet: 2 of	



IRONICS INC ITHACA, NY USA		
Title: IV1819 - CNINT		
Size: B	Document Number: 504301	REV: 1.1
Date: October 18, 1991	Sheet: 3 of	



NOTE: U8, U9, U17, U18, U19 SPARE DIP20

IRONICS INC ITHACA, NY USA		
Title: IV1819 - DBARB		
Size: B	Document Number: 504301	REV: 1.1
Date: October 26, 1991	Sheet 4 of	

C
Parts List

Table C-1 Abbreviations for Manufacturers

Abbreviation	Manufacturer
3M	3M
AI	Advanced Interconnect
AMD	Advanced Micro Devices
AMP	Amp
APT	Aptronics
AVX	AVX
BOU	Bourns
EC2	EC2 Incorporated
IRO	Ironics Inc.
SIG	Signetics Corporation

C.1 Core Assembly

C.1.1 Sockets

Qty	Loc. No.	Manu	Manu. Part No.	Inventory Part No.	Description
4	U2, U3,U13,U23	AI	KS324-176TG	335231	24-pin low-profile socket (0.3" spacing)

C.1.2 Integrated Circuits

Qty	Loc. No.	Manu	Manu. Part No.	Inventory Part No.	Description
1	U1	EC2	MTTLDL-50	621000	Triple 50ns delay line
4	U4, U5,U14, U15	SIG	74F543	444543	IC 74F543
3	U6, U7,U16	SIG	74FCT245	481245	IC 74FCT245
3	U10, U21,U22	SIG	74FCT244	481244	IC 74FCT244
2	U11, U12	SIG	74FCT273	481273	IC 74FCT273
1	U20	SIG	74AS757	449757	IC 74AS757
0	U8, U9,U17, U18, U19	DO NOT INSTALL			20P SPARE

C.1.3 PLAs

Qty	Loc. No.	Manu	Manu. Part No.	Inventory Part No.	Description
2	U2, U3	AMD	AMPAL22V10ADC	450000	PLA 5A0 PLA 5A3
2	U13, U23	AMD (AMD (AMD	AMPAL22V10BPC AMPAL22V10-15PC AMPAL22V10-15BSB	450033))	PLA 5A2 PLA 5A1

C.1.4 Resistor Networks

Qty	Loc. No.	Manu	Manu. Part No.	Inventory Part No.	Description
1	RN1	BOU	4610X-101-221	604037	10-pin 220 ohm SIP

C.1.5 Resistors

Qty	Loc. No.	Manu	Manu. Part No.	Inventory Part No.	Description
2	R1,R2	-------	----------	600471	470 ohm 1/4W 5%

C.1.6 Capacitors

Qty	Loc. No.	Manu	Manu. Part No.	Inventory Part No.	Description
18	C1-C7, C10-C16, C20-C23	AVX (AVX	MD015E104ZAA MD015E104ZAB	610104)	0.1 uF dipguard
0	C8, C9, C17-C19	DO NOT INSTALL			SPARE

C.1.7 Miscellaneous

Qty	Loc. No.	Manu	Manu. Part No.	Inventory Part No.	Description
1	PC	IRO	504301	504301	IV-1019 1.1 PC board
1	J1-J8:1	3M	929800-01-36	334158-08	1 x 8 LP header
1	J1-J8:2-3	3M	929800-01-36	334162-08	2 x 8 LP header
2	P3, P4	AMP	104078-8	335101	Receptacle 100 pos. .050 H. density minimax

MANUAL EVALUATION FORM

As part of our continuing effort to improve the quality of Ironics products and documentation, we ask that you provide us with your comments and suggestions for improving this publication. Your response to the questionnaire below represents a critical link in our manual review and update process, and your contribution is greatly appreciated.

Manual Title:	Manual Version:†	Manual Part No.:†

† Manual version and part number are given on the back of the title page.

- How are you using this manual?

  ❑ Learning ❑ Installation ❑ Operation ❑ Reference ❑ Maintenance ❑ Sales

- How do you rate this manual with respect to...

	EXCELLENT	GOOD	FAIR	POOR	VERY POOR
...technical completeness?	❑	❑	❑	❑	❑
...organization?	❑	❑	❑	❑	❑
...readability?	❑	❑	❑	❑	❑
...accuracy?	❑	❑	❑	❑	❑
...usefulness as a reference?	❑	❑	❑	❑	❑
...visual presentation?	❑	❑	❑	❑	❑
• How do you rate its overall effectiveness?	❑	❑	❑	❑	❑

Ironics has made every effort to ensure the accuracy, completeness, and consistency of the information provided to our customers; oversights and mistakes, however, can occur. Please inform us of any errors or inconsistencies that you may have noticed in the text, noting the page and section numbers. We also welcome any general comments or suggestions for improvement that you might have.

_____

_____

_____

_____

_____

_____

_____

Date: _____ Your Name: _____

Company: _____

Mailing Address: _____

_____

_____

Telephone Number: _____

Just tear this sheet off at the perforation, fold it as shown on the back, tape it closed (please use transparent tape), and mail it to us. *No staples, please.* Thanks!

PLEASE...

...make the folds in the order indicated
...fold carefully on the lines
...use transparent tape to seal
...do not use staples

FOLD HERE FIRST

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 944 ITHACA, NY

POSTAGE WILL BE PAID BY ADDRESSEE

IRONICS Incorporated

ATTN: Publications Manager
Computer Systems Division
798 Cascadilla Street
Ithaca NY 14850-9908

FOLD HERE LAST