

ARMY RESEARCH LABORATORY



Image Reconstruction Board

by Daniel F. McCarthy

ARL-MR-308

June 1996

PROPERTY OF
US ARMY
RESEARCH LABORATORY
LIBRARY

copy 3

Approved for public release; distribution unlimited.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE June 1996		3. REPORT TYPE AND DATES COVERED Final, January 1994 to December 1996
4. TITLE AND SUBTITLE Image Reconstruction Board			5. FUNDING NUMBERS PE: 62120A	
6. AUTHOR(S) Daniel F. McCarthy				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory Attn: AMSRL-SE-ES 2800 Powder Mill Road Adelphi, MD 20783-1197			8. PERFORMING ORGANIZATION REPORT NUMBER ARL-MR-308	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) U.S. Army Research Laboratory 2800 Powder Mill Road Adelphi, MD 20783-1197			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES AMS code: 622120.H16 ARL PR: 6NE420				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) This report describes an image reconstruction board developed as part of an interface requirement for an acousto-optic (AO) range doppler radar processor, and presents a brief summary of the program for which the board was designed. A description is given of the charge coupled device (CCD) camera interface in which the board resides, and the board's algorithm and schematic block diagram are discussed. A full set of board-level schematics is given and briefly discussed. The paper also lists and discusses the VHSIC hardware description language (VHDL) code used to program the field programmable gate arrays (FPGAs) used in the design. The CAD CAM system used to simulate and build the board is described. VHDL and simulator command files used in the simulation are included. Finally, simulation and hardware test results are presented and compared. Sections 3.2 and 3.3 give detailed descriptions of the electronics schematics and VHDL code. These sections are for the reader who requires a detailed understanding of the electronics of the board; the casual reader may wish to skip these sections.				
14. SUBJECT TERMS TOPS, radar, FPGA, VHDL			15. NUMBER OF PAGES 101	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

Contents

1. Introduction—TOPS-MICOM Radar	5
2. CCD Detector Array Interface	7
3. Board Design	9
3.1 Algorithm	9
3.2 Schematic Block Diagram	10
3.3 FPGA and VHDL	12
3.3.1 Switch matrix	12
3.3.2 Controller	12
3.4 Simulation, PCB Design, and Testing	16
3.5 I-O Specifications	18
4. Conclusions and Current Status	22
References	23
Acknowledgments	23
Distribution	109

Appendices

A.—Image Reconstruction Electrical Schematics	25
B.—VHDL Files	49
C.—Viewsim Command Files	99

Figures

1. MICOM MRSR optical processor insertion	5
2. Image reconstruction board	6
3. 512 by 512 split-frame CCD imager	7
4. Image reconstruction algorithm	9
5. Image reconstruction schematic block diagram	10
6. Input data timing waveforms	19
7. Output data timing waveforms	20

Table

1. Input/output (I/O) mapping for four VHDL cases	12
---	----

1. Introduction—TOPS-MICOM Radar

The Advanced Research Project Agency (ARPA) sponsored the Transition of Optical Processors into Systems (TOPS) program to demonstrate that optical processors can perform effectively in military systems. The Army Research Laboratory (ARL) and Dynetics developed an optical radar signal processor to interface with the multi-role survivable radar (MRSR) developed by the U.S. Army Missile Command (MICOM). Figure 1 shows a block diagram of the MICOM radar and the TOPS optical processing system.

TOPS-MICOM radar presently consists of a receiver/exciter, analog to digital (A/D) converter, clutter canceler, doppler filters, code correlator, target detector, and radar computer. The optical processing system developed by ARL and Dynetics replaces the doppler filters and the code correlator, and contains a receive-signal conditioner, a radio frequency (rf) interface, an optical processor, a charge coupled device (CCD) camera interface, a high-speed post-processor, and an output signal conditioner. The rf interface, optical processor, and a CCD camera interface were built by ARL; the receive-signal conditioner, a high-speed post-processor, and an output-signal conditioner were built by Dynetics [1,2].

The image reconstruction board (see fig. 2) is part of the CCD camera interface; it reformats the digitized CCD output data and passes it to the Dynetics high-speed post-processor. The board is an asynchronous, differential data link between the output of the ARL system and the Dynetics high-speed post-processor.

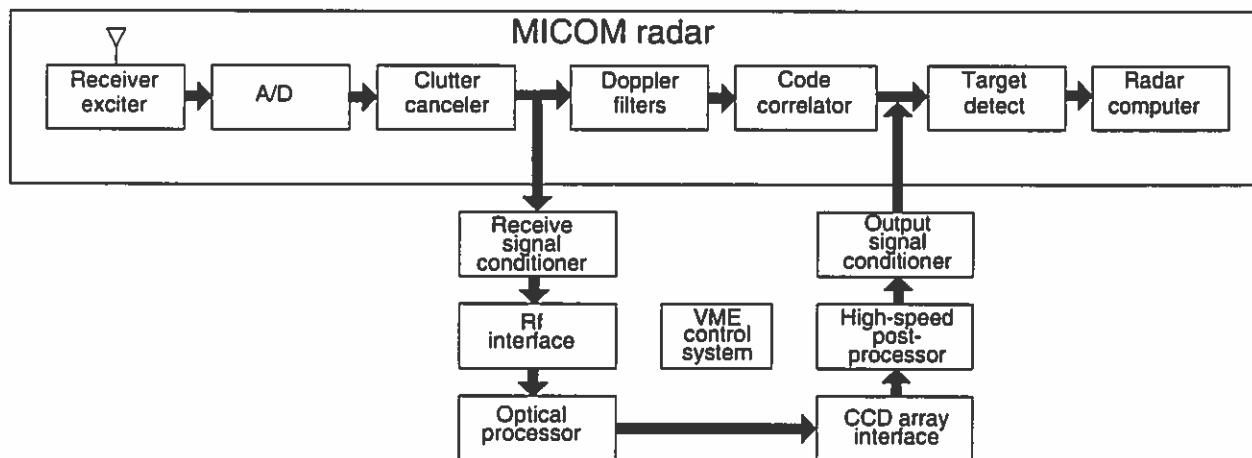
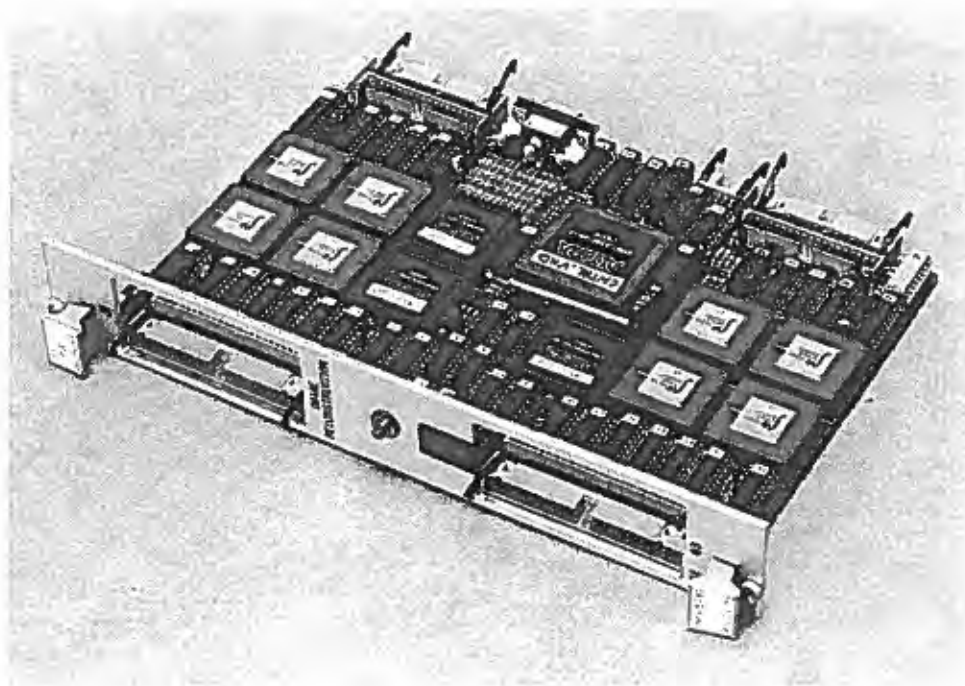


Figure 1. MICOM MRSR optical processor insertion.

**Figure 2. Image
reconstruction board.**

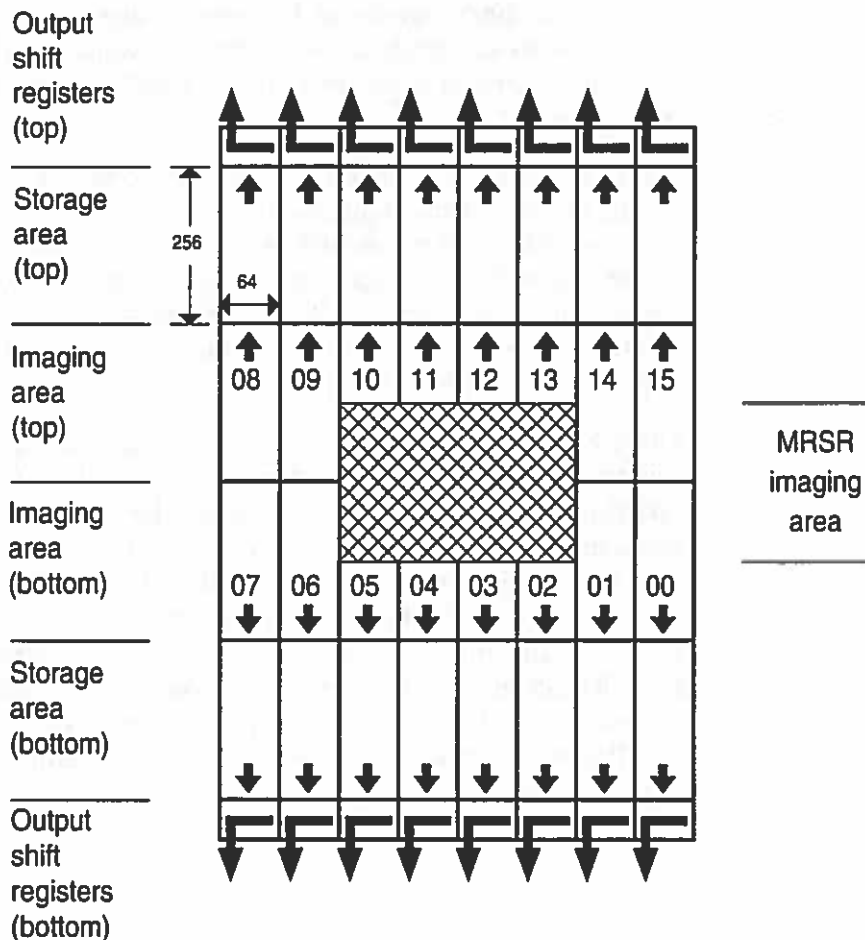


2. CCD Detector Array Interface

The CCD camera interface connects the output of the optical processor to the Dynetics high-speed post-processor. A CCD camera converts the optical output into eight parallel analog video output voltages. Two A/D converter boards digitize the analog video signals; two image reconstruction boards buffer and reformat the digitized data and pass it to the Dynetics high-speed post-processor.

The output of the optical processor is focused on the imaging area of the CCD camera. The CCD imager shown in figure 3 is a split-frame imager with 16 parallel output video channels. Each channel consists of an imaging area of 256 rows of 64 pixels, a storage area of 256 rows of 64 pixels, and a 1-by-64-pixel output shift register. Eight channels make up the top half of the imaging area, 256 by 512 pixels; the other eight channels make up the bottom half. The total imaging area of the device is 512 by 512 pixels. The device is read out by vertically shifting the 256 imaging-area rows into the storage area. The storage area is then shifted out through the output register by vertically shifting a row into the output register, then horizontally shifting the register 64 times. This process is repeated 256 times—once for every row in the storage area. The output of the channel is a 1-by-16,384 pixel analog video signal [3,4].

Figure 3. 512 by 512 split-frame CCD imager.



The MRSR optical processor requires an imaging area of 256 by 256 pixels. The optical processor output is focused on the center of the CCD imaging area. The image lies on the center eight channels of the CCD—four on the top half and four on the bottom half. Each channel is only half-illuminated; the 128 rows toward the center of the channel are illuminated. The device package physically masks unilluminated portions of the CCD array. Eight half-channels are used to achieve the higher CCD read-out rate (frame rate) required by the MRSR radar signal-processor system. Using eight half-channels instead of four full channels almost doubles the frame rate. Using the center half of a channel with the outer half masked is equivalent to having a 128-row channel and a 384-row storage array. To shift-out a full 256-by-64 channel requires 512 vertical shifts and 16,384 horizontal shifts. To shift-out a half-channel requires 512 vertical shifts and 8192 horizontal shifts [5].

The radar has two dwell modes, long-dwell and short-dwell. In long dwell mode, a 256-by-256-pixel area is required, using the read-out scheme detailed above. In short-dwell mode, the integration time is reduced and an image resolution of 128 by 256 pixels is required. In both modes, 384 vertical clocks shift the image into storage. In short-dwell mode, two vertical shift clocks parallel load the output register, followed by 64 horizontal shifts, which empty the register; this cycle is repeated 64 times. The total number of clocks required in short-dwell mode to empty the CCD is 512 vertical and 4096 horizontal. Double-shifting the vertical register uses the register as an accumulator, **summing** two adjacent pixel rows. The shorter integration time in short-dwell mode precludes register saturation, due to row summing.

The high-speed post-processor must process each CCD frame of data to extract target-detection information, which corresponds to a dwell of radar data that has been processed by the optical processor. In the long-dwell mode, a 256-by-256 image must be processed into a 31-by-256 range-doppler map at a rate of 800 frames per second (fps). In the short-dwell mode, a 128-by-256 image must be processed into a 31-by-128 range-doppler map at 1600 fps [1].

There are two four-channel A/D boards and two four-channel image reconstruction boards in the system. Each side of the CCD array has an A/D board-image reconstruction board set. Each image reconstruction board reads in four 1-by-64 serial pixel channels from its A/D board, and generates a 1-by-256 serial pixel output. The 1-by-256 serial pixel outputs are stored in four parallel output fifo buffer channels. This process is repeated 128 times in long-dwell mode, and 64 times in short-dwell mode. Each 1-by-256 serial pixel output block is read by the post-processor and converted into a 1-by-32 pixel doppler map. The post-processor uses a 10-MHz read clock and eight parallel data channels to keep up with the frame rate of the CCD camera.

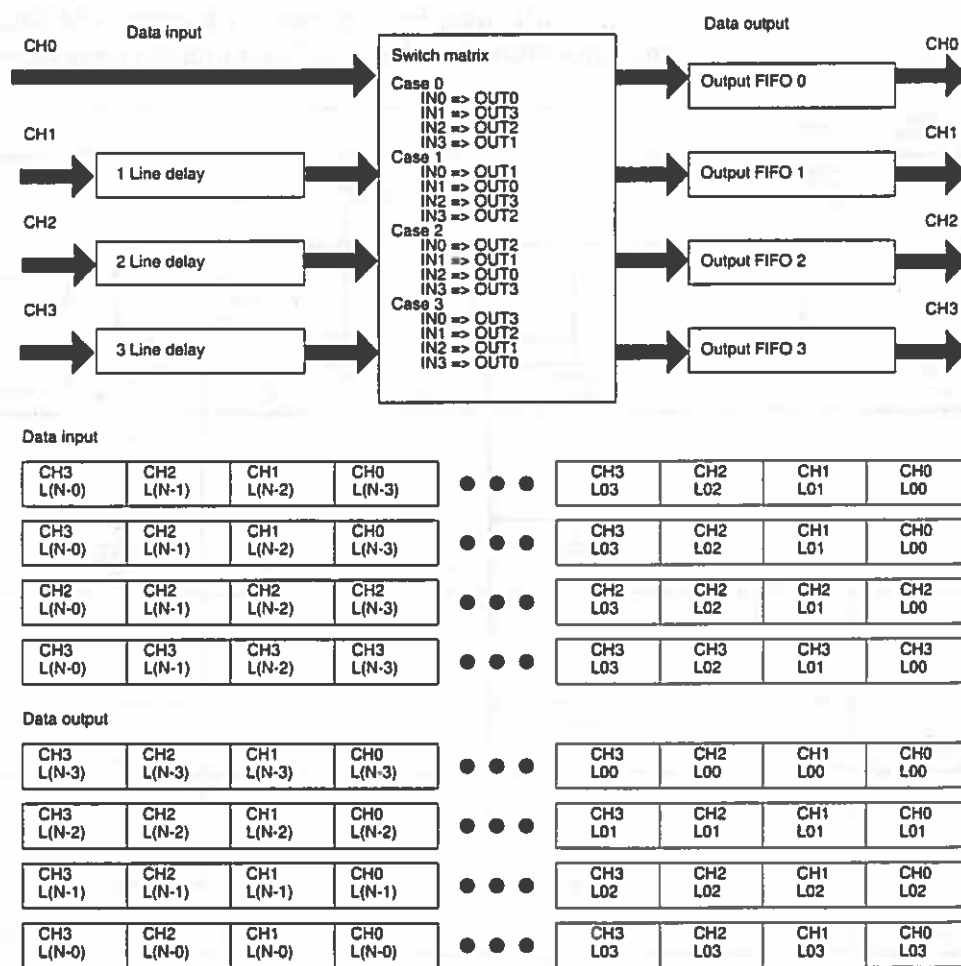
3. Board Design

3.1 Algorithm

Figure 4 shows a block diagram of the algorithm implemented by the image reconstruction board.

The image reconstruction board reads four channels of digitized data from the A/D board. The data is delayed and reordered through a switch matrix, then stored in four output fifo buffer channels. Channel 0 is read directly; channels 1, 2, and 3 are each delayed by one, two, and three data line widths, respectively, before being read. The switch matrix is then used to place the first data line from channels 0 through 3 into output fifo buffer 0. The switch matrix places the second data line from each channel in output fifo buffer 1. The third and fourth lines are placed into output fifo buffers 2 and 3; line four is placed back in output fifo buffer 0, behind the first line. In long-dwell mode, 128 data lines are processed this way; in short-dwell mode, 64 data lines are processed.

Figure 4. Image reconstruction algorithm.



N=128 Long dwell, N=64 Short dwell

3.2 Schematic Block Diagram

Figure 5 gives a schematic block diagram of the image reconstruction board. The board has four synchronous data pipeline stages (A, B, C, and D), and one asynchronous differential output stage. In stage A, latched input data is written to an input fifo bank. In stage B, delayed data is read out of the input fifo bank and presented to the switch matrix input. In stage C, reordered data from the switch matrix is written into the output fifo bank. In stage D, data is read out of the output fifo bank and driven to the Dynetics high-speed post-processor through the differential output buffer. The board has a master control chip that uses buffered control inputs from the CCD camera to generate control signals to push the input data through the synchronous data pipeline. The chip also receives and generates input/output (I/O) signals that control asynchronous data transfer to the Dynetics high-speed post-processor through a differential output control buffer. Appendix A presents a complete electrical schematic for the board.

The input control buffer receives pixel-clock, data-valid, and frame-valid signals from the CCD camera. The board uses DC-biased differential receivers, followed by jumper-programmable delay lines and fan-out transistor-transistor logic (TTL) buffer/drivers to receive and drive these

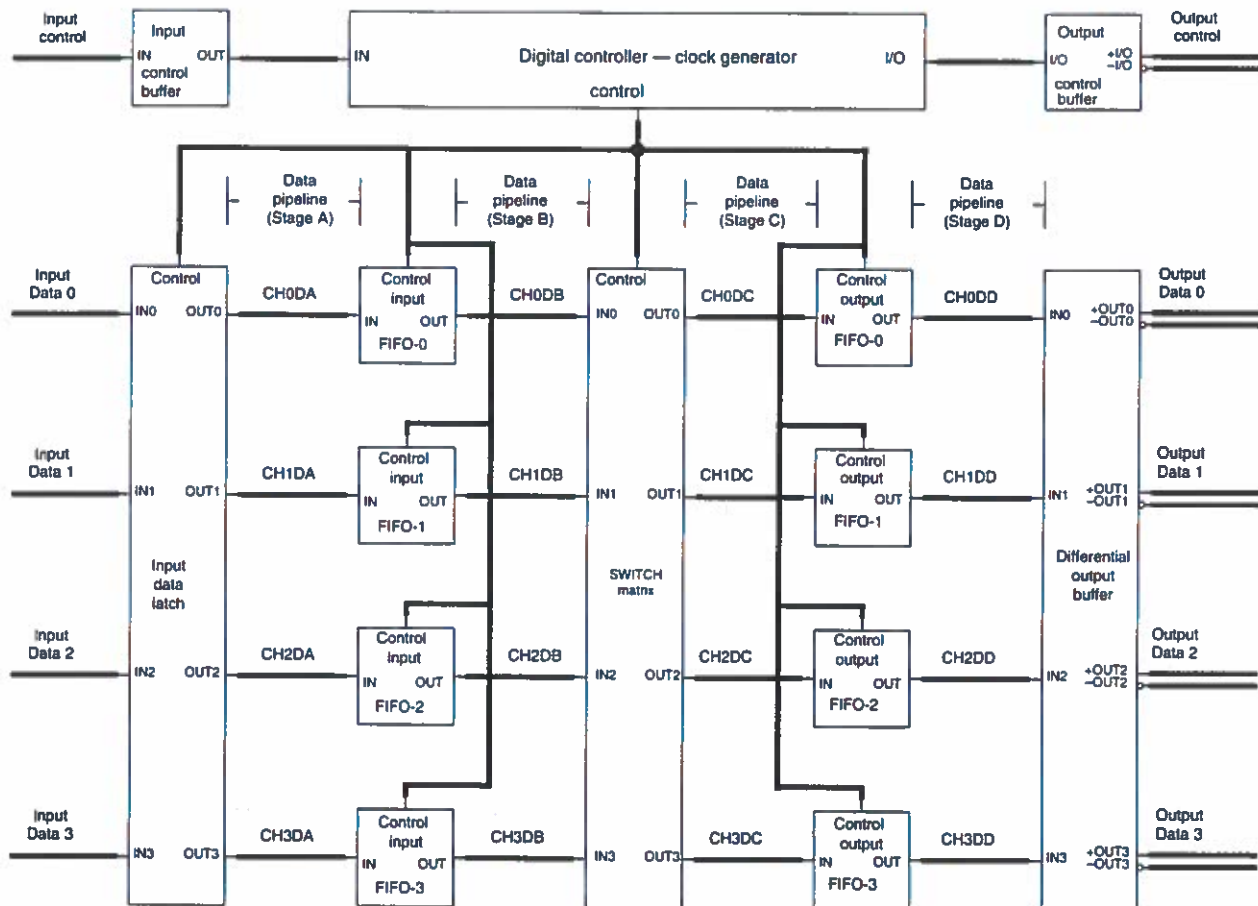


Figure 5. Image reconstruction schematic block diagram.

signals. The DC-biased differential receivers allow the board to receive either differential or single-ended TTL signals. The programmable jumpers allow the board to delay the three inputs relative to each other. The buffer/drivers drive the signals to the rest of the board. (Refer to electrical schematic sheets 2 of 10 and 10 of 10 for more detail.)

The input data latch stage uses 74F574 TTL data latches to capture the input data from the A/D board. Timing-control tags from the digital controller are also generated and captured with the data, and presented to the input fifo bank. Socketed dual in-line package (DIP) integrated circuits (ICs) were used to allow easy access to the input data for probing or rewiring. (Refer to electrical schematic sheets 2 of 10 and 3 of 10.)

The input fifo bank consists of four IDT72265 16K-by-18 clocked fifos [6]. The fifo bank delays the data and presents it to the switch matrix at the proper time. Fifo-0 delays the data by zero data lines, fifo-1 by one data line, fifo-2 by two data lines, and fifo-3 by three data lines. Fifo-0 is not necessary to perform the MRSR algorithm; it was added for symmetry, allowing the board to be reconfigured to perform other algorithms. (Refer to electrical schematic sheet 4 of 10.)

The switch matrix reorders the data lines as described in the algorithm section. Three ALTERA EPM7064LC68 [7] ICs are used to implement the switch matrix. Each EPM7064LC68 IC implements the case statement on four 6-bit data channels. The three ICs form a 4-channel 18-bit switch matrix. The switch matrix ICs receive a data clock from the clock buffer/drivers, and a 2-bit data-valid line counter (DVCNT[1:0]) from the master control chip. (Refer to electrical schematic sheet 5 of 10.)

The output fifo bank consists of four IDT72265 16K-by-18 clocked fifos. The fifo bank reads in the data output by the switch matrix. Data is read out of the fifos on request from the Dynetics high-speed post-processor. (Refer to electrical schematic sheet 6 of 10.)

The differential output buffer uses 75ALS192 and 75ALS193 DIP ICs to receive differential signals from and drive differential signals to the Dynetics high-speed post-processor. Differential signals were chosen to provide common mode rejection over potentially long ribbon cables between the ARL and the Dynetics systems. Differential signals also provide ground isolation between the two systems. Socketed DIP ICs were used to allow easy access to the output data for probing or rewiring. (Refer to electrical schematic sheets 7 of 10 and 8 of 10.)

An ALTERA EPM712GC160 [7] is used to implement the digital controller, generating all the digital signals necessary to control the board. It generates data-control flags that accompany the input data into stage A; generates input fifo control signals that write stage A data into the input fifos and read stage B data from the input fifos; counts data-valid lines to control the switch matrix; generates fifo control signals to write stage C data into the output fifo bank; and generates the signals necessary to read data from each of the output fifos and to communicate with the Dynetics

high-speed post-processor. It also generates controls for the front-panel seven-segment LED display (not shown in the block diagram). For a detailed description of the controller's function, see section 4.3.2 and appendix B, and refer to electrical schematic sheet 9 of 10.

3.3 FPGA and VHDL

3.3.1 Switch matrix

The switch matrix is implemented in three ALTERA EPM7064LC68 field programmable gate array (FPGA) ICs. The devices were programmed using a VHSIC hardware description language (VHDL) file (`v_matrix.vhd`) and the ALTERA VHDL compiler. Each device has four 6-bit inputs and four 6-bit registered outputs. There are also two control inputs, a rising edge clock (`clk`) and a 2-bit data valid count (`dvcnt`). The port statement in the VHDL file corresponds to the device pin-out. A VHDL case statement is used to implement the switch matrix algorithm; there are four cases: `dvcnt` equal from 0, 1, 2, or 3. Table 1 shows the I/O mapping for the four cases. The inputs are transferred to the outputs on the rising edge of the clock, as shown in appendix A, schematics `ir05.1`, `irs05.1`, `irs06.1`, and appendix B-1, VHDL file `v_matrix.vhd`.

Table 1. Input/output (I/O) mapping for four VHDL cases.

Case 0	Case 2
Output 0 = Input 0	Output 0 = Input 2
Output 1 = Input 3	Output 1 = Input 1
Output 2 = Input 2	Output 2 = Input 0
Output 3 = Input 1	Output 3 = Input 3
Case 1	Case 3
Output 0 = Input 1	Output 0 = Input 3
Output 1 = Input 0	Output 1 = Input 2
Output 2 = Input 3	Output 2 = Input 1
Output 3 = Input 2	Output 3 = Input 0

3.3.2 Controller

The controller is implemented in an ALTERA EPM7192GC160 FPGA IC. The device was programmed using a VHDL file (`v_cntrl.vhd`) and the ALTERA VHDL compiler. The device generates all signals necessary to control the operation of the board. The port statement in the VHDL file corresponds to the device pin-out. The VHDL code is divided into six major sections: Section I—Input Data Synchronization, Section II—Reset Cycle, Section III—Extra Data-Valid Cycle, Section IV—Pipeline Controller, Section V—Fifo Control, and Section VI—Output-Display Control. Descriptions may be found in appendix A, schematics `ir09.1`, `irs07.1`, `irs08.1`, and appendix B-2, VHDL file `v_cntrl.vhd`.

Section I of the code includes two processes: `SNC01` and `SNC02`. Process `SNC01` identifies the first four data-valid lines of a frame of data; `SNC02` generates timing-control tags that are sent to the input-data latch stage. `SNC02` has two modes: test mode and `MRSR` mode. The modes are con-

trolled by a dip switch, U29 pins 2 and 7. When the switch is closed, the signal (model) is a 0; this selects test mode. When the switch is open, MRSR mode is selected. In test mode, the first data-valid line in each frame and the first pixel of each data line are identified in each of the four input-data channels. In MRSR mode, the first pixel of the first four data-valid lines is identified for input-data channel 0. This translates through the switch matrix to identifying the first pixel of each frame in each of the four output-fifo channels.

Section II of the code includes seven processes: RST01 through RST07. These processes detect the presence of a reset-request signal on any of the four Dynetics output-data channels, or on the front panel reset button. If a reset is detected, it is validated and synchronized with the frame-valid signal from the CCD camera. A synchronized reset pulse is then sent to various sections of the board, initializing it to receive data at the beginning of the next frame of data. Processes RST01 through RST03 detect and validate a reset request; only requests that stay true for longer than two clock cycles are validated to prevent false triggering due to noise on a reset line. Process RST04 synchronizes the reset with the end of a frame-valid signal. RST05 generates a global-reset pulse and a reset gate pulse. RST06 is a counter that counts the number of pixels in the first data-valid line of a frame when the reset-gate pulse is true. RST07 ends the reset cycle after the first data-valid line length has been counted.*

Section III of the code includes six processes: XDV00 through XDV05. These processes generate three extra data-valid lines at the end of a frame-valid cycle. The three extra data-valid lines are necessary to push the data through the data pipeline and into the output-fifo buffers. The extra three lines make up for the three data-valid lines lost due to the three-data-line delay in input-fifo channel 3. Process XDV00 ends the extra data-valid cycle if a reset cycle is in progress, or if it receives an xstop signal. XDV01 starts the extra data-valid cycle at 13-clock cycles after the end of a frame. A 13-clock delay is added to compensate for the maximum first-word latency encountered when reading the first word written to an empty fifo. (Refer to preliminary data sheet IDT72255-56 page 10, EMPTY FLAG, t_{FWL1} and text detailing section IV of the code, below.)

Processes XDV02 and XDV03 generate a cycle that counts from one up to the number of pixels in the first data line, then sets a hold flag high for two clock cycles; this process is repeated three times. XDV04 detects the completion of the third cycle and generates the signal (xstop), which ends the extra data-valid cycle. XDV05 generates the output of the extra data-valid cycle (the signal xtradv). This generates the three extra data-valid pulses with a 2-pixel deadtime between pulses.

Section IV of the code includes 15 processes: PCT01 through PCT15. This group of processes generates the signals used to push data through the data pipeline. Process PCT01 produces the awrite signal; awrite combines

*The board assumes that all data-valid lines from this point until another reset is issued are of the same length.

the input-frame and data-valid signals from CCD to qualify the input data for writing into the four input fifos. PCT02 and PCT03 are shift registers that produce delayed versions of awrite and input-frame-valid.

PCT04 generates the bread signal; bread is the combination of the awrite signal delayed by 12 clock cycles, plus the xtradv signal delayed by a clock cycle. This signal qualifies the time when reading data from the input fifos is valid. The delay is necessary because there is a 12-clock maximum delay from the time that the first word is written to an empty fifo to the time when the first word can be read. The xtradv signal is delayed enough to provide a 2-clock cycle deadtime between the awrite and xtradv portions of bread. Processes PCT05 and PCT06 generate the B valid gates, bval[0-3]; these four gates are used to qualify bread for each input fifo-channel. Bval[0] qualifies the awrite portion and disqualifies the xtradv portion of bread for input-fifo 0. Bval[1] qualifies the awrite portion minus the first line, plus the first xtradv line of bread for input-fifo 1. Bval[2] disqualifies the first two lines of the awrite portion and qualifies the first two xtradv lines for input-fifo 2. Bval[3] disqualifies the first three lines of the awrite portion and qualifies the xtradv portion for input-fifo 3. Bread and bval[0-3] delay the input data as shown in figure 4 and read it to the switch matrix input at the proper time.

Cwrite[0-3] write the data output by the switch matrix to the output fifos. Each of the cwrite signals consists of two parts—cshort and cextra. PCT07 generates cshort[0-3]; they consist of a delayed awrite qualified with bval[0-3]. (These are the awrite portions of bread delayed by one clock cycle.) The xtradv portions of cwrite[0-3] depend on the mod4 remainder of the number of data-valid lines in the frame of data. For MRSR, the mod4 remainder is 0 in both long- and short-dwell modes, so the cwrite data is the bread data delayed by a clock cycle. The VHDL code also accommodates the other three unused cases. Processes PCT08 through PCT13 deal with all four cases. A comment section of the VHDL code gives timing-diagram representation of the four cases, and Karnaugh maps are given showing the logic reduction for each case. PCT14 combines cshort[0-3] with cextra[0-3] to form cwrite[0-3]. PCT15 produces the 2-bit data-valid counter that drives the switch matrix.

Section V of the code includes 28 processes: FFC01 through FFC28. This group of processes generates the signals used to control the eight fifos on the board. Signals from the reset cycle and the pipeline controller sections are combined to form the fifo read enables, write enables, and reset signals that push the data through the fifos. Control signals from the Dynetics high-speed post-processor are used to form the output-fifo-read enables. Control signals that accompany the data to the Dynetics processor are also generated. Process FFC01 generates the eight fifo-reset signals. FFC02 combines awrite with the reset gate to produce the input-fifo-write enable signals. FFC03 combines bread with the bval gates and the reset gate to form the input-fifo-read enables. FFC04 combines the cwrites with the reset gate to produce the output-fifo-write enables. Processes FFC05 through FFC28 generate the output fifo to the Dynetics post-processor interface. For each output channel, an output-fifo read-data-enable signal

and an output-data-available signal are formed; both of these signals are formed by combining the output-fifo-empty flag with a read-data-enable signal from the Dynetics processor.

The image reconstruction board sends input- and output-ready flags to the Dynetics post-processor. The flags tell the Dynetics processor when the image reconstruction board has data. The Dynetics processor sends the read-data-enable signal to the image reconstruction board when it wants to read data. When the image reconstruction board receives the read-data-enable signal, it transmits data, along with an output-data-available flag qualifying the data. The Dynetics processor continuously transmits a clock signal to read the data from the output fifo. The controller retransmits the clock back to the Dynetics post-processor to be used to latch the incoming data word. The controller can retransmit either the received clock or its inversion; this process is controlled by a dip switch—U29, pins 1 and 8. When the switch is closed, the signal (mode0) is a zero. This selects the non-inverted clock; when the switch is open, the inverted clock is selected. In MRSR mode, the switch is left open, selecting the inverted clock.

The U29 switch has four dip switches that are inputs to the controller chip. The signal mode0 controls the output clock. Mode1 controls test mode; mode2 on pins 3 and 5 is not used. On pins 4 and 5, the signal fifofs selects the fifo master clock. When fifofs is a 0, the fifo-read clock is used; when the switch is left open, the fifo-write clock is used. The switch should be left open in MRSR mode because the pixel clock from the CCD is faster than the Dynetics read clock. In MRSR mode, all three switches (mode0, mode1, and fifofs) should be left open.

Section VI of the code includes four processes: ODC01 through ODC04. This group of processes generates the signals used to control the seven-segment LED front-panel display. The display can output one of six states: a 0 signifies that all eight fifos on the board are empty; a 1 indicates that not all fifos are empty, but none are full; a 2 means all fifos are not empty but none are full (note that this state can never occur, since state 1 must also be true for it to be true); state 3 says that at least one fifo is full; state 4 says all fifos are full (another impossible state because of how the input fifos are set up); and state 5 indicates that the board is not receiving a pixel clock from the CCD.

LED states 0 and 1 are the states that the board should remain in when operating correctly. The user should see a 0 and a 1 superimposed on each other. The relative brightness depends on the Dynetics read-out clock speed, which translates to how much time is left over between the time the Dynetics processor empties the last pixel from the current frame until the board reads the first pixel of the next frame. If a 3 is visible on the display, an error is occurring—the Dynetics processor is not reading data fast enough to keep up with the CCD frame rate. If a 5 is displayed, there is no CCD clock present, and the board will not operate.

3.4 Simulation, PCB Design, and Testing

I built the image reconstruction board using three primary CAD CAM tools: Powerview from Viewlogic, Maxplus2 from Altera, and Scicards from Harris. I used Powerview for schematic capture and board-level simulation; Maxplus2 for FPGA design; and Scicards for printed circuit board (PCB) design. Powerview and Scicards are UNIX-platform-based, and Maxplus2 is PC-based.

I developed a top-level test-bench schematic in Powerview containing three primary sections: an input-data generator symbol, an image reconstruction board symbol, and an output-data reading circuit. The input-data generator simulates the four A/D input-data channels, as well as the pixel clock, and the data-valid and frame-valid signal inputs from the CCD camera. The image reconstruction board symbol contains underlying schematics that provide PCB design information to Scicards and digital simulation information to the Viewlogic-Vantage digital simulator. The output-data reading circuit mimics the Dynetics high-speed post-processor interface that reads the output data from the image reconstruction board. To bring the test-bench schematic up in Powerview, set the project to:

```
/usr/users/geni2_users/dan/mrsr/imreco
```

The library path required:

```
/usr/users/geni2_users/dan/mrsr/imreco  
/cadbackup/powerview5.3/lib/alt_max2  
/cadbackup/powerview5.3/lib/max2sim  
/cadbackup/powerview5.3/lib/synlib  
/cadbackup/powerview5.3/lib/dio  
/wvlib/builtin  
/wvlib/74als  
/wvlib/74as  
/wvlib/74f
```

Use the Viewdraw icon to open the Viewdraw schematic (tbench_reco.1). For the digital simulation to work properly, the schematic must be brought up on a SUN computer; the Vantage simulator is not supported on the Dec-Station.

The test bench is driven by the Viewsim command file (tbench.cmd). The command file initializes and drives the circuit for simulation; it also generates Viewtrace windows used to view timing waveforms. The file is divided into four sections: vector declarations, vector assignment, Viewtrace plot creation, and initialization. The first three sections set up signals for Viewtrace; the fourth section starts the simulator and cycles the board into a known state. The file evolved over the development cycle of the board.

I built 12 separate Viewtrace windows to debug various sections of the board. Sections of the code that are no longer used are commented-out. In its current state, the code initializes the board with a very short frame-

valid cycle, followed by a frame-valid cycle that simulates the MRSR system in short-dwell mode. The code can be configured to produce various frame- and data-valid sizes by assigning values to `dv_on`, `dv_off`, `fv_on`, and `fv_off`. Unwanted Viewtrace windows can be eliminated by commenting-out the wave command that generates the window. See appendix C, Viewsim Command Files.

The input data generator symbol (`icon_reco.1`) is driven by the VHDL file `icon_reco.vhd`; the port statement of the VHDL file matches the pin-out of the device. The code has three sections: data-valid generator, frame-valid generator, and output generator. The data-valid duty cycle is controlled by the values of `dv_on` and `dv_off`; the data-valid signal is low for $(dv_on + 1)$ clock cycles and high for $(dv_off + 1)$ clock cycles. The frame-valid signal is low for $(fv_on + 1)$ data-valid cycles and high for $(fv_off + 1)$ data-valid cycles. The output generator generates four 16-bit pixel output channels. The lower 14 bits of each output are a ramp function, with values from 0 to $(dv_on + 1) \times (fv_on + 1) - 1$. The ramp value starts at 0 at the beginning of a frame and increments every clock cycle for which frame- and data-valid signals are both true. The upper two bits of each output channel represent the channel number, 0 through 3. The VHDL code was compiled on the Viewlogic Vantage VHDL compiler. See appendix C, Viewsim Command Files.

The schematics that drive the image reconstruction symbol (`im_reco.1` through `im_reco.7`) contain the information required to simulate the board. Each component model used in the schematics contains package and signal information used by the PCB design software, and digital behavioral models used by the Vantage simulator. Some of the component models used are from the Viewlogic components libraries, using built-in models and VHDL for simulation behavior and package and signal attributes for PCB design. The schematics `im_reco.1` through `im_reco.7` are equivalent to the schematics `ir01.1` through `ir10.1` listed in appendix A. The original seven c-size schematics were refitted into 10 a-size sheets so they could be included as an appendix. (The a-size schematics will not work for simulation or PCB export.)

The two FPGAs used in this project were written in VHDL and compiled and simulated using the Altera Maxplus2 VHDL compiler. VHDL written for the Maxplus2 compiler is not fully compatible with the Viewlogic Vantage VHDL compiler. To simulate the performance of the FPGAs in the board-level simulation, an electronic design interface format (EDIF) file is exported from the Maxplus2 simulation (the EDIF file is used to create a Viewsim-compatible wir file for simulation). To accomplish this, select the export EDIF 2.0 option in the Maxplus2 compiler, then select the Viewlogic option. The compiler will produce an `.edo` file. Copy the `.edo` file to the Viewlogic project directory, then convert (UNIX `mv`) the `.edo` file to an `.edn` file. To convert the `.edn` file to a Viewlogic wir file run (the EDIF netlist-in-icon from the Powerview cockpit), use the default options given.

Next, go to the wir subdirectory to check for errors. (The wir file created will probably have errors, depending on the Powerview version number.) To fix errors, edit the file and delete the following lines:

```
P IN VDD
I VDD IN VDD
P IN GND
I GND IN GND
```

then add:

```
G VDD
G GND
```

This change replaces power and ground input pins with global variables. At this point, the FPGA wir file is ready for simulation.

The Viewgen icon can be used to generate schematics from the wir file. The schematics can be marginally useful in debug; the main problem is that signals in the schematics are hard to track, since there is not a one-to-one map from VHDL signals to schematic signals.

To design the PCB, my colleagues and I converted the Powerview output to an input data file for the Scicards PCB design software. Within Scicards, we defined a board, placed components, and routed signals. We sent the output from Scicards to a PCB manufacturer and used the to-Scicards icon in the Powerview cockpit to generate an output file from Powerview that was read by the Scicards PCB design software, generating a .UPL file. We created a board in Scicards; read-in the .UPL file, and placed the components on the board. The board we created was a standard VME double-height board (6.229 by 9.187 in.), with no P1 or P2 connectors. We routed the board with the freestyle router, and made Gerber plot files, which were used to manufacture the PCB from the routed board.

3.5 I-O Specifications

Figures 6 and 7 illustrate I/O data-timing waveforms for a fictitious CCD frame size of 4 by 2 (four data-valid lines where each data-valid line is two pixels long). (I chose the sizes for illustration purposes only.) The CCD pixel clock (inclk) rate is 13.767 MHz. The output-data clock (c0clk, c1clk, c2clk, and c3clk) rate is 10 MHz. I chose these clock rates to represent the MRSR system clock rates. In the MRSR system, the data-valid lines are true for 64 pixels, and false for 5 to 10 pixels, depending on the CCD camera configuration. There are 128 data-valid lines per frame-valid in long-dwell mode.

As shown in figure 5, pixel clock is a rising edge clock, data- and frame-valid (indv and infv) are low-true signals that qualify the pixel clock. The four input-pixel-data channels (ch0in, ch1in, ch2in, and ch3in) show the 14-bit ramp function with 2-bit channel address generated by icon_reco.vhd.

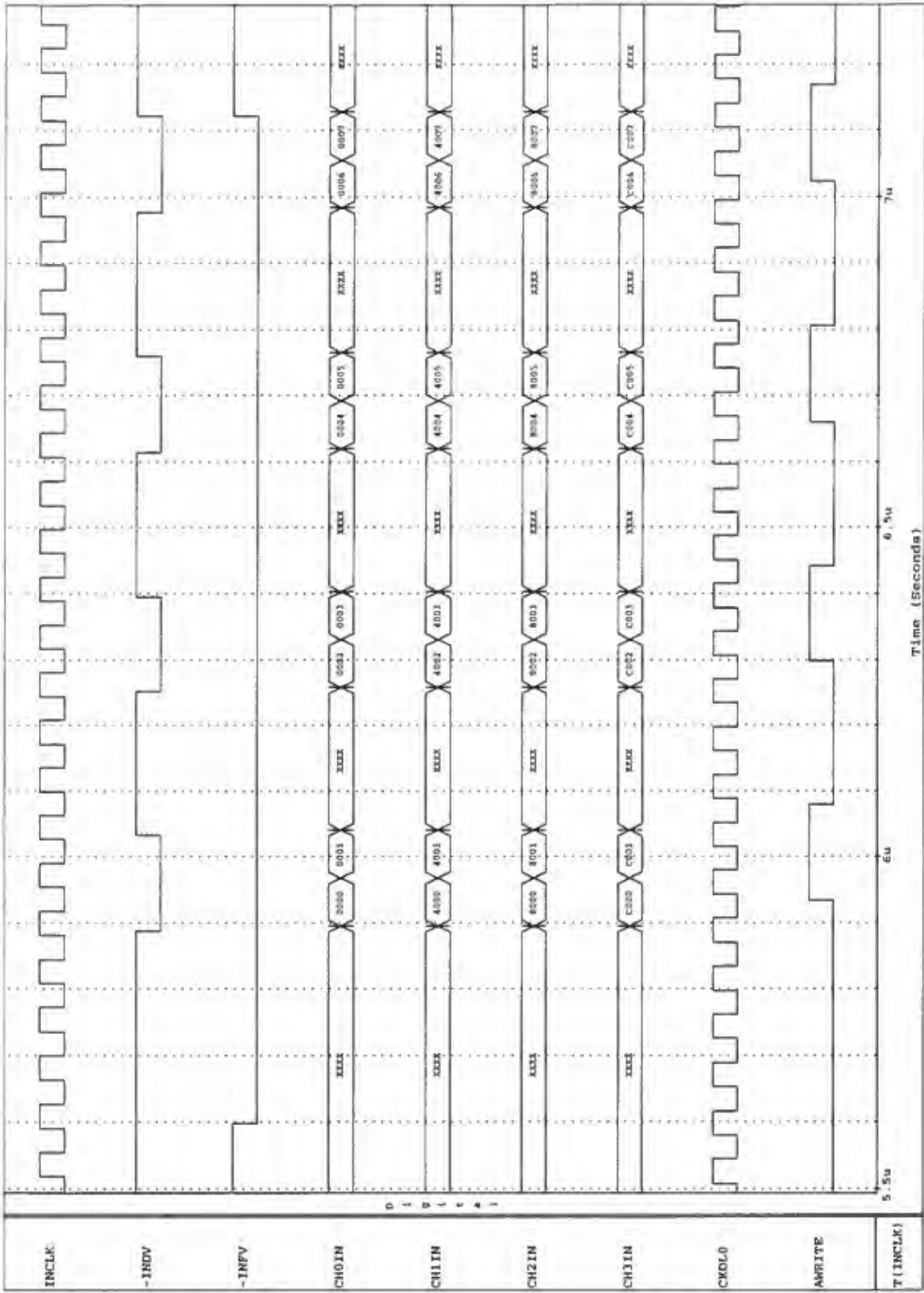


Figure 6. Input data timing waveforms.

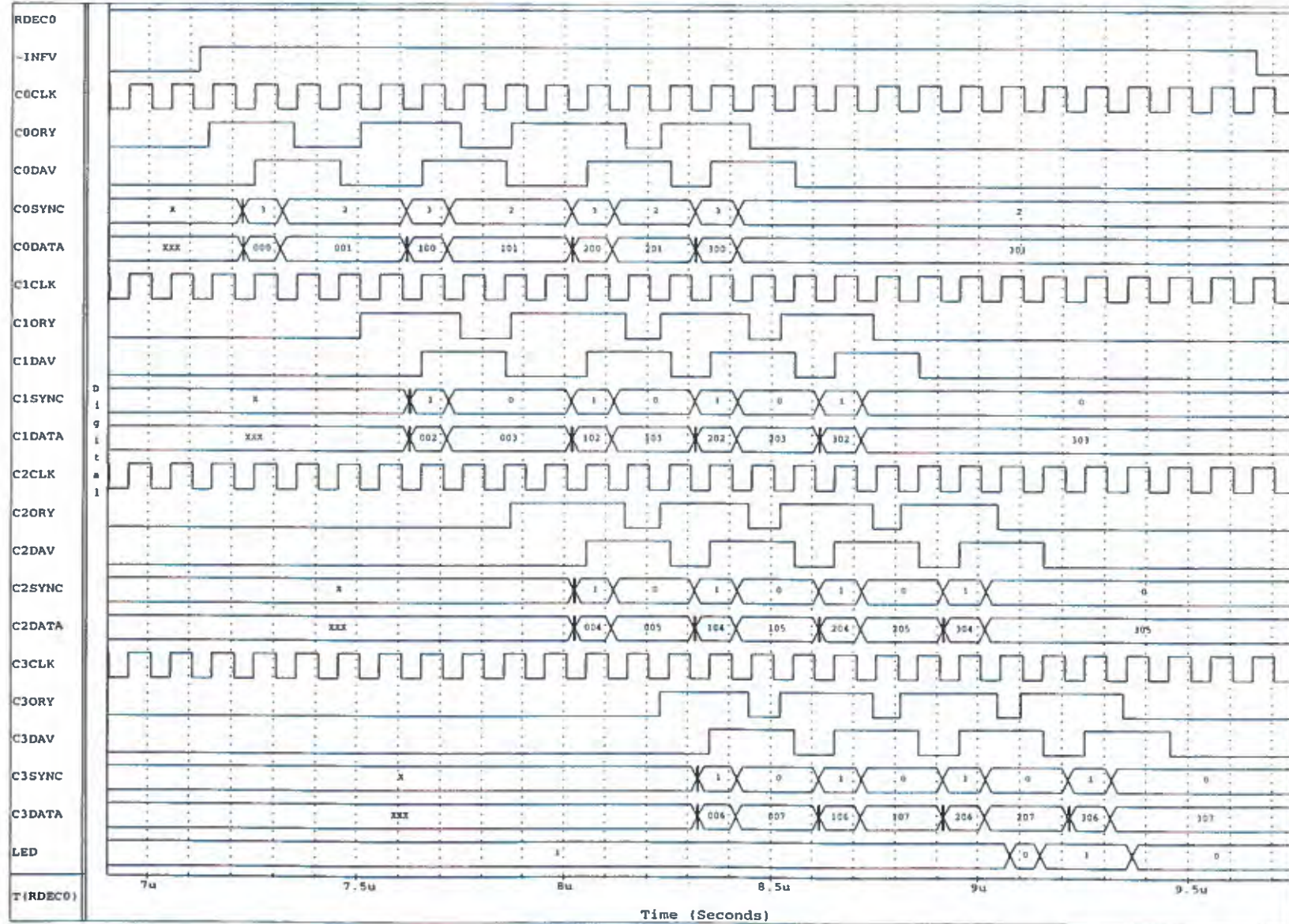


Figure 7. Output data timing waveforms.

The relative timing between the rising edge of pixel clock (indv, inv) and the input pixel data is unspecified. The programmable jumpers on the differential receivers allow the board to adjust to various timings and input clock rates. In this case, the delay for indv and inv is 0 ns, and the delay for inclk is 30 ns (refer to appendix C, Viewsim Command Files). The signals ckd10 and awrite show how the delayed input control signals are combined to latch the input-data and write it into the input-fifo bank.

Figure 7 shows the output timing waveforms for reading-out the fictitious input-data frame. The output clock, output-ready flag, and output-data-available flag, along with the data and synchronization bits that are sent to the Dynetics processor, are shown for each of the four output channels. In this simulation, the read-data-enable signals from the Dynetics processor for each channel are always true. The synchronization bits are in test-mode format.

In this simulation, the input-data frame ends before the output circuit has even begun to read data because the latency through the frame buffer board pipeline is longer than the frame. The simulation shows the latency to be 19 clock cycles—this is an error. The latency for the hardware is actually 30 clock cycles. The difference occurred because the fifo model used in the simulation does not model the first-word to empty-flag latency in the fifo. The simulation fifo sets the empty flag after 1-clock; the actual fifo takes 12 clock cycles.

In the simulation, the LED becomes a 0 for one clock cycle, goes back to a 1 for two clock cycles, and back to a 0 when the entire frame has been transferred to the Dynetics post-processor because, in this case, the input aggregate clock rate is slower than the output clock rate. The input clock rate is 13.767 MHz; the data-valid is true for two pixels and false for three pixels. It takes 17 clock cycles to shift-in 8 pixels, giving an input aggregate clock rate of 6.479 MHz. The output shift-out rate is 10 MHz. This occurs because of the small size of the simulated frame, and is not the case for an actual frame of MRSR data.

4. Conclusions and Current Status

I made a wire-wrap test board to test the image reconstruction board. The wire wrap board generates the input signals required to drive the image reconstruction board. The signals generated are identical to the signals generated by the input-data generator symbol in the Powerview simulation. The test board can be dip-switch configured to produce various sizes of frame-valid and data-valid signals, as in the simulation. The test board has an analog mezzanine board with a digital-to-analog converter and four analog output drivers. The mezzanine card can drive a 4-channel A/D board. For testing, the board can be used to drive the image reconstruction board directly, or with a 4-channel A/D board.

I have not yet integrated the image reconstruction board and A/D board with the Dynetics high-speed post-processor. To read the data out of the image reconstruction board, I made a differential receiver board. I connected this board to a HP logic analyzer, and used the wire-wrap test board to drive the input. In this test setup, the wire-wrap test board and the differential receiver board run the board as it is designed to run in the MRSR system. The HP logic analyzer is used to examine the output data. The drawback to the test setup is that only a single output channel can be examined at a time by the HP logic analyzer; a ribbon cable must be swapped each time a different channel is to be examined.

I tested the board in the test set-up using a 13.7 MHz input pixel clock for both long- and short-dwell modes, with a read-out clock rate of 10 MHz. Input and output clock frequencies were varied over frequencies up to 25 MHz. (The board works only for frequencies up to 25 MHz.)

Simulation results agreed very closely with the test results. The only point that I missed was the first-word to empty-flag latency in the fifo—I did not read the fifos preliminary data sheets closely when I built the fifo simulation model. I discovered the first-word latency during the hardware test, and modified the VHDL controller code to accommodate the fifo latency. I have not updated the fifo model, so there is an 11-clock difference between the fifo model and the actual fifo.

Use of the simulator allowed me to build the final multilayer PCB with no hardware brass board. There was enough extra room in the controller FPGA to fix the fifo latency error. The final controller FPGA was 85-percent filled. Only one jumper wire was required on the board to overcome a bad I/O assignment on the controller.

I have built and tested three image reconstruction boards. At this point they have not been integrated with either the A/D board or the Dynetics post-processor, since neither has been completed. The wire-wrap test board can be used to generate a test pattern to integrate and test the image reconstruction board with the Dynetics system. The analog mezzanine board can then be used to drive an A/D board, facilitating the integration of an A/D-image reconstruction board set with the Dynetics system. The next step is to remove the wire wrap test board and integrate the board set into the ARL CCD array interface.

References

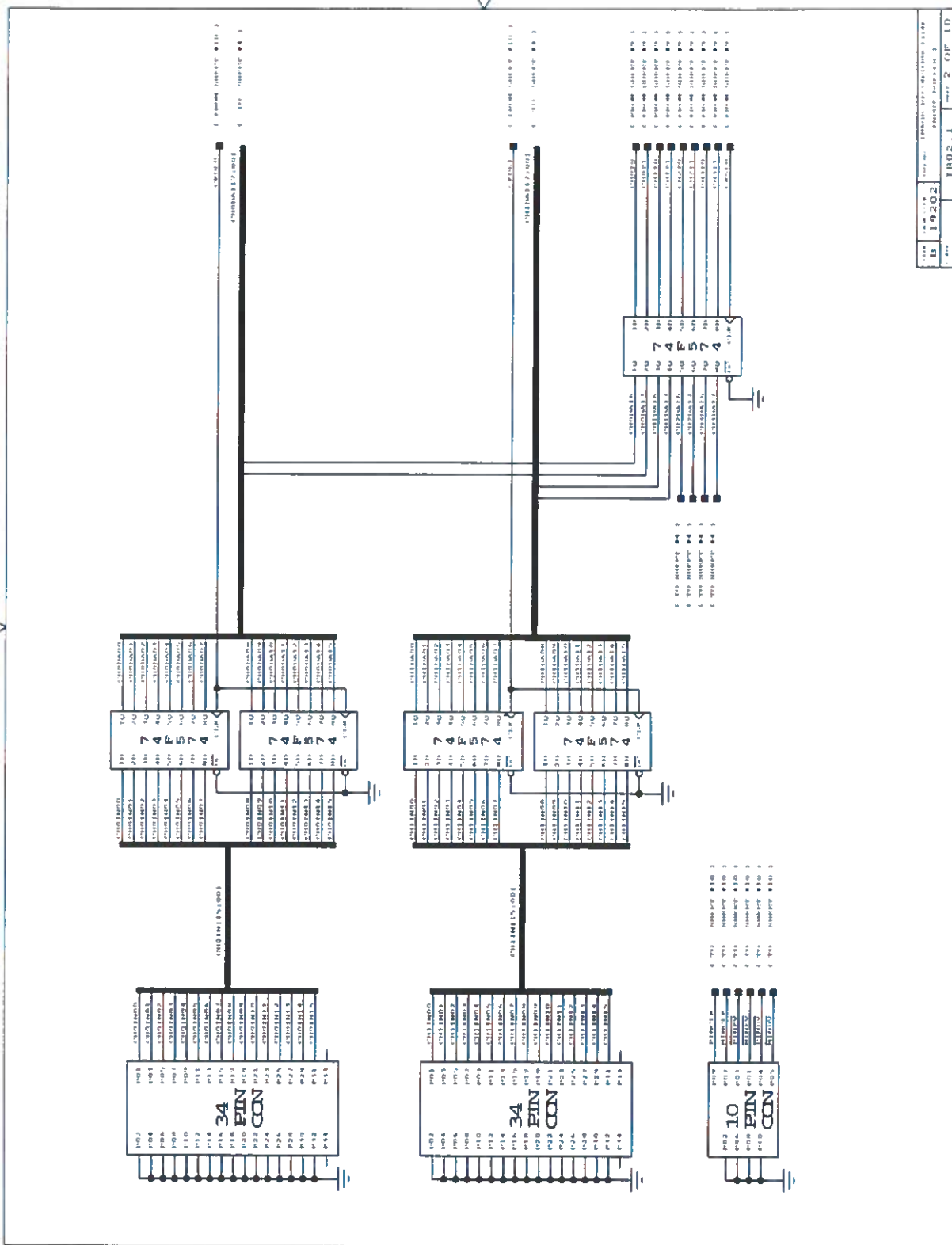
1. R. Durrett, R. Dean (Dynerics Inc.), D. F. McCarthy, and E. A. Viveiros (Army Research Laboratory), *Interface and post processing requirements to insert an acousto-optic range-doppler processor into an advanced radar digital signal processor*, Technical Conference SPIE Proceedings, Vol. 2489, Transition of Optical Processors into Systems (1995).
2. E. A. Viveiros, L. J. Harrison, M. S. Patterson, R. J. Berinato, K. W. Williams, V. R. Riasati, and R. A. Durrett, *Acousto-Optic Range-Doppler Processor Design for Radar Insertion*, SPIE Vol. 1704: Advances in Optical Information Processing V (1992).
3. P. A. Levine, D. J. Sauer, F. L. Hsueh, F. V. Shallcross, G. M. Meray, G. C. Taylor, G. W. Huges, J. Pellegrino, D. Simon, L. Harrison, and W. Lawler, *Performance of a high frame-rate CCD imager*, SPIE Vol. 1693: Surveillance Technologies II (1992).
4. D. J. Sauer, F. L. Hsueh, F. V. Shallcross, G. M. Meray, P. A. Levine, G. W. Huges, and J. Pellegrino, *High Fill-Factor CCD Imager with High Frame-Rate Readout*, SPIE Vol. 1291: Optical and Digital GaAs Technologies for Signal Processing Applications (1990).
5. Mark M. Giza, Cary L. Bright, Michael S. Patterson, and William B. Lawler, *High-Performance 2-D CCD Imager Module*, ICSPAT 1993 Conference Proceedings on Signal Processing Applications and Technology.
6. Integrated Device Technology, Inc., *CMOS Supersync fifo IDT72255-65 Preliminary Data Sheet* (October 1994).
7. Altera, *Data Book* (1995).

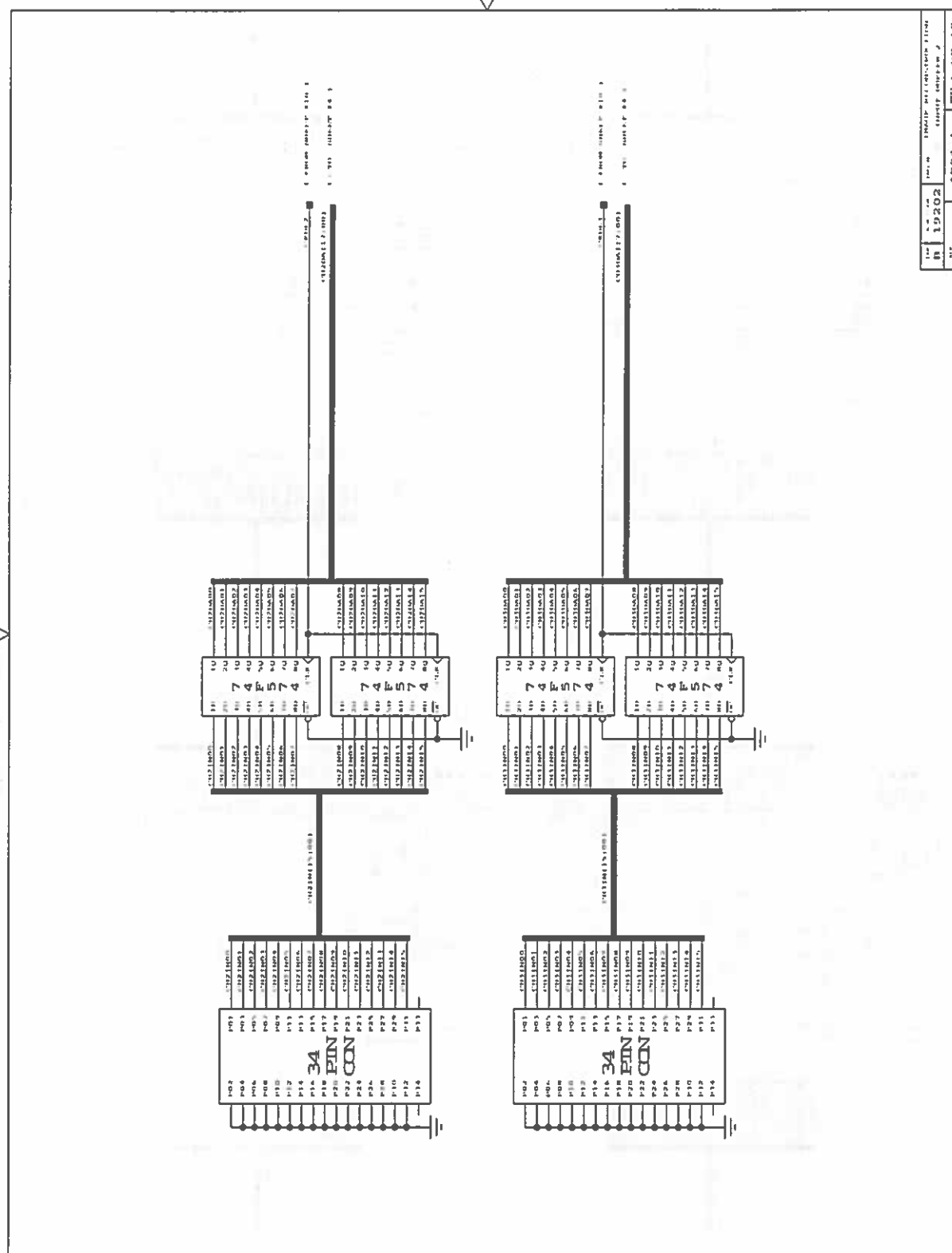
Acknowledgments

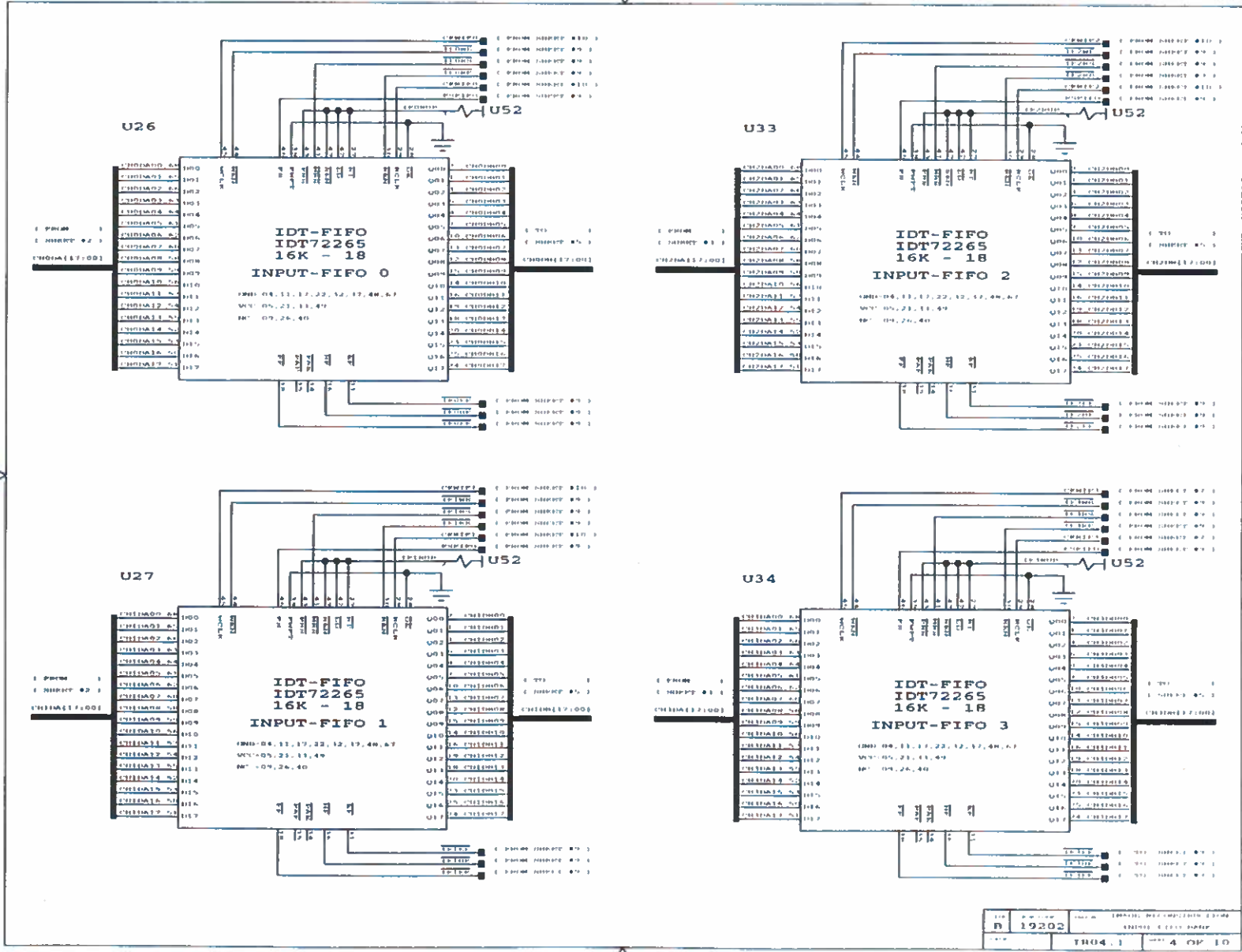
I would like to thank the following people for their help in building this board: Debbie Simon and Harvey Gearhart for their work on the printed circuit board design; Gail Koebke for building the printed circuit boards; Kareem Darwish for Altera and VHDL work; and Mike Younger, Steve Choy, Martha Givan, and Greg Turner for computer system support.

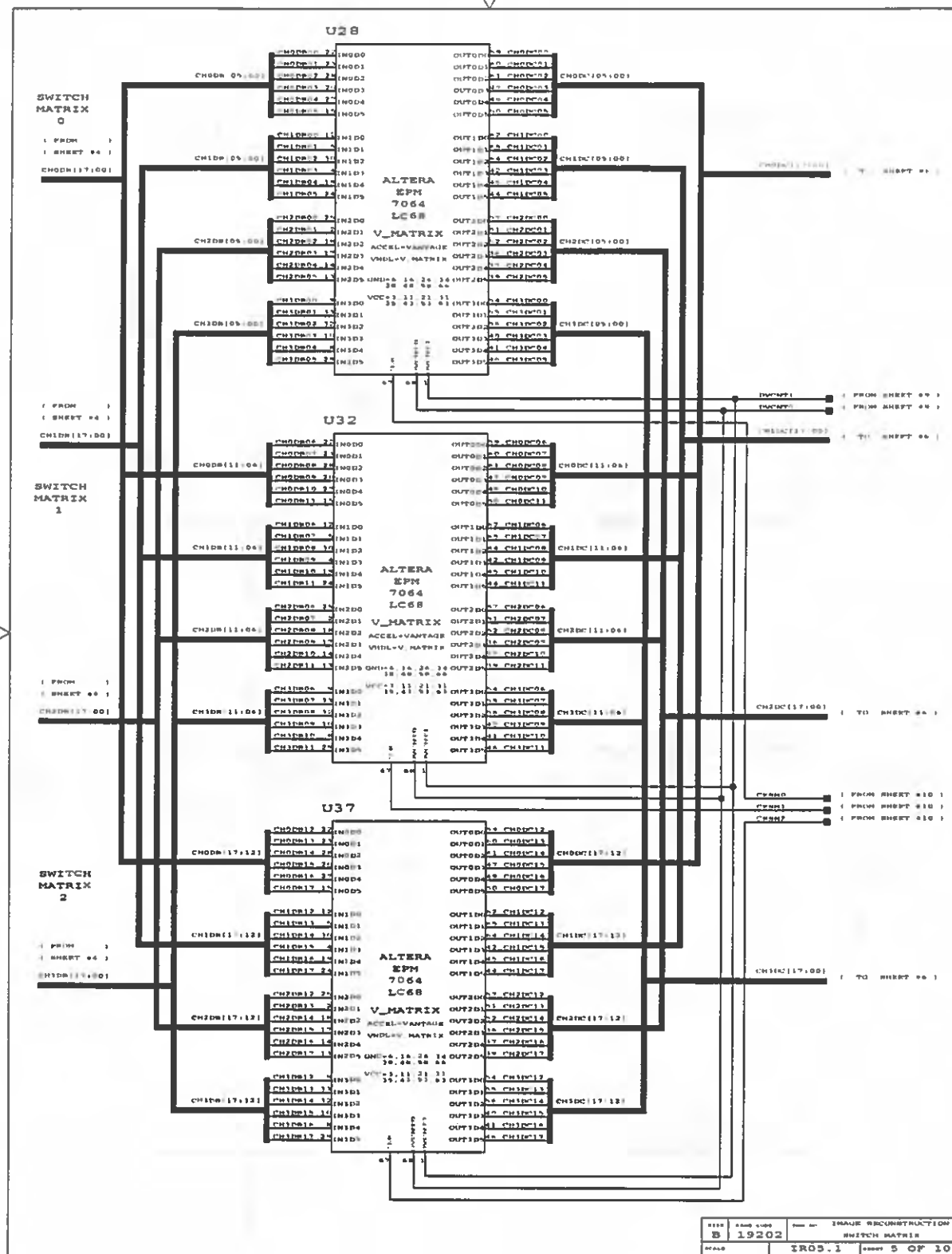
Appendix A.—Image Reconstruction Electrical Schematics

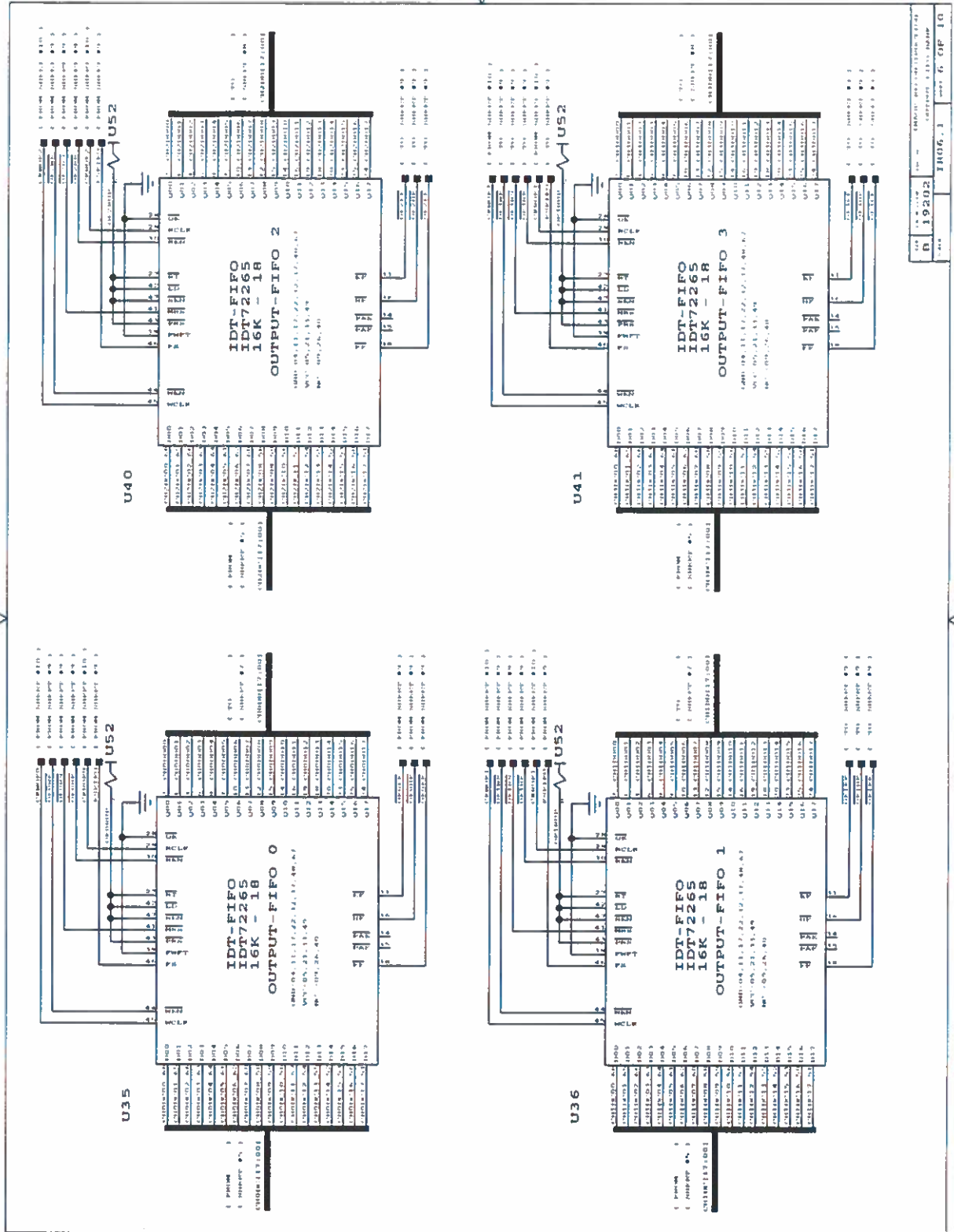
Schematic	Postscript	Schematic name	Sheet
ir01.1	ir01.ps	Image Reconstruction Block Diagram	1 of 10
ir02.1	ir02.ps	Image Reconstruction Input Buffer 1	2 of 10
ir03.1	ir03.ps	Image Reconstruction Input Buffer 2	3 of 10
ir04.1	ir04.ps	Image Reconstruction Input Fifo Bank	4 of 10
ir05.1	ir05.ps	Image Reconstruction Switch Matrix	5 of 10
ir06.1	ir06.ps	Image Reconstruction Output Fifo Bank	6 of 10
ir07.1	ir07.ps	Image Reconstruction Output Buffer 1	7 of 10
ir08.1	ir08.ps	Image Reconstruction Output Buffer 2	8 of 10
ir09.1	ir09.ps	Image Reconstruction Controller	9 of 10
ir10.1	ir10.ps	Image Reconstruction Clock Generator	10 of 10
irbp.1	irbp.ps	Image Reconstruction Bypass Capacitor Module	1 of 1
irs01.1	irs01.ps	Image Reconstruction Symbols Connectors	1 of 8
irs02.1	irs01.ps	Image Reconstruction Symbols DIP ICs	2 of 8
irs03.1	irs01.ps	Image Reconstruction Symbols IDT Fifo Symbol	3 of 8
irs04.1	irs01.ps	Image Reconstruction Symbols IDT Fifo Pinout	4 of 8
irs05.1	irs01.ps	Image Reconstruction Symbols Switch Matrix Symbol	5 of 8
irs06.1	irs01.ps	Image Reconstruction Symbols Switch Matrix Pinout	6 of 8
irs07.1	irs01.ps	Image Reconstruction Symbols Controller Symbol	7 of 8
irs08.1	irs01.ps	Image Reconstruction Symbols Controller Pinout	8 of 8

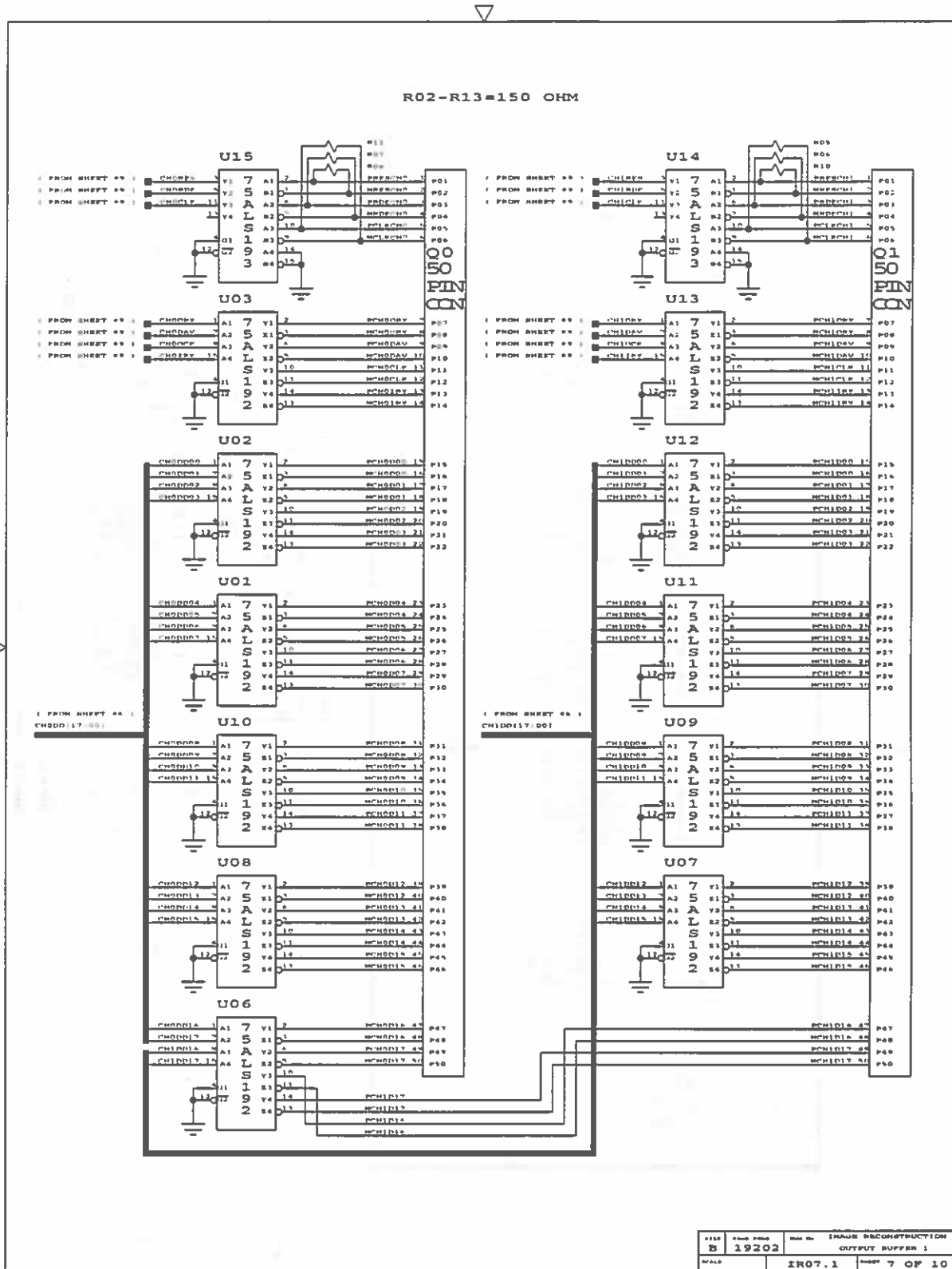


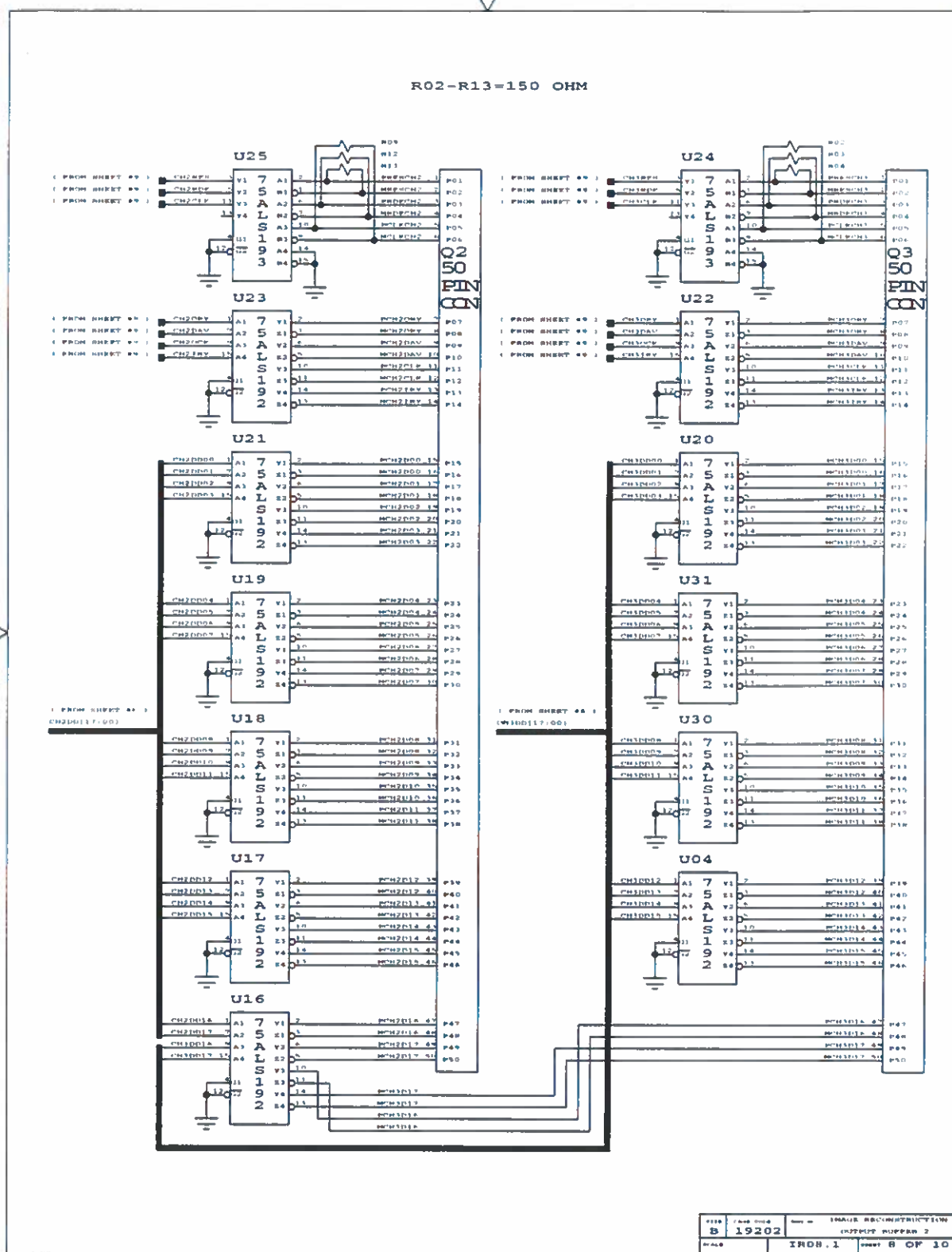


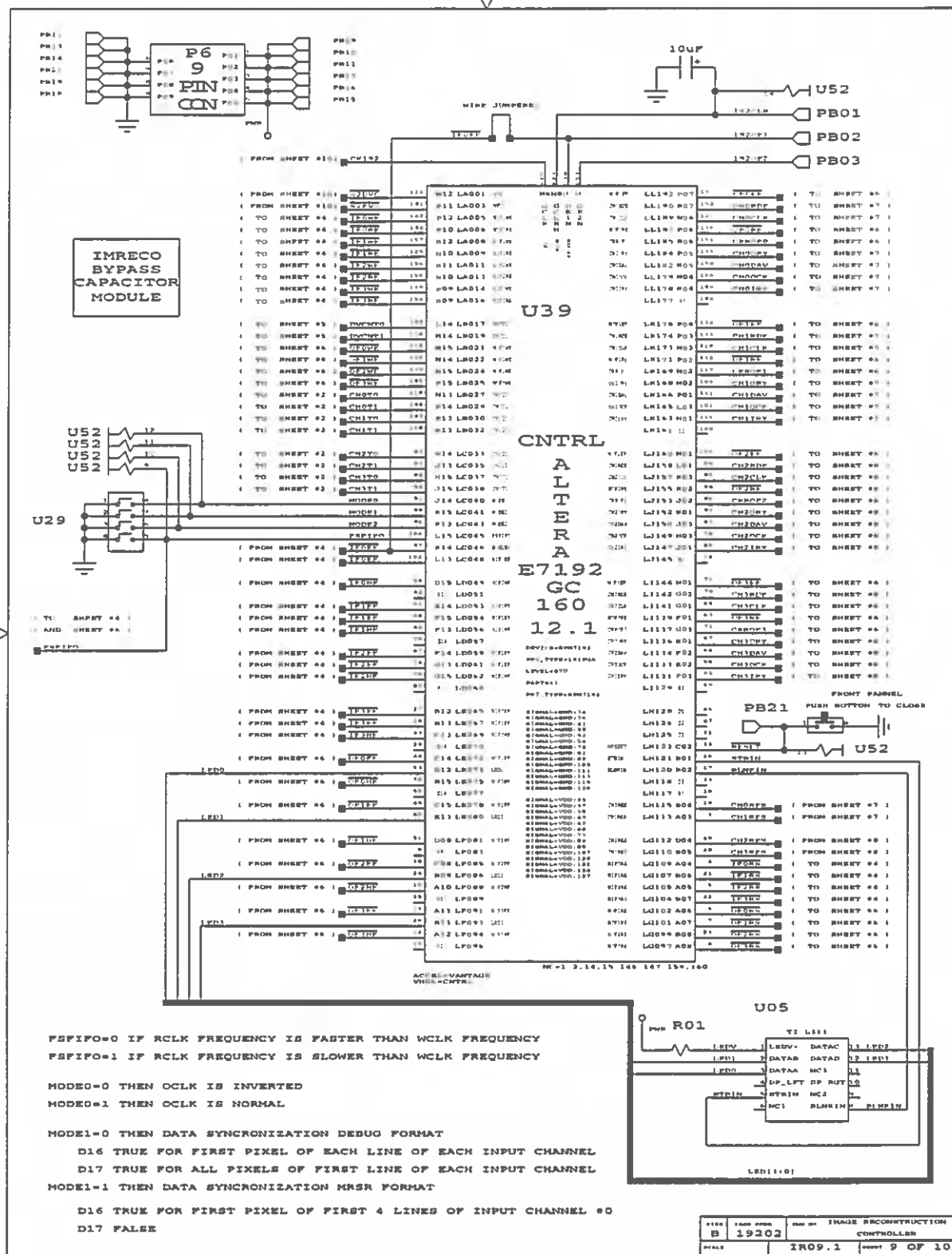




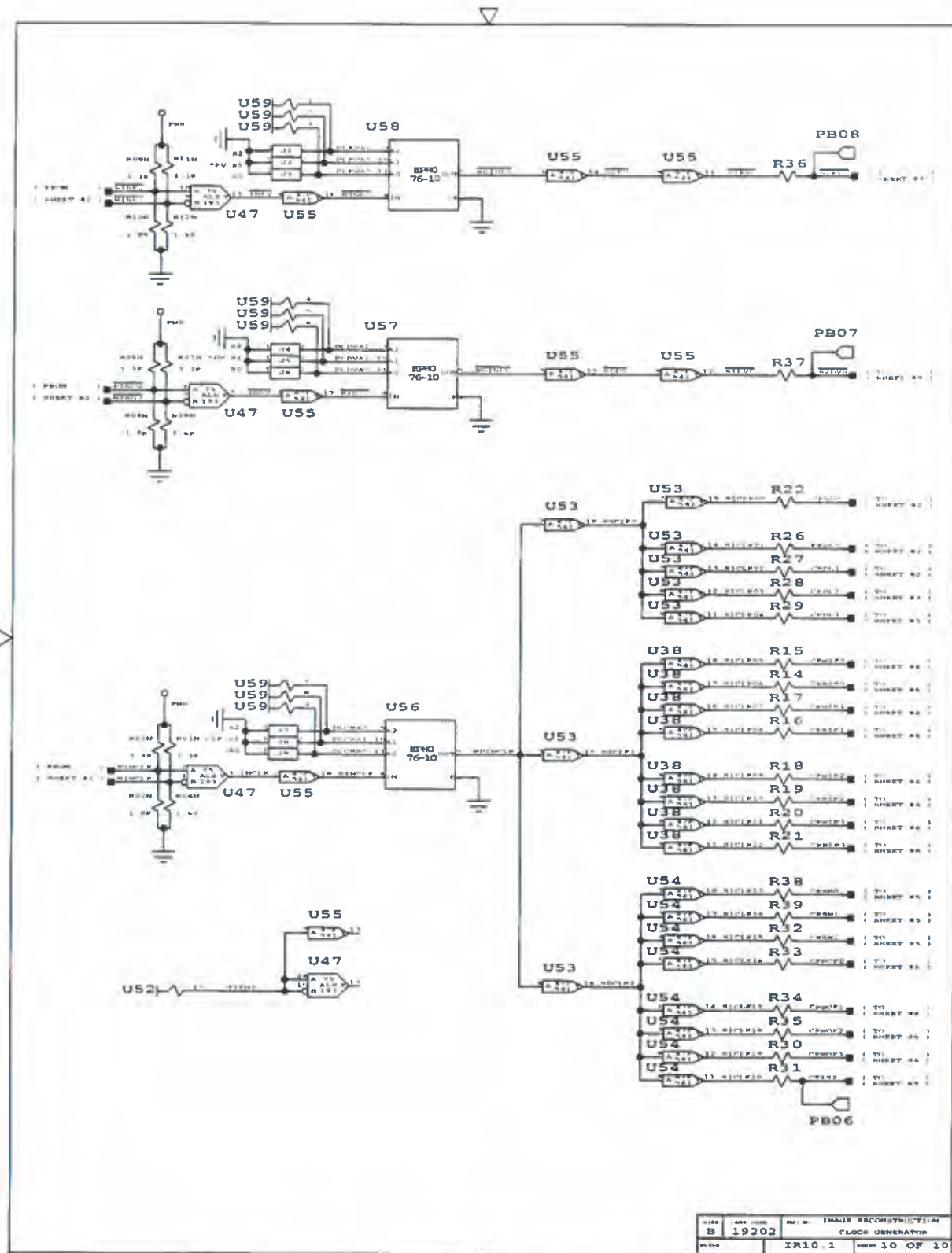


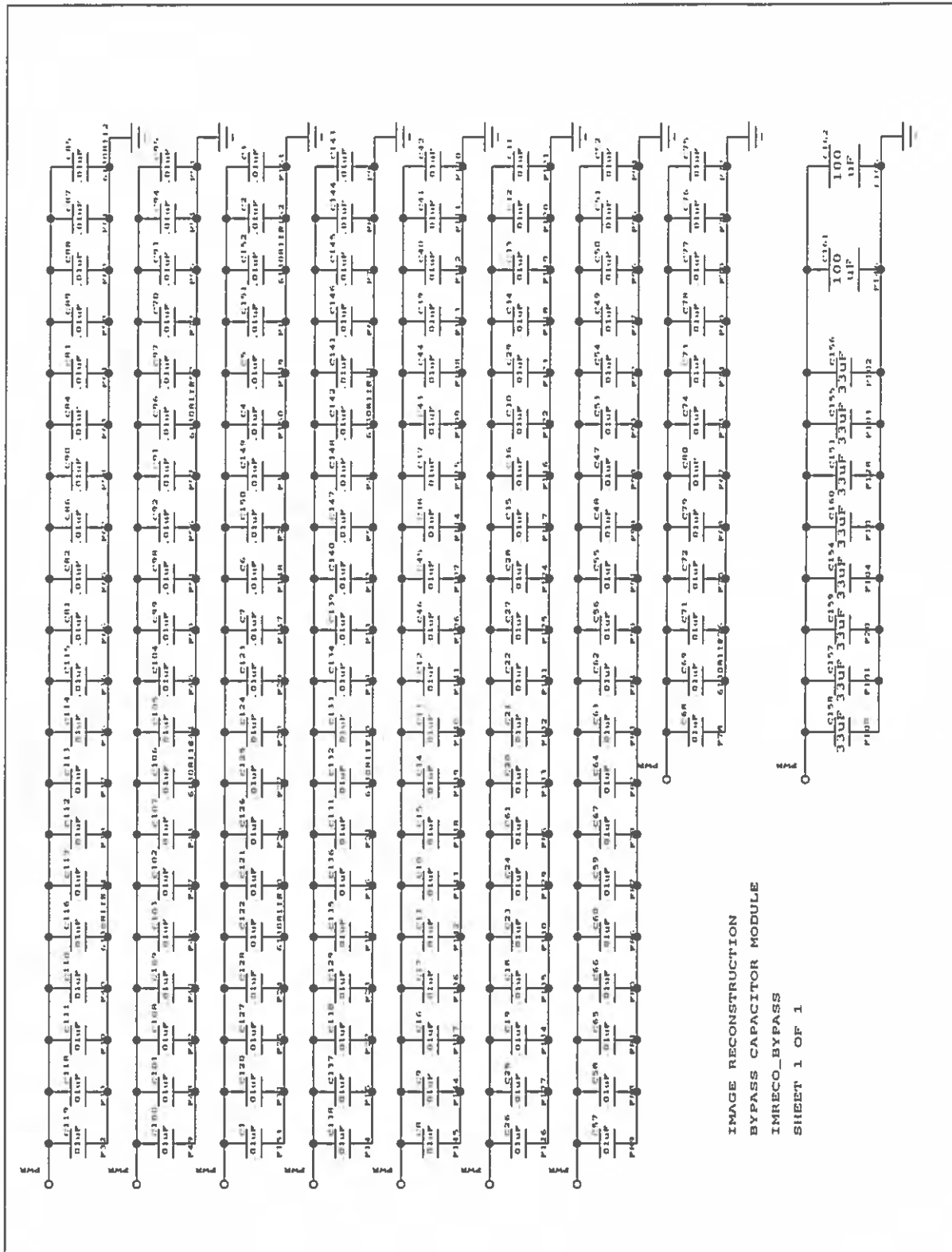


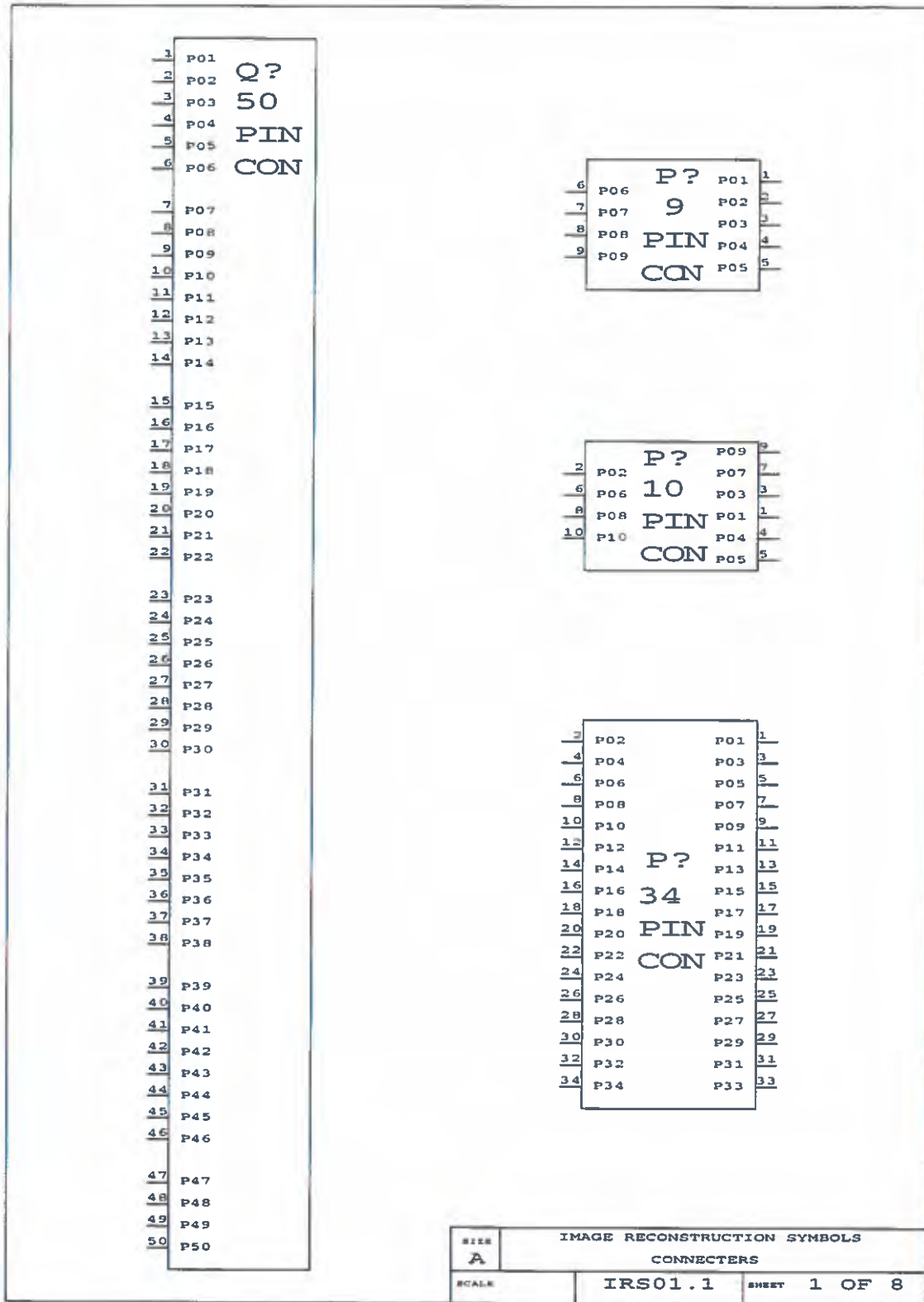




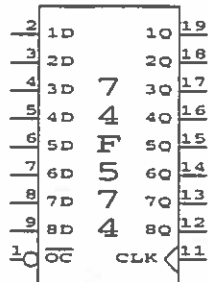
Appendix A



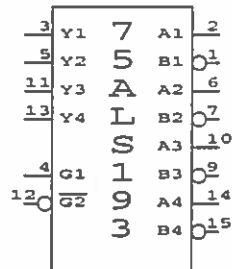




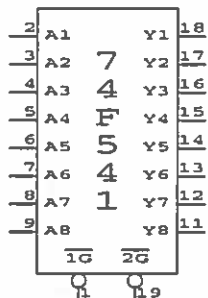
U?



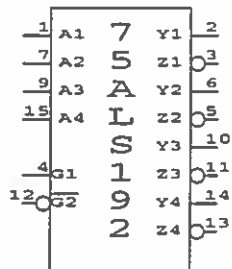
U?



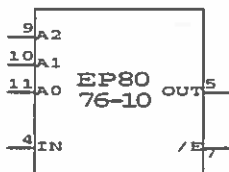
U?



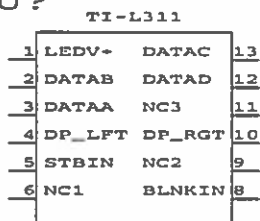
U?



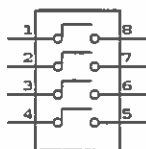
U?



U?

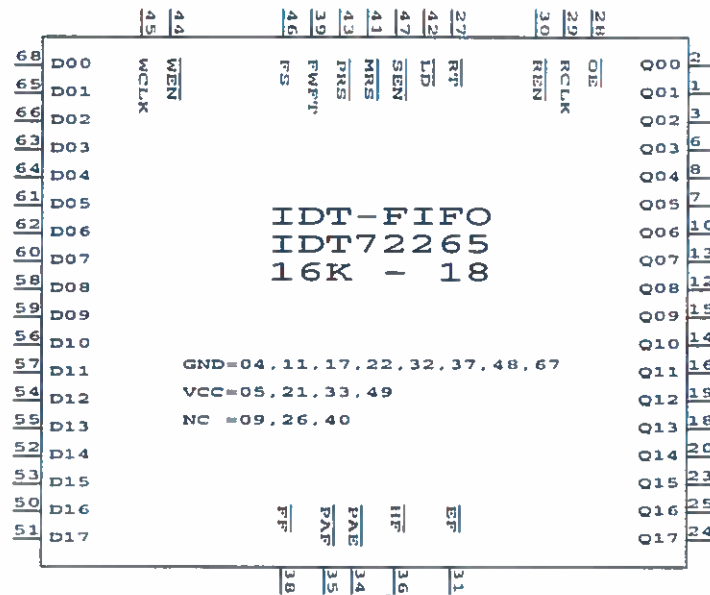


U?



SIZE	IMAGE RECONSTRUCTION SYMBOLS	
A	DIP IC'S	
SCALE	IRS02.1	SHEET 2 OF 8

U?



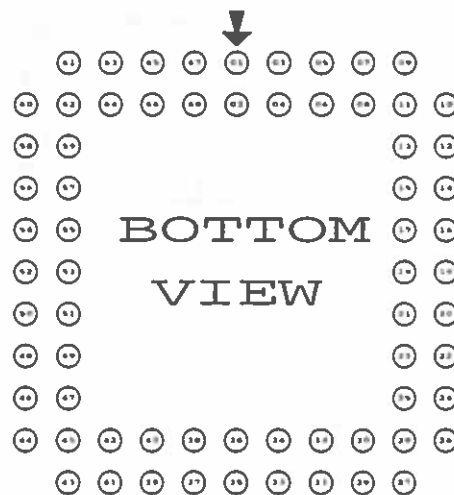
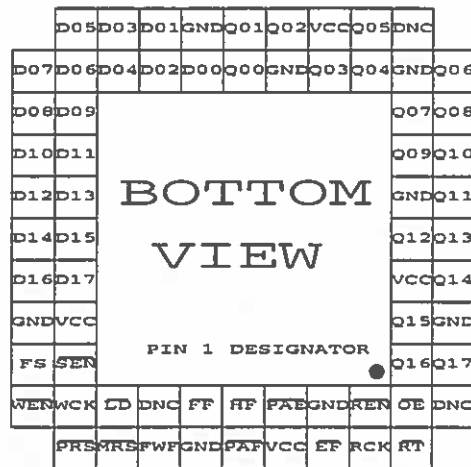
IDT-FIFO
IDT72265
16K - 18

GND=04, 11, 17, 22, 32, 37, 48, 67
VCC=05, 21, 33, 49
NC =09, 26, 40

SIZE	IMAGE RECONSTRUCTION SYMBOLS		
A	IDT-FIFO SYMBOL		
SCALE	IRS03.1	SHEET	3 OF 8

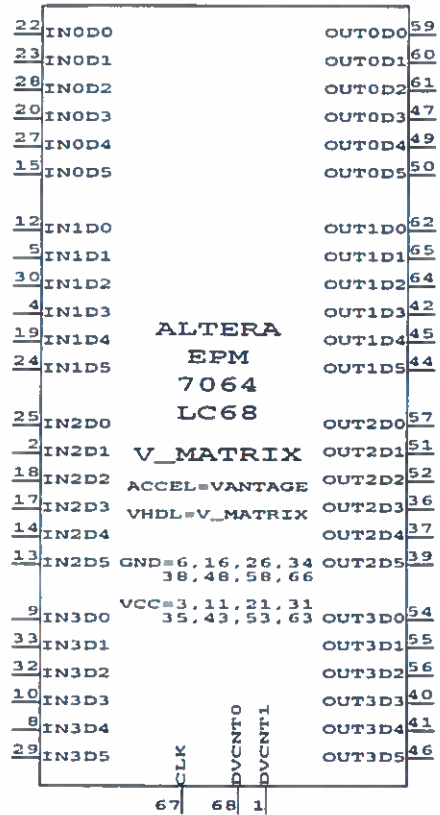
68 PIN-GRID-ARRAY IDT-72265-G68

Q01 01
Q00 02
Q02 03
GND 04
VCC 05
Q03 06
Q05 07
Q04 08
DNC 09
Q06 10
GND 11
Q08 12
Q07 13
Q10 14
Q09 15
Q11 16
GND 17
Q13 18
Q12 19
Q14 20
VCC 21
GND 22
Q15 23
Q17 24
Q16 25
DNC 26
RT 27
OE 28
RCK 29
REN 30
EF 31
GND 32
VCC 33
PAE 34
PAF 35
HF 36
GND 37
FF 38
FWF 39
DNC 40
MRS 41
LD 42
PRS 43
WEN 44
WCK 45
FS 46
SEN 47
GND 48
VCC 49
D16 50
D17 51
D14 52
D15 53
D12 54
D13 55
D10 56
D11 57
D08 58
D09 59
D07 60
D05 61
D06 62
D03 63
D04 64
D01 65
D02 66
GND 67
D00 68



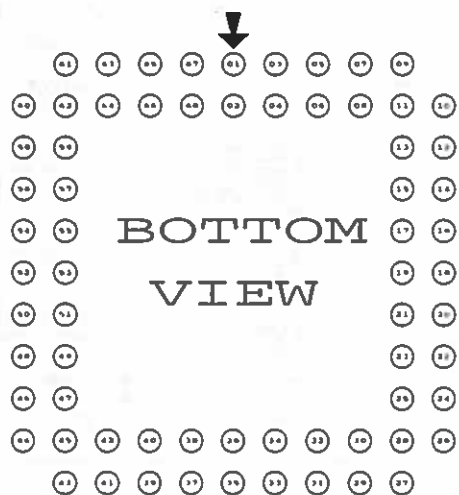
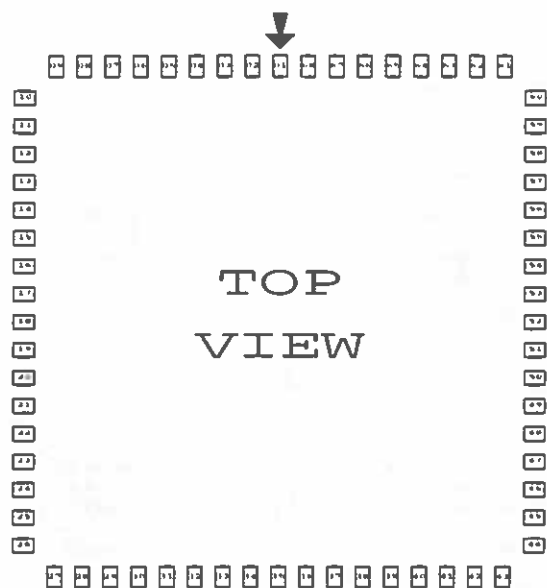
SIZE	IMAGE RECONSTRUCTION SYMBOLS		
A	IDT-FIFO PIN-OUT		
SCALE	IRS04.1	SHEET	4 OF 8

U?



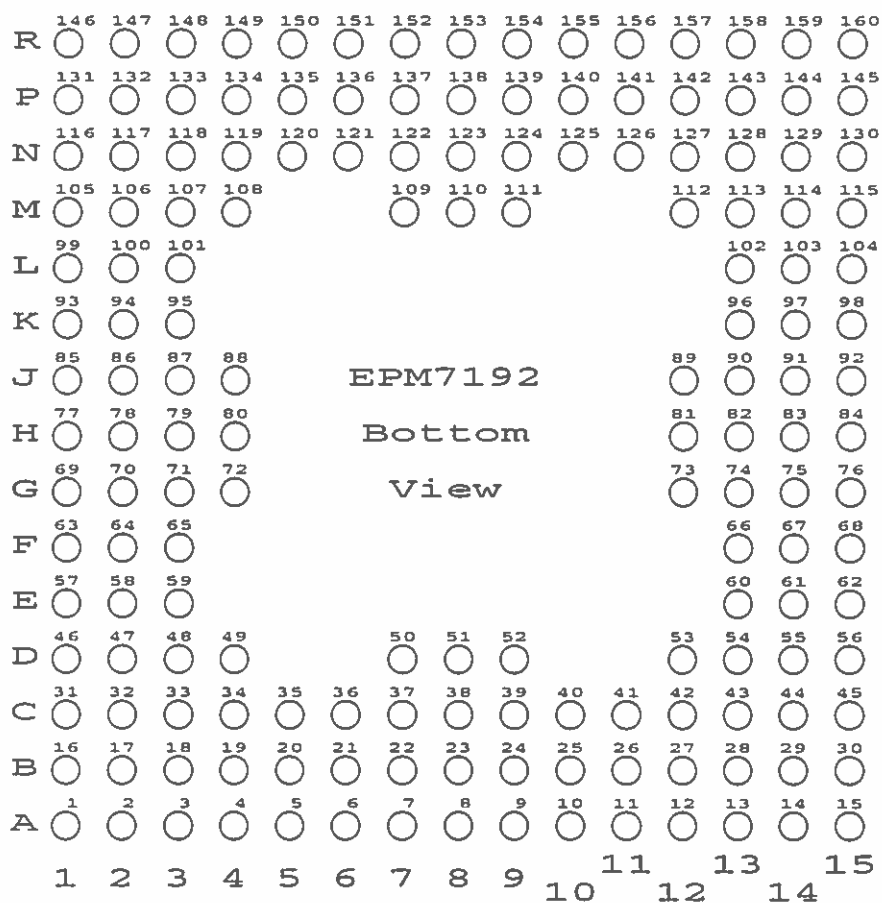
SIZE	IMAGE RECONSTRUCTION SYMBOLS		
A	SWITCH MATRIX SYMBOL		
SCALE	IRS05.1	SHEET	5 OF 8

68 PIN PLCC SOCKET



SIZE	IMAGE RECONSTRUCTION SYMBOLS	
A	SWITCH MATRIX PIN-OUT	
SCALE	IRS06.1	SHEET 6 OF 8

112	M12 LA001	NC	MINI	11	NC	LL192 P07	137
141	P11 LA003	NC	MINI	11	NC	LL190 R07	136
142	P12 LA005	NC	MINI	11	NC	LL189 R06	135
143	P10 LA006	NC	MINI	11	NC	LL187 P06	134
157	R12 LA008	NC	MINI	11	NC	LL185 R06	133
125	N10 LA009	NC	MINI	11	NC	LL184 P05	132
156	R11 LA011	NC	MINI	11	NC	LL182 R05	131
155	R10 LA013	NC	MINI	11	NC	LL179 R04	130
138	P09 LA014	NC	MINI	11	NC	LL178 R04	129
154	R09 LA016	NC	MINI	11	NC	LL177 R	128
U39							
103	L14 LB017	NC	MINI	11	NC	LK176 P04	134
118	M14 LB019	NC	MINI	11	NC	LK174 P03	133
113	M15 LB021	NC	MINI	11	NC	LK173 N03	132
129	N14 LB022	NC	MINI	11	NC	LK171 P02	131
130	N15 LB024	NC	MINI	11	NC	LK169 N02	130
145	P15 LB025	NC	MINI	11	NC	LK168 M02	129
146	N13 LB027	NC	MINI	11	NC	LK166 P01	128
144	P14 LB029	NC	MINI	11	NC	LK165 L03	127
143	P13 LB030	NC	MINI	11	NC	LK163 N01	126
150	R13 LB032	NC	MINI	11	NC	LK161 L	125
CNTRL							
83	K14 LC033	NC	MINI	11	NC	LJ160 M01	105
90	J13 LC035	NC	MINI	11	NC	LJ158 L01	99
94	K15 LC037	NC	MINI	11	NC	LJ157 K03	95
92	J15 LC038	NC	MINI	11	NC	LJ155 K02	94
91	J14 LC040	NC	MINI	11	NC	LJ153 J02	90
98	K15 LC041	NC	MINI	11	NC	LJ152 K01	83
96	K13 LC043	NC	MINI	11	NC	LJ150 J03	87
104	L15 LC045	NC	MINI	11	NC	LJ149 H03	78
97	K14 LC046	NC	MINI	11	NC	LJ147 J01	85
102	L13 LC048	NC	MINI	11	NC	LJ145 H	78
E7192							
94	D15 LD049	NC	MINI	11	NC	LJ144 H01	77
92	E11 LD051	NC	MINI	11	NC	LJ142 G02	70
91	K14 LD053	NC	MINI	11	NC	LJ141 G01	69
90	P15 LD054	NC	MINI	11	NC	LJ139 P01	63
86	P13 LD056	NC	MINI	11	NC	LJ137 G03	71
75	G14 LD057	NC	MINI	11	NC	LJ136 E01	51
61	P14 LD059	NC	MINI	11	NC	LJ134 P02	64
74	G13 LD061	NC	MINI	11	NC	LJ133 E02	56
74	G15 LD062	NC	MINI	11	NC	LJ131 P03	65
82	H11 LD064	NC	MINI	11	NC	LJ129 E	59
GC							
29	M12 LE065	NC	MINI	11	NC	LH128 C	46
28	D13 LE067	NC	MINI	11	NC	LH126 C	47
93	C13 LE069	NC	MINI	11	NC	LH125 C	31
26	H14 LE070	NC	MINI	11	NC	LH123 C03	32
44	C14 LE072	NC	MINI	11	NC	LH121 B01	16
53	D12 LE073	NC	MINI	11	NC	LH120 B02	17
30	H15 LE075	NC	MINI	11	NC	LH118 C	33
55	G14 LE077	NC	MINI	11	NC	LH117 B	18
45	C15 LE078	NC	MINI	11	NC	LH115 B04	19
60	E13 LE080	NC	MINI	11	NC	LH113 A03	1
51	D08 LP081	NC	MINI	11	NC	LQ112 D04	49
9	A8 LP083	NC	MINI	11	NC	LQ110 D05	20
38	C08 LP085	NC	MINI	11	NC	LQ109 A04	4
26	D09 LP086	NC	MINI	11	NC	LQ107 B06	21
10	A10 LP088	NC	MINI	11	NC	LQ105 A05	5
35	H10 LP089	NC	MINI	11	NC	LQ104 B07	22
11	A11 LP091	NC	MINI	11	NC	LQ102 A06	6
24	R11 LP093	NC	MINI	11	NC	LQ101 A07	7
12	A12 LP094	NC	MINI	11	NC	LC099 D08	23
13	A13 LP096	NC	MINI	11	NC	LC097 A08	8
NC=1, 2, 14, 15, 146, 147, 159, 160							
ACCEL=VANTAGE							
VDEL=CNTRL							
IMAGE RECONSTRUCTION SYMBOLS							
CONTROLLER SYMBOL							
IRS07.1							
7 OF 8							



SIZE	IMAGE RECONSTRUCTION SYMBOLS		
A	CONTROLLER PIN-OUT		
SCALE	IRS08.1	SHEET 8 OF 8	

Appendix B.—VHDL Files

B-1.— v_matrix.vhd

```

—
— FILENAME:  V_MATRIX.VHD
—
— FUNCTION:  PERFORMS PIPE-LINE MATRIX TRANSPOSE ON CCD OUTPUT DATA.
—            IMPLEMENTS A SWITCH MATRIX ON 4 CCD OUTPUT CHANNELS,
—            TRANSPOSES THE DATA INTO A FORMAT READABLE BY DYNETICS
—            MRSR FFT BOARD SET.
—
— VHD FILE:  sig1/dan/mrsr/altera/smatrx/v_matrix.vhd
—
— THIS CODE WAS ORIGINALLY WRITTEN BY KAREEM DARWISH ON JUNE 28,1994,
— FOR VIEWLOGIC VIEWSIM VHDL SIMULATION.
—
— MODIFIED BY DAN MCCARTHY ON AUG 24,1995, FOR ALTERA VHDL COMPILATION.
—
— THIS MODEL IS TARGETED FOR AN ALTERA EPM7064LC68-12 IC.

```

```

—
—
— INPUTS:  CLOCK          - CLK  —
—          DATA VALID COUNT - DVCNT[1:0] —
—          INPUT DATA      - IN0D[5:0], IN1D[5:0], IN2D[5:0], IN3D[5:0]
—          OUTPUTS:  OUTPUT DATA      - OUT0D[5:0], OUT1D[5:0], OUT2D[5:0], OUT3D[5:0]

```

```

—  ALGORITHM  —
—
— CASE 0
—   OUT0 <= IN0
—   OUT1 <= IN3
—   OUT2 <= IN2
—   OUT3 <= IN1
— CASE 1
—   OUT0 <= IN1
—   OUT1 <= IN0
—   OUT2 <= IN3
—   OUT3 <= IN2
— CASE 2
—   OUT0 <= IN2
—   OUT1 <= IN1
—   OUT2 <= IN0
—   OUT3 <= IN3
— CASE 3
—   OUT0 <= IN3
—   OUT1 <= IN2
—   OUT2 <= IN1
—   OUT3 <= IN0

```

Appendix B

```
— ENTITY v_matrix IS
  PORT( clk      : IN  BIT;
        dvcnt    : IN  INTEGER RANGE 0 TO 3;
        in0d     : IN  INTEGER RANGE 0 TO 63;
        in1d     : IN  INTEGER RANGE 0 TO 63;
        in2d     : IN  INTEGER RANGE 0 TO 63;
        in3d     : IN  INTEGER RANGE 0 TO 63;
        out0d    : OUT INTEGER RANGE 0 TO 63;
        out1d    : OUT INTEGER RANGE 0 TO 63;
        out2d    : OUT INTEGER RANGE 0 TO 63;
        out3d    : OUT INTEGER RANGE 0 TO 63);
END v_matrix;
ARCHITECTURE ARCHv_matrix OF v_matrix IS
BEGIN
  PROCESS( clk )
  BEGIN
    IF(clk'EVENT AND clk='1') THEN
      CASE dvcnt IS
        WHEN 0 =>
          out0d <= in0d;
          out1d <= in3d;
          out2d <= in2d;
          out3d <= in1d;
        WHEN 1 =>
          out0d <= in1d;
          out1d <= in0d;
          out2d <= in3d;
          out3d <= in2d;
        WHEN 2 =>
          out0d <= in2d;
          out1d <= in1d;
          out2d <= in0d;
          out3d <= in3d;
        WHEN OTHERS =>
          out0d <= in3d;
          out1d <= in2d;
          out2d <= in1d;
          out3d <= in0d;
      END CASE;
    END IF;
  END PROCESS;
END ARCHv_matrix;
```

Appendix B.—VHDL Files (cont'd)

B-2. —v_cntrl.vhd


```

- VIEWLOGIC FILE MODIFIED TO BE ALTERA COMPATIBLE,
- VIEWLOGIC COMPATIBILITY NOT MAINTAINED.
- THIS PORT STATEMENT DEFINES THE PINOUT OF AN ALTERA 7192GC160-12.1,
- U39 IMAGE RECONSTRUCTION BOARD SCHEMATIC SHEET# ( 6 OF 7 ).
- THIS FPGA IMPLEMENTS THE CONTROL LOGIC FOR
- THE IMAGE RECONSTRUCTION BOARD.

```

```

LIBRARY ieee;
USE ieee.std_logic_1164.all;

```

```

ENTITY cntrl IS

```

```

PORT(
NOF2HF: IN BIT;  - PIN A10  OUTPUT FIFO #2 HALF FULL
NOF3FF: IN BIT;  - PIN A11  OUTPUT FIFO #3 FULL FLAG
NOF3HF: IN BIT;  - PIN A12  OUTPUT FIFO #3 HALF FULL
-A13   : IN BIT;  - PIN A13
CH1RES: IN BIT;  - PIN A3   RESET CONTROL CHANNEL 1
NIF0RS: OUT BIT; - PIN A4   INPUT FIFO #0 RESET
NIF2RS: OUT BIT; - PIN A5   INPUT FIFO #2 RESET
NOF0RS: OUT BIT; - PIN A6   OUTPUT FIFO #0 RESET
NOF1RS: OUT BIT; - PIN A7   OUTPUT FIFO #1 RESET
NOF3RS: OUT BIT; - PIN A8   OUTPUT FIFO #3 RESET
-A9: IN BIT;  - PIN A9
STBIN  : OUT BIT; - PIN B1   7-SEG RISING EDGE STROBE
-B10   : IN BIT;  - PIN B10
LED3   : OUT BIT; - PIN B11  7-SEG OUTPUT DRIVE BIT 3
NIF3FF: IN BIT;  - PIN B12  INPUT FIFO #3 FULL FLAG
NIF3EF: IN BIT;  - PIN B13  INPUT FIFO #3 EMPTY FLAG
-B14   : IN BIT;  - PIN B14
NOF0HF: IN BIT;  - PIN B15  OUTPUT FIFO #0 HALF FULL
-BLNKIN: OUT BIT; - PIN B2   7-SEG OUTPUT BLANKING CLOCK
-B3    : IN BIT;  - PIN B3
CH0RES: IN BIT;  - PIN B4   RESET CONTROL CHANNEL 0
CH3RES: IN BIT;  - PIN B5   RESET CONTROL CHANNEL 3
NIF1RS: OUT BIT; - PIN B6   INPUT FIFO #1 RESET
NIF3RS: OUT BIT; - PIN B7   INPUT FIFO #3 RESET
NOF2RS: OUT BIT; - PIN B8   OUTPUT FIFO #2 RESET
LED2   : OUT BIT; - PIN B9   7-SEG OUTPUT DRIVE BIT 2
-C1    : IN BIT;  - PIN C1
NIF3HF: IN BIT;  - PIN C13  INPUT FIFO #3 HALF FULL
NOF0FF: IN BIT;  - PIN C14  OUTPUT FIFO #0 FULL FLAG
NOF1FF: IN BIT;  - PIN C15  OUTPUT FIFO #1 FULL FLAG
NRESET: IN BIT;  - PIN C2   MANUAL RESET CONTROL
-C3    : IN BIT;  - PIN C3
NOF2FF: IN BIT;  - PIN C8   OUTPUT FIFO #2 FULL FLAG
-D1    : IN BIT;  - PIN D1
LED0   : OUT BIT; - PIN D12  7-SEG OUTPUT DRIVE BIT 0
-D14   : IN BIT;  - PIN D14
NIF0HF: IN BIT;  - PIN D15  INPUT FIFO #0 HALF FULL
-D2    : IN BIT;  - PIN D2
CH2RES: IN BIT;  - PIN D4   RESET CONTROL CHANNEL 2
NOF1HF: IN BIT;  - PIN D8   OUTPUT FIFO #1 HALF FULL
CH3ORY: OUT BIT; - PIN E1   CHANNEL 3 FIFO OUTPUT READY
LED1   : OUT BIT; - PIN E13  7-SEG OUTPUT DRIVE BIT 1
NIF1FF: IN BIT;  - PIN E14  INPUT FIFO #1 FULL FLAG
-15    : IN BIT;  - PIN E15
CH3OCK: OUT BIT; - PIN E2   CHANNEL 3 OUTPUT CLOCK
-E3    : IN BIT;  - PIN E3
NOF3RE: OUT BIT; - PIN F1   OUTPUT FIFO #3 READ ENABLE
NIF1HF: IN BIT;  - PIN F13  INPUT FIFO #1 HALF FULL

```

Appendix B

```

NIF2FF: IN BIT; - PIN F14 INPUT FIFO #2 FULL FLAG
NIF1EF: IN BIT; - PIN F15 INPUT FIFO #1 EMPTY FLAG
CH3DAV: OUT BIT; - PIN F2 CHANNEL 3 OUTPUT DATA VALID
CH3IRY: OUT BIT; - PIN F3 CHANNEL 3 FIFO INPUT READY
CH3CLK: IN BIT; - PIN G1 CHANNEL 3 INPUT CLOCK
NIF2EF: IN BIT; - PIN G13 INPUT FIFO #2 EMPTY FLAG
-G14 : IN BIT; - PIN G14
NIF2HF: IN BIT; - PIN G15 INPUT FIFO #2 HALF FULL
CH3RDE: IN BIT; - PIN G2 CHANNEL 3 INPUT READ ENABLE
CKROF3: OUT BIT; - PIN G3 OUTPUT FIFO #3 READ CLOCK
NOF3EF: IN BIT; - PIN H1 OUTPUT FIFO #3 EMPTY FLAG
-H13 : IN BIT; - PIN H13
CH2T0 : OUT BIT; - PIN H14 CHANNEL 2 DATA SYNC BIT 0
CH3T0 : OUT BIT; - PIN H15 CHANNEL 3 DATA SYNC BIT 0
-H2 : IN BIT; - PIN H2
CH2OCK: OUT BIT; - PIN H3 OUTPUT FIFO #2 OUTPUT CLOCK
CH2IRY: OUT BIT; - PIN J1 CHANNEL 2 FIFO INPUT READY
CH2T1 : OUT BIT; - PIN J13 CHANNEL 2 DATA SYNC BIT 1
MODE0 : IN BIT; - PIN J14 SELECTS OUTPUT CLOCK POLARITY
CH3T1 : OUT BIT; - PIN J15 CHANNEL 3 DATA SYNC BIT 1
CKROF2: OUT BIT; - PIN J2 OUTPUT FIFO #2 READ CLOCK
CH2DAV: OUT BIT; - PIN J3 CHANNEL 2 OUTPUT DATA VALID
CH2ORY: OUT BIT; - PIN K1 CHANNEL 2 FIFO OUTPUT READY
MODE2 : IN BIT; - PIN K13 NOT USED
BOGUS1: IN BIT; - PIN K14 (WAS) INPUT FIFO #0 FULL FLAG
MODE1 : IN BIT; - PIN K15 SELECTS DATA SYNC BIT FORMAT
- MODE1=0, IS TEST MODE, IN TEST MODE
  - T0=1 FOR FIRST PIXEL OF DATA VALID ELSE T0=0
  - T1=1 FOR DATA VALID OF FRAME VALID ELSE T0=0
- MODE1=1, IS MRSR MODE, IN MRSR MODE
  - T0=1 FOR FIRST VALID PIXEL OF FRAME ELSE T0=0
  - T1=DONT CARE
NOF2RE: OUT BIT; - PIN K2 OUTPUT FIFO #2 READ ENABLE
CH2CLK: IN BIT; - PIN K3 CHANNEL 2 INPUT CLOCK
CH2RDE: IN BIT; - PIN L1 CHANNEL 2 INPUT READ ENABLE
NIF0EF: IN BIT; - PIN L13 INPUT FIFO #0 EMPTY FLAG
- DEFINED AS INTEGER DVCT0 : OUT BIT; - PIN L14
FSFIFO: IN BIT; - PIN L15 SELECT FIFO MASTER CLOCK
- FSFIFO=1 IF RCLK IS SLOWER THAN WCLK
- FSFIFO=0 IF RCLK IS FASTER THAN WCLK
-L2 : IN BIT; - PIN L2
CH1OCK: OUT BIT; - PIN L3 CHANNEL 1 OUTPUT CLOCK
NOF2EF: IN BIT; - PIN M1 OUTPUT FIFO #2 EMPTY FLAG
NDV0 : IN BIT; - PIN M12 INPUT DATA VALID
- DEFINED AS INTEGER DVCT1 : OUT BIT; -PIN M14
NOF0WE: OUT BIT; - PIN M15 OUTPUT FIFO #2 WRITE ENABLE
CH1ORY: OUT BIT; - PIN M2 CHANNEL 1 FIFO OUTPUT READY
CH0OCK: OUT BIT; - PIN M4 CHANNEL 0 OUTPUT CLOCK
CLK : IN BIT; - PIN M8 INPUT DATA CLOCK
CH1IRY: OUT BIT; - PIN N1 CHANNEL 1 FIFO INPUT READY
NIF1RE: OUT BIT; - PIN N10 INPUT FIFO #1 READ ENABLE
CH0T0 : OUT BIT; - PIN N13 CHANNEL 0 DATA SYNC BIT 0
NOF1WE: OUT BIT; - PIN N14 OUTPUT FIFO #1 WRITE ENABLE
NOF2WE: OUT BIT; - PIN N15 OUTPUT FIFO #2 WRITE ENABLE
CKROF1: OUT BIT; - PIN N2 OUTPUT FIFO #1 READ CLOCK
CH1CLK: IN BIT; - PIN N3 CHANNEL 1 INPUT CLOCK
CH0CLK: IN BIT; - PIN N6 CHANNEL 0 INPUT CLOCK
NCLR : IN BIT; - PIN N8 POWERUP CLEAR,
- TIED TO VCC THROUGH RC TIME CONSTANT
CH1DAV: OUT BIT; - PIN P1 CHANNEL 1 OUTPUT DATA VALID

```

```

NIF0RE: OUT BIT; - PIN P10 INPUT FIFO #0 READ ENABLE
NFV0 : IN BIT; - PIN P11 INPUT FRAME VALID
NIF0WE: OUT BIT; - PIN P12 INPUT FIFO #0 WRITE ENABLE
CH1T0 : OUT BIT; - PIN P13 CHANNEL 1 DATA SYNC BIT 0
CH0T1 : OUT BIT; - PIN P14 CHANNEL 0 DATA SYNC BIT 1
NOF3WE: OUT BIT; - PIN P15 OUTPUT FIFO #3 WRITE ENABLE
NOF1RE: OUT BIT; - PIN P2 OUTPUT FIFO #1 READ ENABLE
CH1RDE: IN BIT; - PIN P3 CHANNEL 1 INPUT READ ENABLE
NOF1EF: IN BIT; - PIN P4 OUTPUT FIFO #1 EMPTY FLAG
CH0ORY: OUT BIT; - PIN P5 CHANNEL 0 FIFO OUTPUT READY
NOF0RE: OUT BIT; - PIN P6 OUTPUT FIFO #0 READ ENABLE
NOF0EF: IN BIT; - PIN P7 OUTPUT FIFO #0 EMPTY FLAG
NIF0FF: IN BIT; - PIN P8 INPUT FIFO #0 FULL FLAG
NIF3WE: OUT BIT; - PIN P9 INPUT FIFO #3 WRITE ENABLE
NIF2RE: OUT BIT; - PIN R10 INPUT FIFO #2 READ ENABLE
NIF2WE: OUT BIT; - PIN R11 INPUT FIFO #2 WRITE ENABLE
NIF1WE: OUT BIT; - PIN R12 INPUT FIFO #1 WRITE ENABLE
CH1T1 : OUT BIT; - PIN R13 CHANNEL 1 DATA SYNC BIT 1
-R3 : IN BIT; - PIN R3
CH0IRY: OUT BIT; - PIN R4 CHANNEL 0 FIFO INPUT READY
CH0DAV: OUT BIT; - PIN R5 CHANNEL 0 OUTPUT DATA VALID
CKROF0: OUT BIT; - PIN R6 OUTPUT FIFO #0 READ CLOCK
CH0RDE: IN BIT; - PIN R7 CHANNEL 0 INPUT READ ENABLE
-R8 : IN BIT; - PIN R8
NIF3RE: OUT BIT; - PIN R9 INPUT FIFO #3 READ ENABLE
DVCT : OUT INTEGER RANGE 0 TO 3 ;; - DATA VALID COUNT

```

```
END cntrl;
```

```
architecture archCNTRL of CNTRL is
```

```

- THIS REALIZATION IS TARGETED FOR AN ALTERA 7192-10 ALL INTERNAL SIGNAL
- ASSIGNMENTS ARE MADE AFTER 5 ns, ALL EXTERNAL ASSIGNMENTS ( DEVICE I/O
- THROUGH THE PORT ) ARE MADE AFTER 10 ns. THIS IS TO APPROXIMATE INTER-
- NAL AND I/O DELAYS ASSOCIATED WITH THE DEVICE.

```

```
- NOTE : SIGNAL NAMES BEGINNING WITH N ARE LOW TRUE.
```

```

- NXYZ == *XYZ == /XYZ == XYZ

```

```

- NOT(NXYZ) == XYZ

```

```
- THIS CODE HAS SIX MAJOR SECTIONS.
```

```
- I ) INPUT DATA SYNCHRONIZATION.
```

```

- GENERATES SYNCHRONIZATION BITS THAT ARE WRITTEN WITH THE DATA
- INTO THE INPUT FIFOS.

```

```
- II ) RESET CYCLE.
```

```

- GENERATES A BOARD WIDE RESET PULSE SYNCHRONOUS WITH THE INPUT
- FRAME VALID SIGNAL.

```

```
- III ) EXTRA DATA VALID CYCLE.
```

```

- GENERATES THREE EXTRA DATA VALID LINES AFTER THE END OF FRAME
- VALID, USED TO PUSH INPUT DATA THROUGH THE FIFO PIPELINE.

```

```
- IV ) PIPELINE CONTROLLER.
```

```

- GENERATES CONTROL SIGNALS THAT FOLLOW THE DATA THROUGH THE
- DATA PIPELINE.

```

```
- V ) FIFO CONTROL.
```

Appendix B

```
-          COMBINES INPUT SIGNALS WITH SIGNALS GENERATED IN PREVIOUS
-          STAGES TO PRODUCE FIFO CONTROL SIGNALS.
- VI  ) OUTPUT DISPLAY CONTROL.
-          GENERATES SIGNALS TO DRIVE 7-SEGMENT OUTPUT DISPLAY.
-
- DEFINE INTERNAL SIGNALS. THE SIGNALS ARE LISTED
- WITH THE PROCESS OR GROUP OF PROCESSES IN WHICH THEY APPEAR. THE GROUPS
- ARE LISTED IN THE ORDER WHICH THEY APPEAR IN THE CODE.

- INPUT DATA SYNC  ( SNC(N) )

SIGNAL flgate1 : BIT ; - FIRST DATA VALID LINE OF FRAME.
SIGNAL flgate2 : BIT ; - SECOND DATA VALID LINE OF FRAME.
SIGNAL flgate3 : BIT ; - THIRD DATA VALID LINE OF FRAME.
SIGNAL flgate4 : BIT ; - FOURTH DATA VALID LINE OF FRAME.

- RESET CYCLE  ( RST(N) )

SIGNAL res0      : BIT ; - RESET STROBE # 0
SIGNAL res1      : BIT ; - RESET STROBE # 1
SIGNAL resval    : BIT ; - RESET VALID
SIGNAL rstop     : BIT ; - STOP RESET CYCLE
SIGNAL resb1     : BIT ; - RESET BUFFER 1
SIGNAL resb2     : BIT ; - RESET BUFFER 2
SIGNAL resb3     : BIT ; - RESET BUFFER 3
SIGNAL resb4     : BIT ; - RESET BUFFER 4
SIGNAL resb5     : BIT ; - RESET BUFFER 5
SIGNAL resb6     : BIT ; - RESET BUFFER 6
SIGNAL resb7     : BIT ; - RESET BUFFER 7
SIGNAL glbres    : BIT ; - GLOBAL RESET
SIGNAL resgte    : BIT ; - GLOBAL RESET GATE
SIGNAL plcnt     : INTEGER RANGE 0 TO 127 ; - FIRST DV PIXEL COUNT
SIGNAL rcycle    : BIT ; - RESET CYCLE VALID

- EXTRA DATA VALID CYCLE

SIGNAL clrx      : BIT ; - ASYNCHRONOUS FLIP-FLOP CLEAR
SIGNAL xcycle    : BIT ; - EXTRA DATA VALID CYCLE VALID
SIGNAL xcycle1   : BIT ; - XCYLE DELAYED BY 1 CLOCK CYCLE
SIGNAL xstop     : BIT ; - STOP EXTRA DATA VALID CYCLE
SIGNAL p2cnt     : INTEGER RANGE 0 TO 127 ; - EXTRA DV PIXEL COUNT
SIGNAL ceql      : BIT ; - EXTRA DV COUNT = FIRST DV COUNT
SIGNAL holdeq    : BIT ; - EXTEND CEQL TO CREATE DEAD TIME
                  - BETWEEN EXTRA DATA VALID LINES
SIGNAL holdeq1   : BIT ; - HOLDEQ DELAYED BY 1 CLOCK CYCLE
SIGNAL xcnt      : INTEGER RANGE 0 TO 3 ; - EXTRA DV LINE COUNT
SIGNAL xtradv    : BIT ; - EXTRA DATA VALID PULSE

- PIPELINE CONTROLLER

SIGNAL awrite    : BIT ; - DATA VALID AT INPUT FIFO
SIGNAL aw01      : BIT ; - AWRITE DELAYED BY 01 CLOCK CYCLES
SIGNAL aw02      : BIT ; - AWRITE DELAYED BY 02 CLOCK CYCLES
SIGNAL aw03      : BIT ; - AWRITE DELAYED BY 03 CLOCK CYCLES
SIGNAL aw04      : BIT ; - AWRITE DELAYED BY 04 CLOCK CYCLES
SIGNAL aw05      : BIT ; - AWRITE DELAYED BY 05 CLOCK CYCLES
SIGNAL aw06      : BIT ; - AWRITE DELAYED BY 06 CLOCK CYCLES
SIGNAL aw07      : BIT ; - AWRITE DELAYED BY 07 CLOCK CYCLES
SIGNAL aw08      : BIT ; - AWRITE DELAYED BY 08 CLOCK CYCLES
SIGNAL aw09      : BIT ; - AWRITE DELAYED BY 09 CLOCK CYCLES
```

SIGNAL aw10 : BIT ; - AWRITE DELAYED BY 10 CLOCK CYCLES
 SIGNAL aw11 : BIT ; - AWRITE DELAYED BY 11 CLOCK CYCLES
 SIGNAL aw12 : BIT ; - AWRITE DELAYED BY 12 CLOCK CYCLES
 SIGNAL aw13 : BIT ; - AWRITE DELAYED BY 13 CLOCK CYCLES

 SIGNAL nfvo1 : BIT ; - NFV0 DELAYED BY 01 CLOCK CYCLES
 SIGNAL nfvo2 : BIT ; - NFV0 DELAYED BY 02 CLOCK CYCLES
 SIGNAL nfvo3 : BIT ; - NFV0 DELAYED BY 03 CLOCK CYCLES
 SIGNAL nfvo4 : BIT ; - NFV0 DELAYED BY 04 CLOCK CYCLES
 SIGNAL nfvo5 : BIT ; - NFV0 DELAYED BY 05 CLOCK CYCLES
 SIGNAL nfvo6 : BIT ; - NFV0 DELAYED BY 06 CLOCK CYCLES
 SIGNAL nfvo7 : BIT ; - NFV0 DELAYED BY 07 CLOCK CYCLES
 SIGNAL nfvo8 : BIT ; - NFV0 DELAYED BY 08 CLOCK CYCLES
 SIGNAL nfvo9 : BIT ; - NFV0 DELAYED BY 09 CLOCK CYCLES
 SIGNAL nfvo10 : BIT ; - NFV0 DELAYED BY 10 CLOCK CYCLES
 SIGNAL nfvo11 : BIT ; - NFV0 DELAYED BY 11 CLOCK CYCLES
 SIGNAL nfvo12 : BIT ; - NFV0 DELAYED BY 12 CLOCK CYCLES
 SIGNAL nfvo13 : BIT ; - NFV0 DELAYED BY 13 CLOCK CYCLES

 SIGNAL bread0 : BIT ; - 1 CLOCK CYCLE BEFORE BREAD.
 SIGNAL bread : BIT ; - DATA VALID FOR READ PIPELINE STAGE B
 SIGNAL bval0 : BIT ; - DATA VALID GATE FOR READ INPUT FIFO 0
 SIGNAL bval1 : BIT ; - DATA VALID GATE FOR READ INPUT FIFO 1
 SIGNAL bval2 : BIT ; - DATA VALID GATE FOR READ INPUT FIFO 2
 SIGNAL bval3 : BIT ; - DATA VALID GATE FOR READ INPUT FIFO 3
 SIGNAL cshort0 : BIT ; - (BREAD0 - ETRADV) DELAYED 1 CLOCK CYCLE
 SIGNAL cshort1 : BIT ; - (BREAD1 - ETRADV) DELAYED 1 CLOCK CYCLE
 SIGNAL cshort2 : BIT ; - (BREAD2 - ETRADV) DELAYED 1 CLOCK CYCLE
 SIGNAL cshort3 : BIT ; - (BREAD3 - ETRADV) DELAYED 1 CLOCK CYCLE
 SIGNAL clrp : BIT ; - ASYNCHRONOUS FLIP-FLOP CLEAR
 SIGNAL dcnt : INTEGER RANGE 0 TO 3 ; - DV LINE COUNT
 SIGNAL aextra0 : BIT ; - A0 EXTRA DATA VALID LINE
 SIGNAL aextra1 : BIT ; - A1 EXTRA DATA VALID LINE
 SIGNAL aextra2 : BIT ; - A2 EXTRA DATA VALID LINE
 SIGNAL aextra3 : BIT ; - A3 EXTRA DATA VALID LINE
 SIGNAL bextra0 : BIT ; - AEXTRA(0) DELAYED BY 1 CLOCK CYCLE
 SIGNAL bextra1 : BIT ; - AEXTRA(1) DELAYED BY 1 CLOCK CYCLE
 SIGNAL bextra2 : BIT ; - AEXTRA(2) DELAYED BY 1 CLOCK CYCLE
 SIGNAL bextra3 : BIT ; - AEXTRA(3) DELAYED BY 1 CLOCK CYCLE
 SIGNAL cextra0 : BIT ; - AEXTRA(0) DELAYED BY 2 CLOCK CYCLE
 SIGNAL cextra1 : BIT ; - AEXTRA(1) DELAYED BY 2 CLOCK CYCLE
 SIGNAL cextra2 : BIT ; - AEXTRA(2) DELAYED BY 2 CLOCK CYCLE
 SIGNAL cextra3 : BIT ; - AEXTRA(3) DELAYED BY 2 CLOCK CYCLE
 SIGNAL cwrite0 : BIT ; - (CSHORT0 + CEXTRA0) DLYED. 1 CLOCK CYC.
 SIGNAL cwrite1 : BIT ; - (CSHORT1 + CEXTRA1) DLYED. 1 CLOCK CYC.
 SIGNAL cwrite2 : BIT ; - (CSHORT2 + CEXTRA2) DLYED. 1 CLOCK CYC.
 SIGNAL cwrite3 : BIT ; - (CSHORT3 + CEXTRA3) DLYED. 1 CLOCK CYC.
 SIGNAL dtemp : INTEGER RANGE 0 TO 3 ; - DCNT DELAYED 1 CLOCK CYCLE

- OUTPUT DISPLAY CONTROL

SIGNAL alle : BIT ; - ALL 8 FIFOS EMPTY
 SIGNAL allne : BIT ; - ALL 8 FIFOS NOT EMPTY
 SIGNAL allf : BIT ; - ALL 8 FIFOS FULL
 SIGNAL allnf : BIT ; - ALL 8 FIFOS NOT FULL
 SIGNAL ledck : BIT ; - 7 SEGMENT DISPLAY CLOCK
 SIGNAL ledgt : BIT ; - 7 SEGMENT DISPLAY GATE

Appendix B

```

--
--
--
begin -- architecture archCNTRL of CNTRL
--
LED3 <= '0';
--
--
-- *****
-- *** BEGIN INPUT DATA SYNCHRONIZATION BITS ***** SNC(N) ***
-- *****
--
SNC01: process( CLK )

    -- RISING EDGE FLIP-FLOP WITH SYNCHRONOUS RESET AND ENABLE.
    -- FLGATE IS TRUE DURING THE FIRST DATA VALID LINE OF A FRAME.

    begin

        IF(CLK'EVENT AND CLK='1') THEN          -- RISING EDGE CLOCK.

            IF( NFV0='1' ) THEN

                flgate4 <= '1'      AFTER 5 ns ; -- TRUE FOR NEXT FRAME.
                flgate3 <= '1'      AFTER 5 ns ; -- TRUE FOR NEXT FRAME.
                flgate2 <= '1'      AFTER 5 ns ; -- TRUE FOR NEXT FRAME.
                flgate1 <= '1'      AFTER 5 ns ; -- TRUE FOR NEXT FRAME.

                ELSIF( aw01='0' AND aw02='1' ) THEN -- RISING EDGE OF AW02.

                    flgate4 <= flgate3 AFTER 5 ns ; -- FOURTH LINE FINISHED.
                    flgate3 <= flgate2 AFTER 5 ns ; -- THIRD LINE FINISHED.
                    flgate2 <= flgate1 AFTER 5 ns ; -- SECOND LINE FINISHED.
                    flgate1 <= '0'     AFTER 5 ns ; -- FIRST LINE FINISHED.

                END IF;

            END IF;

        end process;

--
--
SNC02: process( MODEL, awrite, aw01, flgate1, flgate4 )

    -- THIS PROCESS GENERATES SYNCHRONIZATION BITS THAT FOLLOW
    -- THE DATA THROUGH THE FIFO PIPELINE.
    -- THERE ARE TWO SYNCHRONIZATION BIT FORMATS
    -- 1) (MODEL=0) DEBUG FORMAT
    -- 2) (MODEL=1) MRSR FORMAT

    begin

        IF( MODEL='0' ) THEN -- TEST MODE

            -- T0 TAGS FIRST PIXEL OF EACH LINE.
            -- T1 TAGS FIRST LINE OF EACH FRAME.

            CH0T0 <= awrite AND NOT( aw01 ) AFTER 10 ns;
            CH0T1 <= awrite AND flgate1     AFTER 10 ns;

```

```

CH1T0 <= awrite AND NOT( aw01 ) AFTER 10 ns;
CH1T1 <= awrite AND flgate1 AFTER 10 ns;

CH2T0 <= awrite AND NOT( aw01 ) AFTER 10 ns;
CH2T1 <= awrite AND flgate1 AFTER 10 ns;

CH3T0 <= awrite AND NOT( aw01 ) AFTER 10 ns;
CH3T1 <= awrite AND flgate1 AFTER 10 ns;

ELSE                                - MRSR MODE

    - CH0T0 TAGS FIRST PIXEL IN EACH OF THE FIRST
    - FOUR LINES. AFTER THE SWITCH MATRIX, THIS
    - TRANSLATES TO THE FIRST PIXEL OF THE FRAME
    - STORED IN EACH OF THE FOUR OUTPUT FIFO CHANNELS.

    CH0T0 <= awrite AND NOT( aw01 )
              AND flgate4 AFTER 10 ns ;
    CH0T1 <= '0' AFTER 10 ns ;

    CH1T0 <= '0' AFTER 10 ns ;
    CH1T1 <= '0' AFTER 10 ns ;

    CH2T0 <= '0' AFTER 10 ns ;
    CH2T1 <= '0' AFTER 10 ns ;

    CH3T0 <= '0' AFTER 10 ns ;
    CH3T1 <= '0' AFTER 10 ns ;

END IF;

end process;

- *****
- *** END INPUT DATA SYNCHRONIZATION BITS ***** SNC(N) ***
- *****
-
- *****
- *** BEGIN RESET CYCLE ***** RST(N) ***
- *****
-
- THE PURPOSE OF THE RESET CYCLE IS TO CLEAR ALL THE BOARDS FIFOS
- AND SYNCHRONIZE THE BOARD TO START RECEIVING DATA AT THE BEGINNING
- OF THE NEXT FRAME VALID CYCLE.
-
- THE RESET CYCLE ACCOMPLISHES 3 THINGS.
-
- 1 ) DE-GLITCH THE RESET REQUEST.
- 2 ) GENERATE A GLOBAL RESET PULSE AT END OF PRESENT FRAME VALID.
- 3 ) COUNT NUMBER OF PIXEL IN FIRST DATA VALID LINE RECEIVED
-     AFTER THE GLOBAL RESET.
-
- THE RESET CYCLE HAS 5 STEPS.
-
- 1 ) VALIDATE THE REQUEST.
- 2 ) AT END OF THE PRESENT FRAME, START RESET CYCLE.
- 3 ) INITIATE GLOBAL RESET AND HOLD HIGH FOR 2 CLOCK CYCLES.
- 4 ) AT BEGINNING OF THE NEXT FRAME, AT START OF FIRST DATA VALID LINE,
-     ENABLE PIXEL COUNTER # 1. ( COUNT HOW MANY PIXELS ARE IN THE

```

Appendix B

```
- FIRST DV LINE. )
- 5 ) AT END OF FIRST DATA VALID LINE, DISABLE PIXEL COUNTER # 1,
- AND END RESET CYCLE.
-
- INPUTS.
- CLK                      ( MASTER INPUT CLOCK )
- NCLR                     ( ALTERA POWERUP INPUT PIN )
- NRESET                   ( MANUAL RESET BUTTON )
- CHORES CH1RES CH2RES CH3RES ( RESETS FROM OUTPUT READING SYSTEM )
- NFV11 NFV12
- AW1 AW2
-
- INTERNAL SIGNALS.
- RES0 RES1 RESVAL
- RSTOP RESB1 RESB2 RESB3 RESB4 RESB5 RESB6 RESB7
-
- OUTPUTS.
- GLBRES                   GLOBLE RESET PULSE
- RESGTE                   GLOBAL RESET GATE
- P1CNT[6:0]               DATA VALID LENGTH COUNT
- RCYCLE                   RESET CYCLE VALID PULSE
-
- STEP 1 ) VALIDATE THE REQUEST.
-
-
RST01: process( NRESET, CH0RES, CH1RES, CH2RES, CH3RES )
- DETECT EXTERNAL OR MANUAL INPUT RESET REQUEST.
-
begin
    res0 <= NOT(NRESET)
        OR CHORES OR CH1RES OR CH2RES OR CH3RES AFTER 5 ns;
-
end process;
-
-
RST02: PROCESS(CLK, res0)
- QUALIFY RESET REQUEST
- USING ASYNCHRONOUS CLEAR FLIP-FLOP,
- AND SYNCHRONOUS CLOCKED REGISTER OUTPUT.
-
- RES0 MUST REMAIN TRUE (=1) FOR LONGER THAN 1 CLOCK CYCLE
- FOR RESVAL TO BECOME TRUE (=1).
-
- IF res0='1', '1' TO OUTPUT ON RISING EDGE OF .
- IF res0='0', '0' THE OUTPUT.
-
- MAX+plus II VHDL Template
- Clearable flipflop
-
BEGIN
    IF(res0='0') THEN - ASYNCHRONOUS CLEAR.
        resval <= '0' AFTER 5 ns;
        res1 <= '0' AFTER 5 ns;
```

```

ELSIF(CLK'EVENT AND CLK='1') THEN

    resval <= res1 AFTER 5 ns;
    res1  <= '1'  AFTER 5 ns;

    END IF;

END PROCESS;

--
--
-- STEP 2 ) BEGIN RESET CYCLE.
--
-- LATCH RESVAL AS RESB1. ONLY VALID RESET SIGNALS ( TRUE FOR LONGER
-- THAN 2 CLOCK CYCLES ) WILL INITIATE A RESET CYCLE ( RESB1 ).
-- RESB1 IS THE FIRST BUFFER OF A DOUBLE BUFFERED RESET START PULSE.
RST03: process(CLK)

    -- IF RESVAL TRUE THEN START CYCLE.
    -- IF RSTOP TRUE THEN END CYCLE.

    begin

        IF(CLK'EVENT AND CLK='1') THEN          -- RISING EDGE CLOCK.

            IF( rstop = '1' ) THEN              -- STOP BUFFER1.

                resb1 <= '0' AFTER 5 ns ;

            ELSE

                IF( resval='1' ) THEN            -- START BUFFER1.

                    resb1 <= '1' AFTER 5 ns ;

                ELSE                             -- DO NOTHING.

                    resb1 <= resb1 AFTER 5 ns ;

                END IF;

            END IF;

        END IF;

    end process;

--
--
RST04: process(CLK)

    -- BUFFER # 2, ON RISING EDGE OF NFV02.
    -- NFV02 IS USED TO AVOID POSSIBLE RACE CONDITIONS BETWEEN
    -- THE END OF DATA VALID AND THE END OF FRAME VALID.
    -- THIS IS THE SECOND BUFFER OF THE RESET START PULSE
    -- THIS PROCESS SYNCHRONIZES THE RESET CYCLE WITH THE
    -- INPUT FRAME VALID. THIS ASSURES THAT THE GLOBAL
    -- RESET WILL HAPPEN AT THE END OF A FRAME AND THE
    -- BOARD WILL BE SET UP TO RECEIVE DATA AT THE BEGINNING
    -- OF THE NEXT FRAME VALID CYCLE.

    begin

```

Appendix B

```
IF(CLK'EVENT AND CLK='1') THEN          - RISING EDGE CLOCK.

    IF( rstop = '1' ) THEN                - STOP BUFFER2.

        resb2 <= '0'    AFTER 5 ns ;

    ELSE

        IF( nfv11='1' AND nfv12='0' ) THEN - START BUFFER2.

            resb2 <= resb1 AFTER 5 ns ;

        ELSE                                - DO NOTHING.

            resb2 <= resb2 AFTER 5 ns ;

        END IF;

    END IF;

END IF;

end process;

-
-
- STEP 3 ) INITIATE GLOBAL RESET AND HOLD HIGH FOR 2 CLOCK CYCLES.
-
-
RST05: process(CLK) - SHIFT RESB2 BY 3 CLK'S AND GENERATE GLBRES
begin

    IF(CLK'EVENT AND CLK='1') THEN

        glbres <= resb3 AND NOT(resb6) AFTER 5 ns ;
        resgte <= resb2 AND NOT(resb7) AFTER 5 ns ;
        resb7 <= resb6 AFTER 5 ns ;
        resb6 <= resb5 AFTER 5 ns ;
        resb5 <= resb4 AFTER 5 ns ;
        resb4 <= resb3 AFTER 5 ns ;
        resb3 <= resb2 AFTER 5 ns ;

    END IF;

end process;

-
-
- STEP 4 ) AT START OF FIRST DATA VALID LINE, ENABLE PIXEL COUNTER # 1.
-
-
RST06: process(CLK)

    - 7 BIT RISING EDGE COUNTER WITH SYNCHRONOUS CLEAR.
    - COUNTS HOW MANY PIXELS IN FIRST DV LINE.

begin

    IF(CLK'EVENT AND CLK='1') THEN          - RISING EDGE CLOCK.
```

```

IF( glbres='1' ) THEN                                - ZERO THE COUNT.

    plcnt <= 0 AFTER 5 ns ;

ELSE

    IF( aw01='1' AND resb2='1' ) THEN - COUNT.

        plcnt <= plcnt + 1 AFTER 5 ns ;

    ELSE                                            - DON'T COUNT.

        plcnt <= plcnt AFTER 5 ns ;

    END IF ;

END IF ;

END IF ;

end process;

-
-
- STEP 5 ) AT END OF FIRST DATA VALID LINE DISABLE PIXEL COUNTER # 1
- AND END RESET CYCLE.
-
RST07: process( CLK, NCLR)

    - IF RESET CYCLE IS TRUE, THEN END THE RESET CYCLE AFTER THE FIRST
    - DATA VALID LINE. THIS ALLOWS YOU TO COUNT HOW MANY PIXELS ARE IN
    - THE FIRST DATA VALID YOU RECEIVE AFTER A RESET.
    - RCYCLE IS A SIGNAL THAT IS VALID FOR THE ENTIRE RESET CYCLE.

    begin

        IF( NCLR='0' ) THEN                        - POWERUP

            rstop <= '1' AFTER 5 ns ;

            ELSIF( CLK'EVENT AND CLK='1' ) THEN

                rstop <= ( resb4 AND NOT(aw01) AND aw02 ) AFTER 5 ns ;
                rcycle <= ( resb1 OR resb5 ) AFTER 5 ns ;

            END IF;

        end process;

    -
    - *****
    - *** END RESET CYCLE ***** RST(N) *****
    - *****
    -
    - *****
    - *** BEGIN EXTRA DATA VALID CYCLE ***** XDV(N) ***
    - *****
    -
    - THE PURPOSE OF THIS CYCLE IS TO GENERATE 3 EXTRA DATA VALID
    - LINES AFTER THE INPUT FRAME VALID ENDS. THE 3 ADDITIONAL LINES

```

Appendix B

```

- ARE REQUIRED TO PUSH THE INPUT DATA THROUGH THE IMAGE RECON-
- STRUCTION BOARDS DATA PIPELINE.
-
- THE EXTRA DATA VALID CYCLE HAS 4 STEPS.

- 1) CHECK FOR BEGINNING OF CYCLE.
- 2) SET EXTRA DATA VALID HIGH AND BEGIN COUNTING PIXELS.
- 3) IF PIXEL COUNT IS EQUAL TO DATA VALID LENGTH THEN ZERO THE
-     COUNT, SET EXTRA DATA VALID LOW, AND WAIT 2 CLOCK CYCLES
-     THEN INCREMENT DATA VALID LINE COUNTER, SET EXTRA DATA VALID
-     HIGH AND BEGIN COUNTING.
- 4) CHECK FOR END OF CYCLE. IF DATA VALID LINE COUNTER IS EQUAL
-     TO 3 THEN SET EXTRA DATA VALID LOW AND END THE CYCLE.
-
- INPUTS.
-   CLK
-   NFV011          FRAME VALID DELAYED BY 11 CLOCK CYCLES
-   NFV012          FRAME VALID DELAYED BY 12 CLOCK CYCLES
-   RCYCLE          RESET CYCLE VALID
-   P1CNT[6:0]      FIRST DATA VALID LINE LENGTH
-
- INTERNAL SIGNALS.
-   XCYCLE          EXTRA DATA VALID CYCLE VALID
-   XCYCLE1         XCYCLE EXTENSION
-   P2CNT[6:0]      EXTRA DATA VALID LINE LENGTH
-   CEQL           DV LENGTH = FIRST DV LENGTH
-   HOLDEQ         EXTEND CEQL
-   HOLDEQ1        EXTEND HOLDEQ
-   XCNT[1:0]       EXTRA DATA VALID LINE COUNT
-   XSTOP          STOP THE CYCLE
-   XTRADV         EXTRA DATA VALID PULSES
-
- OUTPUTS.
-   BREAD          DATA VALID FOR READ AT PIPELINE STAGE
-
- STEP 1 ) CHECK FOR BEGINNING OF CYCLE.
-           IF NOT A RESET CYCLE ( RESB1 = 0 ) THEN AT THE END
-           OF THE NEXT FRAME BEGIN THE CYCLE.
-
-
-
XDV00: process( rcycle, xstop )

- LOGIC FOR FLIP-FLOP CLEAR .

begin

    clr_x <= rcycle OR xstop AFTER 5 ns ;

end process;

-
-
XDV01: process(CLK)

- USING SYNCHRONOUS CLEAR FLIP-FLOP,
- START CYCLE ON RISING EDGE OF NFV12 ( AFTER FRAME VALID ).
- END THE CYCLE WHEN WE ARE TOLD TO STOP BY XSTOP,
- OR IF A RESET CYCLE OCCURS.

begin

```

```

IF(CLK'EVENT AND CLK='1') THEN          - RISING EDGE CLOCK.

    IF( clr_x='1' ) THEN

        xcycle <= '0'    AFTER 5 ns ;      - STOP XCYCLE.

    ELSIF( nfv11='1' AND nfv12='0' ) THEN  - RISING EDGE OF Nfv12.

        xcycle <= '1'    AFTER 5 ns ;      - START XCYCLE.

    END IF;

xcycle1 <= xcycle AFTER 5 ns ;

END IF;

end process;

-
-
- STEP 2 ) SET EXTRA DATA VALID HIGH AND BEGIN COUNTING PIXELS.
-
XDV02: process(CLK)

    - 7 BIT RISING EDGE COUNTER WITH SYNCHRONOUS CLEAR.
    - COUNTS OUT THE PIXELS IN A DATA VALID LINE.
    - GENERATES THE 3 EXTRA DATA VALID LINES NECESSARY TO
    - PUSH THE LAST 3 DATA LINES INTO THE OUTPUT FIFOS.

begin

    IF(CLK'EVENT AND CLK='1') THEN      - RISING EDGE CLOCK.

        IF( xcycle='0' OR holdeq='1' ) THEN - IF NOT XCYCLE OR
                                           - HOLD THEN COUNT=1.

            p2cnt <= 1 AFTER 5 ns ;

        ELSE

            - IF XCYCLE AND NO HOLD
            - THEN COUNT.

            p2cnt <= p2cnt + 1 AFTER 5 ns ;

        END IF ;

    END IF ;

end process;

-
-
- STEP 3 ) IF PIXEL COUNT IS EQUAL TO DATA VALID LENGTH THEN ZERO THE
- COUNT, SET EXTRA DATA VALID LOW, AND WAIT 2 CLOCK CYCLES THEN
- INCREMENT DATA VALID LINE COUNTER, SET EXTRA DATA VALID
- HIGH AND BEGIN COUNTING.
-
XDV03: process( CLK )

    - COMPARES THE PIXEL COUNT OF THE CURRENT EXTRA
    - DATA VALID LINE TO THE LENGTH OF THE FIRST DATA
    - VALID LINE. IF THE TWO COUNTS ARE EQUAL THEN
    - CEQL IS SET HIGH.

```

Appendix B

```

begin

IF(CLK'EVENT AND CLK='1') THEN

    IF( xcycle='1' ) THEN

        -- PIXEL COUNT = DV LENGTH ??

        IF( p2cnt = plcnt ) THEN -- START CEQL AND HOLDEQ

            ceql    <= '1'    AFTER 5 ns ;
            holdeq   <= '1'    AFTER 5 ns ;
            holdeq1  <= '0'    AFTER 5 ns ;

        ELSE

            ceql    <= '0'    AFTER 5 ns ; -- END CEQL
            holdeq1  <= holdeq AFTER 5 ns ;

            IF( holdeq1='1' ) THEN

                holdeq <= '0' AFTER 5 ns ; -- END HOLDEQ

            END IF ;

        END IF ;

    ELSE -- NOT AN XTRADV CYCLE

        ceql    <= '0' AFTER 5 ns ;
        holdeq   <= '0' AFTER 5 ns ;
        holdeq1  <= '0' AFTER 5 ns ;

    END IF ;

END IF ;

end process;

--
--
-- STEP 4) CHECK FOR END OF CYCLE. IF DATA VALID LINE COUNTER IS EQUAL
-- TO 3 THEN SET EXTRA DATA VALID LOW AND END THE CYCLE.
--
XDV04: process(CLK)

    -- 2 BIT RISING EDGE COUNTER WITH SYNCHRONOUS CLEAR.
    -- COUNTS OUT THE NUMBER OF EXTRA DATA VALID LINES.
    -- GENERATES XSTOP TO END COUNT AFTER THIRD LINE.

    begin

        IF(CLK'EVENT AND CLK='1') THEN

            IF( xcycle='0' ) THEN -- ZERO THE COUNT.

                xcnt <= 0 AFTER 5 ns ;

            ELSIF( ceql='1' ) THEN -- COUNT EXTRA DATA VALID LINES.

                xcnt <= xcnt + 1 AFTER 5 ns ;

            
```

```

        END IF ;

        IF( xcnt=3 ) THEN          - STOP CYCLE

            xstop <= '1' AFTER 5 ns ;

        ELSE                      - DON'T STOP YET

            xstop <= '0' AFTER 5 ns ;

        END IF ;

    END IF ;

end process;

--
--
XDV05: process(CLK)

    - THIS IS THE EXTRA DATA VALID PULSE GENERATED BY THIS CYCLE.

    begin

        IF(CLK'EVENT AND CLK='1') THEN

            xtradv <= xcycle AND NOT(holdeq) AND NOT(xstop) AFTER 5 ns ;

        END IF;

    end process;

--
--
-- *****
-- *** END EXTRA DATA VALID CYCLE ***** XDV(N) ***
-- *****
--
-- *****
-- *** BEGIN PIPELINE CONTROLLER ***** PCT(N) ***
-- *****
--
-- THE PURPOSE OF THIS BLOCK IS TO GENERATE THE SIGNALS NEEDED
-- TO PUSH THE DATA THROUGH THE VARIOUS STAGES OF THE IMAGE
-- RECONSTRUCTION BOARDS DATA PIPELINE.
--
-- INPUTS.
--   CLK          INPUT MASTER CLOCK
--   NDV0         INPUT DATA VALID
--   NFV0         INPUT FRAME VALID
--   GLBRES       GLOBAL RESET FROM RESET CYCLE
--   XCYCLE       FROM EXTRA DATA VALID CYCLE
--   XCNT[ 0 - 1 ] FROM EXTRA DATA VALID CYCLE
--   XTRADV       FROM EXTRA DATA VALID CYCLE
--   XSTOP        FROM EXTRA DATA VALID CYCLE
--
-- INTERNAL SIGNALS.
--   CLRP
--   DCNT[ 0 - 1 ]

```

Appendix B

```
- CEXTRA[ 0 - 3 ]
-
- OUTPUTS.
- NFV[ 01 - 13 ]
- AWRITE
- AEXTRA[ 3:0 ]
- AW[ 01 - 13 ]
- BREAD
- BVAL[ 0 - 3 ]
- CWRITE[ 0 - 3 ]
- DVCNT[ 0 - 1 ]
- CSHORT[ 3:0 ]
- CEXTRA[ 3:0 ]

PCT01: process( NDV0, NFV0 )

    begin

        awrite <= NOT(NDV0) AND NOT(NFV0) AFTER 5 ns ;

        - AT THIS TIME DATA IS VALID AT THE FIFO INPUT,
        - STAGE (A) OF THE DATA PIPELINE.
        - SEE. IMAGE RECONSTRUCTION BOARD SCHEMATIC
        - SHEET #'S ( 1 & 2 OF 7 ).

    end process;

-
-
PCT02: process(CLK)

    - DELAY AWRITE BY 1 AND 2 CLOCK CYCLES

    begin

        IF(CLK'EVENT AND CLK='1') THEN

            aw13 <= aw12    AFTER 5 ns ;
            aw12 <= aw11    AFTER 5 ns ;
            aw11 <= aw10    AFTER 5 ns ;
            aw10 <= aw09    AFTER 5 ns ;
            aw09 <= aw08    AFTER 5 ns ;
            aw08 <= aw07    AFTER 5 ns ;
            aw07 <= aw06    AFTER 5 ns ;
            aw06 <= aw05    AFTER 5 ns ;
            aw05 <= aw04    AFTER 5 ns ;
            aw04 <= aw03    AFTER 5 ns ;
            aw03 <= aw02    AFTER 5 ns ;
            aw02 <= aw01    AFTER 5 ns ;
            aw01 <= awrite  AFTER 5 ns ;

            END IF;

        end process;

-
-
PCT03: process(CLK) - SHIFT REGISTER

    - USED TO DELAY NFV0 BY N CLOCK CYCLES.
```

```

begin

    IF(CLK'EVENT AND CLK='1') THEN

        IF( glbres = '1' ) THEN      - INITIALIZE REGISTER TO ALL 1'S.

            nfv13 <= '1'              AFTER 5 ns ;
            nfv12 <= '1'              AFTER 5 ns ;
            nfv11 <= '1'              AFTER 5 ns ;
            nfv10 <= '1'              AFTER 5 ns ;
            nfv09 <= '1'              AFTER 5 ns ;
            nfv08 <= '1'              AFTER 5 ns ;
            nfv07 <= '1'              AFTER 5 ns ;
            nfv06 <= '1'              AFTER 5 ns ;
            nfv05 <= '1'              AFTER 5 ns ;
            nfv04 <= '1'              AFTER 5 ns ;
            nfv03 <= '1'              AFTER 5 ns ;
            nfv02 <= '1'              AFTER 5 ns ;
            nfv01 <= '1'              AFTER 5 ns ;

        ELSE                          - SHIFT REGISTER.

            nfv13 <= nfv12            AFTER 5 ns ;
            nfv12 <= nfv11            AFTER 5 ns ;
            nfv11 <= nfv10            AFTER 5 ns ;
            nfv10 <= nfv09            AFTER 5 ns ;
            nfv09 <= nfv08            AFTER 5 ns ;
            nfv08 <= nfv07            AFTER 5 ns ;
            nfv07 <= nfv06            AFTER 5 ns ;
            nfv06 <= nfv05            AFTER 5 ns ;
            nfv05 <= nfv04            AFTER 5 ns ;
            nfv04 <= nfv03            AFTER 5 ns ;
            nfv03 <= nfv02            AFTER 5 ns ;
            nfv02 <= nfv01            AFTER 5 ns ;
            nfv01 <= NFV0             AFTER 5 ns ;

        END IF;

    END IF;

end process;

--
--
PCT04: process(CLK)

    - DATA VALID FOR READ PIPELINE STAGE B

    begin

        IF(CLK'EVENT AND CLK='1') THEN

            bread <= bread0           AFTER 5 ns ;
            bread0 <= aw11 OR xtradv   AFTER 5 ns ;

        END IF;

    end process;

--
--

```

Appendix B

```

PCT05: process(CLK)

    - VALIDATION GATE FOR READ FROM INPUT FIFO #0.

    begin

        IF(CLK'EVENT AND CLK='1') THEN

            bval0 <= NOT(xcycle1) AFTER 5 ns ;

        END IF;

    end process;

-
-
PCT06: process(CLK) - SHIFT REGISTER

    - CLOCKED ON FALLING EDGE OF BREAD.
    - USED TO GENERATE GATES TO VALIDATE READ FOR INPUT FIFOS.
    - GATES BVAL 0-3 GENERATE THE INPUT FIFO DELAYS.
    -
    -
    - AWRITE  _____
    -
    - XTRADV  _____
    -
    - BVAL0   _____
    -
    - BVAL1   _____
    -
    - BVAL2   _____
    -
    - BVAL3   _____

    begin

        IF(CLK'EVENT AND CLK='1') THEN

            IF( glbres = '1' ) THEN                - INITIALIZE REGISTER
                                                    - TO ALL 0'S.

                bval3 <= '0'   AFTER 5 ns ;
                bval2 <= '0'   AFTER 5 ns ;
                bval1 <= '0'   AFTER 5 ns ;

            ELSIF(bread0='0' AND bread='1') THEN - SHIFT REGISTER
                                                    - ON FALLING EDGE
                                                    - OF BREAD.

                bval3 <= bval2 AFTER 5 ns ;
                bval2 <= bval1 AFTER 5 ns ;
                bval1 <= bval0 AFTER 5 ns ;

            END IF;

        END IF;

    end process;

-
-
PCT07: process(CLK)

```

```

- CSHORT 0-3 ARE EQUIVALENT TO THE INPUT FIFO READ
- SIGNALS NIF[0-3]RE MINUS THE LINE READS GENERATED
- BY THE EXTRA DATA VALIDIS.
-
-
- AW11      _ _ _ _ _ . . . _ _ _ _ _
-
- CSHORT0    _ _ _ _ _ . . . _ _ _ _ _
-
- CSHORT1    _ _ _ _ _ . . . _ _ _ _ _
-
- CSHORT2    _ _ _ _ _ . . . _ _ _ _ _
-
- CSHORT3    _ _ _ _ _ . . . _ _ _ _ _
-
begin
    IF (CLK'EVENT AND CLK='1') THEN
        cshort0 <= aw13 AND bval0 AFTER 10 ns ;
        cshort1 <= aw13 AND bval1 AFTER 10 ns ;
        cshort2 <= aw13 AND bval2 AFTER 10 ns ;
        cshort3 <= aw13 AND bval3 AFTER 10 ns ;
    END IF;
end process;
-
-
PCT08: process( glbres, xstop )
    - LOGIC FOR FLIP-FLOP CLEAR SIGNAL .
begin
    clrp <= glbres OR xstop AFTER 5 ns ;
end process;
-
-
PCT09: process(CLK)
    - 2 BIT FALLING EDGE COUNTER WITH SYNCHRONOUS CLEAR.
    - COUNTS OUT BREAD DATA VALID LINES. ( DCNT )
    - COUNTS ON FALLING EDGE OF BREAD.
begin
    IF (CLK'EVENT AND CLK='1') THEN
        IF ( clrp='1' ) THEN - ZERO THE COUNT
            dcnt <= 0 AFTER 5 ns ;
        ELSIF(bread0='0' AND bread='1') THEN - COUNT
            dcnt <= dcnt + 1 AFTER 5 ns ;
        END IF ;
    END IF ;
end process;

```

Appendix B

```

END IF ;

end process;

-
-
- CALCULATION OF CWRITE[0-3].
-
- THE B-READ DATA IS READ OUT OF THE INPUT FIFOS, TRANSFORMED
- THROUGH THE SWITCH MATRIX, THEN WRITTEN INTO THE OUTPUT FIFOS
- AS C-WRITE DATA. IF THE SWITCH MATRIX WAS A STRAIGHT PASS THROUGH
- LATCH THEN THE BREAD[0-3] COULD BE DELAYED BY A CLOCK CYCLE
- TO FORM CWRITE[0-3]. SINCE THE SWITCH MATRIX TRANSFORMS THE
- DATA BY CHANGING WHICH INPUT DATA LINES GO TO WHICH OUTPUT FIFO,
- THE DATA TRANSFORMATION MUST BE ACCOUNTED FOR WHEN WRITING TO THE
- OUTPUT FIFOS. THE TRANSFORMATION SENDS THE FIRST DATA LINE FROM
- EACH OF THE 4 INPUT TO THE FIRST OUT FIFO, THE SECOND LINE FROM EACH
- INPUT TO THE SECOND OUTPUT, AND SO FORTH UNTIL THE LAST DATA LINE OF
- THE FRAME. BECAUSE OF THE TRANSFORMATION THE TOTAL NUMBER OF LINES
- IN THE FRAME DETERMINES WHICH OUTPUT FIFOS THE LAST THREE DATA
- LINES GO TO. THE LAST THREE DATA LINES ARE THE DATA LINES
- GENERATED BY THE EXTRA DATA VALID SIGNAL. WHERE THE LAST THREE
- DATA LINES GO IS DETERMINED BY THE LINE COUNT, MODULO-4, OF THE
- LAST THREE DATA LINES. THERE ARE FOUR POSSIBLE CASES, THE CASES
- ARE LISTED GRAPHICALLY FOR CLARITY. IT SHOULD BE NOTED THE IN
- THE MRSR SYSTEM THERE ARE 256 DATA LINES OR 0 MODULO-4 WHICH
- IS EQUIVALENT TO NO TRANSFORMATION OR A STRAIGHT DELAY OF THE
- BREAD[0-3] SIGNALS. THE CODE CALCULATES THE OTHER THREE CASES
- TO ALLOW FOR CHANGES TO THE LINE COUNT IN THE FIELD.
-
- CASE I    MOD 4 REMAINDER = 0
-
-   DCNT      0 1 2 3 | 0 1 2
-   XCNT      | 0 1 2
-
-   DATA      0 0 0 0 |
-   LINE        1 1 1 | 1
-                2 2 | 2 2
-                3 | 3 3 3
-
-   OUTPUT FIFO 0
-   OUTPUT FIFO 1
-   OUTPUT FIFO 2
-   OUTPUT FIFO 3
-
- CASE II   MOD 4 REMAINDER = 1
-
-   DCNT      0 1 2 3 0 | 1 2 3
-   XCNT      | 0 1 2
-
-   DATA      0 0 0 0 4 | 4 4 4
-   LINE        1 1 1 1 |
-                2 2 2 | 2
-                3 3 | 3 3
-
-   OUTPUT FIFO 0
-   OUTPUT FIFO 1
-   OUTPUT FIFO 2
-   OUTPUT FIFO 3
-
- CASE III  MOD 4 REMAINDER = 2
-
-   DCNT      0 1 2 3 0 1 | 2 3 0
-   XCNT      | 0 1 2
-
-   DATA      0 0 0 0 4 4 | 4 4
-   LINE        1 1 1 1 5 | 5 5 5
-                2 2 2 2 |
-
-   OUTPUT FIFO 0
-   OUTPUT FIFO 1
-   OUTPUT FIFO 2

```

```

-           3 3 3 | 3           OUTPUT FIFO 3
-
-
- CASE IV MOD 4 REMAINDER = 3
-
-   DCNT  0 1 2 3 0 1 2 | 3 0 1
-   XCNT           | 0 1 2
-
-   DATA 0 0 0 0 4 4 4 | 4           OUTPUT FIFO 0
-   LINE  1 1 1 1 5 5 | 5 5           OUTPUT FIFO 1
-           2 2 2 2 6 | 6 6 6           OUTPUT FIFO 2
-           3 3 3 3 |           OUTPUT FIFO 3
-
- THE 4 CASES SHOW US WHEN TO WRITE TO EACH FIFO DURING THE
- EXTRA DATA VALID CYCLE.
-
- WE WRITE TO FIFO #0 WHEN
-
-   (D=1 AND X=0) OR (D=2 AND X=1) OR (D=3 AND X=2)
-   OR (D=2 AND X=0) OR (D=3 AND X=1)
-   OR (D=3 AND X=0)
-
- WE WRITE TO FIFO #1 WHEN
-
-   (D=2 AND X=0) OR (D=3 AND X=1) OR (D=0 AND X=2)
-   OR (D=3 AND X=0) OR (D=0 AND X=1)
-   OR (D=0 AND X=0)
-
- WE WRITE TO FIFO #2 WHEN
-
-   (D=3 AND X=0) OR (D=0 AND X=1) OR (D=1 AND X=2)
-   OR (D=0 AND X=0) OR (D=1 AND X=1)
-   OR (D=1 AND X=0)
-
- WE WRITE TO FIFO #3 WHEN
-
-   (D=0 AND X=0) OR (D=1 AND X=1) OR (D=2 AND X=2)
-   OR (D=1 AND X=0) OR (D=2 AND X=1)
-   OR (D=2 AND X=0)
-
- WE CAN NOW USE KARNOGH MAPS TO DERIVE THE BINARY EQUATIONS
- THAT TELL US WHEN WE WRITE DATA TO EACH OUTPUT FIFO.
-
-   FIFO                                FIFO
-   #0 D1 0 0 1 1                      #1 D1 0 0 1 1
-       D0 0 1 1 0                      D0 0 1 1 0
-
-   X0 X1 | -|-|-|-|                    X0 X1 | -|-|-|-|
-   0 0   | 0 | 1 | 1 | 1 |              0 0   | 1 | 0 | 1 | 1 |
-       -|-|-|-|-|                    -|-|-|-|-|
-   1 0   | 0 | 0 | 1 | 1 |              1 0   | 1 | 0 | 1 | 0 |
-       -|-|-|-|-|                    -|-|-|-|-|
-   1 1   | 0 | 0 | 0 | 0 |              1 1   | 0 | 0 | 0 | 0 |
-       -|-|-|-|-|                    -|-|-|-|-|
-   0 1   | 0 | 0 | 1 | 0 |              0 1   | 1 | 0 | 0 | 0 |
-       -|-|-|-|-|                    -|-|-|-|-|
-
-   FIFO #0 = X1*D1 + X0*X1*D0 + X0*D0*D1
-
-   - - - - -

```

Appendix B

```

- FIFO #1 = X1*D0*D1 + X1*D0*D1 + X0*D0*D1 + X0*X1*D1
-
-
-
- FIFO
- #2 D1 0 0 1 1
- D0 0 1 1 0
- X0 X1 | - | - | - | - | X0 X1 | - | - | - | - |
- 0 0 | 1 | 1 | 1 | 0 | 0 0 | 1 | 1 | 0 | 1 |
- | - | - | - | - | | - | - | - | - |
- 1 0 | 1 | 1 | 0 | 0 | 1 0 | 0 | 1 | 0 | 1 |
- | - | - | - | - | | - | - | - | - |
- 1 1 | 0 | 0 | 0 | 0 | 1 1 | 0 | 0 | 0 | 0 |
- | - | - | - | - | | - | - | - | - |
- 0 1 | 0 | 1 | 0 | 0 | 0 1 | 0 | 0 | 0 | 1 |
- | - | - | - | - | | - | - | - | - |
-
-
- FIFO #2 = X1*D1 + X0*X1*D0 + X0*D0*D1
-
-
- FIFO #3 = X1*D0*D1 + X1*D0*D1 + X0*D0*D1 + X0*X1*D1
-
-
PCT10: process(CLK)
- OUTPUT FIFO WRITE QUALIFIERS FOR EXTRA DATA VALID.
- AEXTRA[0-3] ARE DELAYED AN EXTRA CLOCK CYCLE TO
- ALIGN CEXTRA[0-3] WITH CSHORT[0-3] AND THE STAGE-C
- PIPELINE DATA.
begin
IF(CLK'EVENT AND CLK='1') THEN

    cextra0 <= bextra0 AFTER 5 ns ;
    bextra0 <= aextra0 AFTER 5 ns ;

    IF( xtradv='1' AND ( (dcnt=1 AND xcnt=0)
                        OR (dcnt=2 AND xcnt=1)
                        OR (dcnt=3 AND xcnt=2)
                        OR (dcnt=2 AND xcnt=0)
                        OR (dcnt=3 AND xcnt=1)
                        OR (dcnt=3 AND xcnt=0))) THEN

        aextra0 <= '1' AFTER 5 ns ;

    ELSE

        aextra0 <= '0' AFTER 5 ns ;

    END IF;

END IF;

end process;
-
-
PCT11: process(CLK)
- OUTPUT FIFO WRITE QUALIFIERS FOR EXTRA DATA VALID.
- AEXTRA[0-3] ARE DELAYED AN EXTRA CLOCK CYCLE TO
- ALIGN CEXTRA[0-3] WITH CSHORT[0-3] AND THE STAGE-C
- PIPELINE DATA.

```

```

begin
    IF(CLK'EVENT AND CLK='1') THEN

        cextral1 <= bextral1      AFTER 5 ns ;
        bextral1 <= aextral1      AFTER 5 ns ;

        IF( xtradv='1' AND ( (dcnt=2 AND xcnt=0)
                               OR (dcnt=3 AND xcnt=1)
                               OR (dcnt=0 AND xcnt=2)
                               OR (dcnt=3 AND xcnt=0)
                               OR (dcnt=0 AND xcnt=1)
                               OR (dcnt=0 AND xcnt=0))) THEN

            aextral1 <= '1' AFTER 5 ns ;

        ELSE

            aextral1 <= '0' AFTER 5 ns ;

        END IF;

    END IF;

end process;
--
--
PCT12: process(CLK)
    -- OUTPUT FIFO WRITE QUALIFIERS FOR EXTRA DATA VALID.
    -- AEXTRA[0-3] ARE DELAYED AN EXTRA CLOCK CYCLE TO
    -- ALIGN CEXTRA[0-3] WITH CSHORT[0-3] AND THE STAGE-C
    -- PIPELINE DATA.
begin
    IF(CLK'EVENT AND CLK='1') THEN

        cextra2 <= bextra2      AFTER 5 ns ;
        bextra2 <= aextra2      AFTER 5 ns ;

        IF( xtradv='1' AND ( (dcnt=3 AND xcnt=0)
                               OR (dcnt=0 AND xcnt=1)
                               OR (dcnt=1 AND xcnt=2)
                               OR (dcnt=0 AND xcnt=0)
                               OR (dcnt=1 AND xcnt=1)
                               OR (dcnt=1 AND xcnt=0))) THEN

            aextra2 <= '1' AFTER 5 ns ;

        ELSE

            aextra2 <= '0' AFTER 5 ns ;

        END IF;

    END IF;

end process;
--
--
PCT13: process(CLK)

```

Appendix B

```

        - OUTPUT FIFO WRITE QUALIFIERS FOR EXTRA DATA VALID.
        - AEXTRA[0-3] ARE DELAYED AN EXTRA CLOCK CYCLE TO
        - ALIGN CEXTRA[0-3] WITH CSHORT[0-3] AND THE STAGE-C
        - PIPELINE DATA.
begin
    IF(CLK'EVENT AND CLK='1') THEN

        cextra3 <= bextra3      AFTER 5 ns ;
        bextra3 <= aextra3     AFTER 5 ns ;

        IF( xtradv='1' AND ( (dcnt=0 AND xcnt=0)
                               OR (dcnt=1 AND xcnt=1)
                               OR (dcnt=2 AND xcnt=2)
                               OR (dcnt=1 AND xcnt=0)
                               OR (dcnt=2 AND xcnt=1)
                               OR (dcnt=2 AND xcnt=0))) THEN

            aextra3 <= '1' AFTER 5 ns ;

        ELSE

            aextra3 <= '0' AFTER 5 ns ;

        END IF;

    END IF;

end process;

-
-
PCT14: process(CLK)    - CWRITE[0-3]

begin

    IF(CLK'EVENT AND CLK='1') THEN

        cwrite0 <= ( cshort0 OR cextra0 ) AFTER 5 ns ;

        cwrite1 <= ( cshort1 OR cextra1 ) AFTER 5 ns ;

        cwrite2 <= ( cshort2 OR cextra2 ) AFTER 5 ns ;

        cwrite3 <= ( cshort3 OR cextra3 ) AFTER 5 ns ;

    END IF;

end process;

-
-
PCT15: process(CLK)
    - ASSIGN ALTERA OUTPUTS.
    - DCNT TO OUTPUTS DVCT0,DVCT1.
    - DVCT0,DVCT1 CONTROL THE SWITCH MATRIX.
    - DCNT IS DELAYED 2 CLOCK CYCLES TO SYNCHRONIZE
    - THE SWITCH MATRIX TRANSITION WITH THE RISING
    - OF NOF[0-3]WE. NOF[0-3]WE IS CWRITE[0-3]
    - INVERTED AND DELAYED 1 CLOCK CYCLE.

begin

```

```

        IF(CLK'EVENT AND CLK='1') THEN

            DVCT  <= dtemp  AFTER 10 ns ;
            dtemp <= dcnt   AFTER 10 ns ;

        END IF;

    end process;

--
-- *****
-- *** END PIPELINE CONTROLLER ***** PCT(N) ***
-- *****
--
-- *****
-- *** BEGIN FIFO CONTROL ***** FFC(N) ***
-- *****
--
-- FIFO RESET CONTROL. PROCESS ( FFC01 ).
--
--
--
FFC01: process( glbres ) -- ASSIGN FIFO RESETS ( ASSIGN ALTERA OUTPUTS )

    -- THIS PROCESS INITIALIZES THE INPUT AND OUTPUT FIFOS.

    begin

        NIF0RS <= NOT(glbres) AFTER 10 ns ;
        NIF1RS <= NOT(glbres) AFTER 10 ns ;
        NIF2RS <= NOT(glbres) AFTER 10 ns ;
        NIF3RS <= NOT(glbres) AFTER 10 ns ;
        NOF0RS <= NOT(glbres) AFTER 10 ns ;
        NOF1RS <= NOT(glbres) AFTER 10 ns ;
        NOF2RS <= NOT(glbres) AFTER 10 ns ;
        NOF3RS <= NOT(glbres) AFTER 10 ns ;

    end process;

--
--
-- FIFO INPUT DATA READ AND WRITE CONTROL, PROCESS ( FFC02 - FFC04 ).
--
--
FFC02: process(CLK)

    -- ASSIGN INPUT FIFO WRITES ( ASSIGN ALTERA OUTPUTS ).
    -- INPUT DATA IS CAPTURED BY WRITING IT TO THE 4 INPUT FIFOS.
    -- THIS IS STAGE (A) OF THE DATA PIPELINE. SEE IMAGE RECON-
    -- STRUCTION BOARD SCHEMATIC SHEET #'S ( 1 & 2 OF 7 ).

    begin

        IF(CLK'EVENT AND CLK='1') THEN

            -- INPUT FIFO WRITES IF[3:0]WE.

            NIF0WE <= NOT( awrite AND NOT(resgte) ) AFTER 10 ns ;
            NIF1WE <= NOT( awrite AND NOT(resgte) ) AFTER 10 ns ;
            NIF2WE <= NOT( awrite AND NOT(resgte) ) AFTER 10 ns ;

```

Appendix B

```
NIF3WE <= NOT( awrite AND NOT(resgte) ) AFTER 10 ns ;

END IF;

end process;

--
--
FFC03: process(CLK)

-- ASSIGN INPUT FIFO READS ( ASSIGN ALTERA OUTPUTS ).
-- INPUT FIFOS CAPTURE THE 4 INPUT DATA STREAMS AND PRESENT
-- THEM TO THE SWITCH MATRIX. THE INPUT FIFOS DELAY EACH
-- STREAM SO THEY APPEAR AT THE SWITCH MATRIX INPUT AT THE
-- CORRECT TIME. THE 4 INPUT FIFO READ SIGNALS ACCOMPLISH
-- INPUT DATA DELAY FOR EACH CHANNEL. THE 4 INPUT CHANNEL
-- DELAYS ARE:

-- DELAY CHANNEL #0 = 0 DATA VALID LINES,
-- DELAY CHANNEL #1 = 1 DATA VALID LINES,
-- DELAY CHANNEL #2 = 2 DATA VALID LINES,
-- DELAY CHANNEL #3 = 3 DATA VALID LINES.

-- STAGE (B) OF THE DATA PIPELINE.
-- SEE. IMAGE RECONSTRUCTION BOARD SCHEMATIC
-- SHEET #'S ( 2 & 3 OF 7 ).

begin

    IF(CLK'EVENT AND CLK='1') THEN

        NIF0RE <= NOT( bread AND bval0 AND NOT(resgte) ) AFTER 10 ns;
        NIF1RE <= NOT( bread AND bval1 AND NOT(resgte) ) AFTER 10 ns;
        NIF2RE <= NOT( bread AND bval2 AND NOT(resgte) ) AFTER 10 ns;
        NIF3RE <= NOT( bread AND bval3 AND NOT(resgte) ) AFTER 10 ns;

    END IF;

end process;

--
--
FFC04: process(CLK)

begin

    IF(CLK'EVENT AND CLK='1') THEN

        -- OUTPUT FIFO WRITES NOF[3:0]WE, WRITE THE OUTPUT DATA INTO
        -- THE 4 OUTPUT FIFOS.

        NOF0WE <= NOT( cwrite0 AND NOT(resgte) ) AFTER 10 ns ;
        NOF1WE <= NOT( cwrite1 AND NOT(resgte) ) AFTER 10 ns ;
        NOF2WE <= NOT( cwrite2 AND NOT(resgte) ) AFTER 10 ns ;
        NOF3WE <= NOT( cwrite3 AND NOT(resgte) ) AFTER 10 ns ;

    END IF;

end process;

--
--
-- OUTPUT CHANNEL 0 CONTROL PROCESS ( FFC05 - FFC10 ).
```

```

-
- I )   OUTPUT FIFO 0 CONTROL.
- II)   CHANNEL 0 OUTPUT HANDSHAKE CONTROL.
-
-
FFC05: process( MODE0,CH0CLK ) - ASSIGN ALTERA OUTPUT
      - OUTPUT DATA CLOCK
      begin
          CH0OCK <= (      MODE0  AND      CH0CLK )
                    OR ( NOT(MODE0) AND NOT(CH0CLK) ) AFTER 10 ns ;
      end process;
-
-
FFC06: process( CH0CLK ) - ASSIGN ALTERA OUTPUT.
      - FIFO READ CLOCK.
      begin
          CKROF0 <= NOT( CH0CLK ) AFTER 10 ns ;
      end process;
-
-
FFC07: process( NOF0EF ) - ASSIGN ALTERA OUTPUT.
      - OUTPUT READY FLAG.
      begin
          CH0ORY <= NOF0EF AFTER 10 ns ;
      end process;
-
-
FFC08: process( NOF0FF ) - ASSIGN ALTERA OUTPUT.
      - INPUT READY FLAG.
      begin
          CH0IRY <= NOF0FF AFTER 10 ns ;
      end process;
-
-
FFC09: process(CH0CLK,resgte)
      - ASSIGN ALTERA OUTPUT NOFORE, ENABLE READING FROM
      - OUTPUT FIFO #0 IF READ ENABLE REQUEST (RDE) IS TRUE
      - AND FIFO IS NOT EMPTY, AND FIFO IS NOT BEING RESET.
      BEGIN
          IF(resgte='1') THEN - ASYNCRONOUS CLEAR.
              NOFORE <= '1' AFTER 10 ns;
          ELSIF(CH0CLK'EVENT AND CH0CLK='1') THEN - RISING CLOCK
              NOFORE <= NOT(CHORDE AND NOF0EF) AFTER 10 ns;
          END IF;

```

Appendix B

```
        end process;
--
--
FFC10: process(CH0CLK, resgte)

    -- ASSIGN ALTERA OUTPUT CH0DAV, DELAY READ ENABLE BY ONE
    -- CLOCK CYCLE AND USE IT TO QUALIFY THE OUTPUT ( DAV ).

    BEGIN

        IF(resgte='1') THEN -- ASYNCHRONOUS CLEAR.

            CH0DAV <= '0' AFTER 10 ns;

            ELSIF(CH0CLK'EVENT AND CH0CLK='0') THEN -- FALLING CLOCK

                CH0DAV <= CHORDE AND NOF0EF AFTER 10 ns;

            END IF;

        end process;
--
--
-- OUTPUT CHANNEL 1 CONTROL PROCESS ( FFC11 - FFC16 ).
--
-- I )   OUTPUT FIFO 1 CONTROL.
-- II)   CHANNEL 1 OUTPUT HANDSHAKE CONTROL.
--
--
FFC11: process( MODE0, CH1CLK ) -- ASSIGN ALTERA OUTPUT
                                -- OUTPUT DATA CLOCK
    begin

        CH1LOCK <= (      MODE0  AND      CH1CLK )
                    OR ( NOT(MODE0) AND NOT(CH1CLK) ) AFTER 10 ns ;

    end process;
--
--
FFC12: process( CH1CLK ) -- ASSIGN ALTERA OUTPUT.
                        -- FIFO READ CLOCK.
    begin

        CKROF1 <= NOT( CH1CLK ) AFTER 10 ns ;

    end process;
--
--
FFC13: process( NOF1EF ) -- ASSIGN ALTERA OUTPUT.
                        -- OUTPUT READY FLAG.
    begin

        CH1ORY <= NOF1EF AFTER 10 ns ;

    end process;
--
--
FFC14: process( NOF1FF ) -- ASSIGN ALTERA OUTPUT.
                        -- INPUT READY FLAG.
    begin
```

```

        CH1IRY <= NOF1FF AFTER 10 ns ;

    end process;

--
--
FFC15: process(CH1CLK, resgte)

    -- ASSIGN ALTERA OUTPUT NOF1RE, ENABLE READING FROM
    -- OUTPUT FIFO #1 IF READ ENABLE REQUEST (RDE) IS TRUE
    -- AND FIFO IS NOT EMPTY, AND FIFO IS NOT BEING RESET.

    BEGIN

        IF(resgte='1') THEN -- ASYNCHRONOUS CLEAR.

            NOF1RE <= '1' AFTER 10 ns;

            ELSIF(CH1CLK'EVENT AND CH1CLK='1') THEN -- RISING CLOCK

                NOF1RE <= NOT(CH1RDE AND NOF1EF) AFTER 10 ns;

            END IF;

        end process;

--
--
FFC16: process(CH1CLK, resgte)

    -- ASSIGN ALTERA OUTPUT CH1DAV, DELAY READ ENABLE BY ONE
    -- CLOCK CYCLE AND USE IT TO QUALIFY THE OUTPUT ( DAV ).

    BEGIN

        IF(resgte='1') THEN -- ASYNCHRONOUS CLEAR.

            CH1DAV <= '0' AFTER 10 ns;

            ELSIF(CH1CLK'EVENT AND CH1CLK='0') THEN -- FALLING CLOCK

                CH1DAV <= CH1RDE AND NOF1EF AFTER 10 ns;

            END IF;

        end process;

--
--
-- OUTPUT CHANNEL 2 CONTROL PROCESS ( FFC17 - FFC22 ).
--
-- I )   OUTPUT FIFO 2 CONTROL.
-- II)   CHANNEL 2 OUTPUT HANDSHAKE CONTROL.
--
--
FFC17: process( MODE0, CH2CLK ) -- ASSIGN ALTERA OUTPUT
                                -- OUTPUT DATA CLOCK

    begin

        CH2OCK <= (      MODE0 AND      CH2CLK )
                   OR ( NOT(MODE0) AND NOT(CH2CLK) ) AFTER 10 ns ;
    end

```

Appendix B

```
end process;
--
--
FFC18: process( CH2CLK ) -- ASSIGN ALTERA OUTPUT.
      -- FIFO READ CLOCK.
begin
      CKROF2 <= NOT( CH2CLK ) AFTER 10 ns ;
end process;
--
--
FFC19: process( NOF2EF ) -- ASSIGN ALTERA OUTPUT.
      -- OUTPUT READY FLAG.
begin
      CH2ORY <= NOF2EF AFTER 10 ns ;
end process;
--
--
FFC20: process( NOF2FF ) -- ASSIGN ALTERA OUTPUT.
      -- INPUT READY FLAG.
begin
      CH2IRY <= NOF2FF AFTER 10 ns ;
end process;
--
--
FFC21: process( CH2CLK, resgte)
-- ASSIGN ALTERA OUTPUT NOF2RE, ENABLE READING FROM
-- OUTPUT FIFO #2 IF READ ENABLE REQUEST (RDE) IS TRUE
-- AND FIFO IS NOT EMPTY, AND FIFO IS NOT BEING RESET.
BEGIN
      IF(resgte='1') THEN -- ASYNCHRONOUS CLEAR.
              NOF2RE <= '1' AFTER 10 ns;
      ELSIF(CH2CLK'EVENT AND CH2CLK='1') THEN -- RISING CLOCK
              NOF2RE <= NOT(CH2RDE AND NOF2EF) AFTER 10 ns;
      END IF;
end process;
--
--
FFC22: process( CH2CLK, resgte)
-- ASSIGN ALTERA OUTPUT CH2DAV, DELAY READ ENABLE BY ONE
-- CLOCK CYCLE AND USE IT TO QUALIFY THE OUTPUT ( DAV ).
BEGIN
      IF(resgte='1') THEN -- ASYNCHRONOUS CLEAR.
```

```

        CH2DAV <= '0' AFTER 10 ns;

        ELSIF(CH2CLK'EVENT AND CH2CLK='0') THEN -- FALLING CLOCK

            CH2DAV <= CH2RDE AND NOF2EF AFTER 10 ns;

        END IF;

    end process;

--
--
-- OUTPUT CHANNEL 3 CONTROL PROCESS ( FFC23 - FFC28 ).
--
-- I )   OUTPUT FIFO 3 CONTROL.
-- II)   CHANNEL 3 OUTPUT HANDSHAKE CONTROL.
--
--
FFC23: process( MODE0,CH3CLK ) -- ASSIGN ALTERA OUTPUT
                                -- OUTPUT DATA CLOCK
    begin

        CH3OCK <= (      MODE0 AND      CH3CLK )
                   OR ( NOT(MODE0) AND NOT(CH3CLK) ) AFTER 10 ns ;

    end process;

--
--
FFC24: process( CH3CLK ) -- ASSIGN ALTERA OUTPUT.
                        -- FIFO READ CLOCK.
    begin

        CKROF3 <= NOT( CH3CLK ) AFTER 10 ns ;

    end process;

--
--
FFC25: process( NOF3EF ) -- ASSIGN ALTERA OUTPUT.
                        -- OUTPUT READY FLAG.
    begin

        CH3ORY <= NOF3EF AFTER 10 ns ;

    end process;

--
--
FFC26: process( NOF3FF ) -- ASSIGN ALTERA OUTPUT.
                        -- INPUT READY FLAG.
    begin

        CH3IRY <= NOF3FF AFTER 10 ns ;

    end process;

--
--
FFC27: process(CH3CLK,resgte)

    -- ASSIGN ALTERA OUTPUT NOF3RE, ENABLE READING FROM
    -- OUTPUT FIFO #3 IF READ ENABLE REQUEST (RDE) IS TRUE
    -- AND FIFO IS NOT EMPTY, AND FIFO IS NOT BEING RESET.

```

Appendix B

```
BEGIN

    IF(resgte='1') THEN -- ASYNCHRONOUS CLEAR.

        NOF3RE <= '1' AFTER 10 ns;

        ELSIF(CH3CLK'EVENT AND CH3CLK='1') THEN -- RISING CLOCK

            NOF3RE <= NOT(CH3RDE AND NOF3EF) AFTER 10 ns;

        END IF;

    end process;

--
--
FFC28: process(CH3CLK,resgte)

    -- ASSIGN ALTERA OUTPUT CH3DAV, DELAY READ ENABLE BY ONE
    -- CLOCK CYCLE AND USE IT TO QUALIFY THE OUTPUT ( DAV ).

    BEGIN

        IF(resgte='1') THEN -- ASYNCHRONOUS CLEAR.

            CH3DAV <= '0' AFTER 10 ns;

            ELSIF(CH3CLK'EVENT AND CH3CLK='0') THEN -- FALLING CLOCK

                CH3DAV <= CH3RDE AND NOF3EF AFTER 10 ns;

            END IF;

        end process;

--
--
-- *****
-- *** END FIFO CONTROL ***** FFC(N) ***
-- *****
--
-- *****
-- *** BEGIN OUTPUT DISPLAY CONTROL ***** ODC(N) ***
-- *****
--
ODC01: process( NIF0EF, NIF1EF, NIF2EF, NIF3EF,
               NOF0EF, NOF1EF, NOF2EF, NOF3EF )

    -- ALLE EQUALS ALL 8 FIFOS ARE EMPTY.
    -- ALLNE EQUALS ALL 8 FIFOS ARE NOT EMPTY.

    begin

        alle <= NOT(NIF0EF) AND NOT(NIF1EF)
              AND NOT(NIF2EF) AND NOT(NIF3EF)
              AND NOT(NOF0EF) AND NOT(NOF1EF)
              AND NOT(NOF2EF) AND NOT(NOF3EF) AFTER 10 ns ;

        allne <= NIF0EF AND NIF1EF
              AND NIF2EF AND NIF3EF
              AND NOF0EF AND NOF1EF
```

```

                AND NOF2EF AND NOF3EF AFTER 10 ns ;

    end process;

-
-
ODC02: process( NIF0FF, NIF1FF, NIF2FF, NIF3FF,
                NOF0FF, NOF1FF, NOF2FF, NOF3FF )

    - ALLF EQUALS ALL 8 FIFOS ARE FULL.
    - ALLNF EQUALS ALL 8 FIFOS ARE NOT FULL.

    begin

        allf <= NOT(NIF0FF) AND NOT(NIF1FF)
                AND NOT(NIF2FF) AND NOT(NIF3FF)
                AND NOT(NOF0FF) AND NOT(NOF1FF)
                AND NOT(NOF2FF) AND NOT(NOF3FF) AFTER 5 ns ;

        allnf <= NIF0FF AND NIF1FF
                AND NIF2FF AND NIF3FF
                AND NOF0FF AND NOF1FF
                AND NOF2FF AND NOF3FF AFTER 5 ns ;

    end process;

-
-
ODC03: process( CLK, NCLR, NRESET)

    - ( STBIN ) STROBE STATE INTO 7-SEGMENT ON RISING
    - EDGE OF THE CLOCK OR POWERUP.
    - USE CLOCK TO BLANK LED DUTY CYCLE.

    begin

        IF(NCLR ='0' OR NRESET='0') THEN

            ledck <= '0';
            ledgt <= '0';

            ELSIF(CLK'EVENT AND CLK='1') THEN

                ledck <= NOT(ledck) ;
                ledgt <= '1' ;

            END IF;

            - (IF NO CLOCK THEN USE NCLR) OR (IF CLOCK THEN USE IT)

            STBIN <= (NCLR AND NOT(ledgt)) OR (ledck AND ledgt) ;
            BLNKIN <= (NCLR AND NOT(ledgt)) OR (ledck AND ledgt) ;

        end process;

-
-
ODC04: process(CLK)

    - LOOKS AT THE FULL AND EMPTY FLAGS.
    - DRIVES A 7-SEGMENT DISPLAY.
    - TELLS WHAT STATE THE BOARD IS IN.
    -

```

Appendix B

```
- STATE 0 ( ALL FIFOS EMPTY )
- STATE 1 ( NOT ALL FIFOS EMPTY, NO FIFOS FULL )
- STATE 2 ( ALL FIFOS NOT EMPTY, NO FIFOS FULL )
- STATE 3 ( ANY FIFO FULL )
- STATE 4 ( ALL FIFOS FULL )
- STATE 5 ( POWERUP )
- STATE 6 ( ELSE )
-

begin

  IF( NCLR='0' OR NRESET='0' ) THEN  - POWERUP CLEAR, STATE 5

    LED2 <= '1' AFTER 10 ns ;
    LED1 <= '0' AFTER 10 ns ;
    LED0 <= '1' AFTER 10 ns ;

  ELSIF( CLK'EVENT AND CLK='1' ) THEN
    - CLOCKED STATES 0,1,2,3,4, AND 6

    IF( alle='1' ) THEN  - ALL EMPTY, STATE 0

      LED2 <= '0' AFTER 10 ns ;
      LED1 <= '0' AFTER 10 ns ;
      LED0 <= '0' AFTER 10 ns ;

    ELSIF( alle='0' AND allnf='1' ) THEN  - DATA START, STATE 1

      LED2 <= '0' AFTER 10 ns ;
      LED1 <= '0' AFTER 10 ns ;
      LED0 <= '1' AFTER 10 ns ;

    ELSIF( allne='1' AND allnf='1' ) THEN  - DATA RUN, STATE 2

      LED2 <= '0' AFTER 10 ns ;
      LED1 <= '1' AFTER 10 ns ;
      LED0 <= '0' AFTER 10 ns ;

    ELSIF( allnf='0' ) THEN  - ANY FULL ERROR, STATE 3

      LED2 <= '0' AFTER 10 ns ;
      LED1 <= '1' AFTER 10 ns ;
      LED0 <= '1' AFTER 10 ns ;

    ELSIF( allf='1' ) THEN  - ALL FULL ERROR, STATE 4

      LED2 <= '1' AFTER 10 ns ;
      LED1 <= '0' AFTER 10 ns ;
      LED0 <= '0' AFTER 10 ns ;

    ELSE  - DEFAULT STATE 6

      LED2 <= '1' AFTER 10 ns ;
      LED1 <= '1' AFTER 10 ns ;
      LED0 <= '0' AFTER 10 ns ;

    END IF ;

  END IF ;

END IF ;
```

```
        end process;
--
--
-- *****
-- *** END OUTPUT DISPLAY CONTROL ***** ODC(N) ***
-- *****
--
--
end archCNTRL;
```


Appendix B.—VHDL Files (cont'd)

B-3.— v_icareco.vhd


```

library pack1076;
use pack1076.pack1076.all;

entity ICON_RECO is

    PORT(signal CLK      : IN  vlbit ;
          signal NCLR    : IN  vlbit ;
          signal NPRE     : IN  vlbit ;
          signal DV_ON    : IN  vlbit_vector(7 downto 0);
          signal DV_OFF   : IN  vlbit_vector(7 downto 0);
          signal FV_ON    : IN  vlbit_vector(9 downto 0);
          signal FV_OFF   : IN  vlbit_vector(9 downto 0);
          signal OCLK     : OUT vlbit;
          signal NDV      : OUT vlbit;
          signal NFV      : OUT vlbit;
          signal CH0      : OUT vlbit_vector(15 downto 0);
          signal CH1      : OUT vlbit_vector(15 downto 0);
          signal CH2      : OUT vlbit_vector(15 downto 0);
          signal CH3      : OUT vlbit_vector(15 downto 0));

end ICON_RECO ;

-- SIGNALS IN PORT STATEMENT THAT BEGIN WITH N ARE LOW TRUE

architecture archICON_RECO of ICON_RECO is

    signal DV_CNT      : vlbit_vector(7 downto 0) ; -- data valid count
    signal DV_STOP     : vlbit_vector(7 downto 0) ; -- data valid stop count
    signal STOPDV      : vlbit ; -- data valid stop
    signal TOGGLEDV    : vlbit ; -- data valid toggle
    signal NDVUG       : vlbit ; -- data valid ungated

    signal FV_CNT      : vlbit_vector(9 downto 0) ; -- frame valid count
    signal FV_STOP     : vlbit_vector(9 downto 0) ; -- frame valid stop count
    signal STOPFV      : vlbit ; -- frame valid stop
    signal TOGGLEFV    : vlbit ; -- frame valid toggle

    signal PIX         : vlbit_vector(13 downto 0) ; -- pixel count

begin

-- THIS PROCESS GENERATES THE OUTPUT CLOCK (OCLK)

CLOCK: process( CLK )

    begin

        OCLK <= CLK ;

    end process; -- CLOCK

-- *****
-- *****      DATA VALID  GENERATOR      *****
-- *****

-- 8_BIT 2_TO_1 SELECTOR

```

Appendix B

```
DV_2TO1: process( TOGGLEDV, DV_STOP, DV_ON, DV_OFF )
    begin -- DV_2TO1

    if( TOGGLEDV = '0' ) then

        DV_STOP <= DV_ON AFTER 1 ns ;
        -- PUTLINE("DV_STOP = DV_ON");
        end if;

    if( TOGGLEDV = '1' ) then

        DV_STOP <= DV_OFF AFTER 1 ns ;
        -- PUTLINE("DV_STOP = DV_OFF");
        end if;

    end process; -- DV_2TO1

-- 8 BIT RISING EDGE COUNTER WITH SYNCHRONOUS CLEAR
DV_CNTR: process

    variable dcnt : INTEGER;
    variable temp : VLBIT_VECTOR(31 downto 0);

    begin -- DV_COUNTER

    wait until ( prising(CLK) );

    if( (NCLR = '1') and (STOPDV = '0') ) then
        dcnt := dcnt + 1 ;          -- PUTLINE("increment count ");
    end if;

    if ( (NCLR = '0') or (STOPDV = '1')) then
        dcnt := 0 ;                -- PUTLINE("reset count ");
    end if;

    temp := int2vld(dcnt);         -- pass count out to DV_CNT
    DV_CNT <= temp(7 downto 0) AFTER 1 ns ;

    end process; -- DV_COUNTER

-- 8 BIT IDENTITY COMPARITOR
DV_CMPR: process( DV_CNT, DV_STOP, NCLR )

    variable cnt : INTEGER;
    variable temp : VLBIT_VECTOR(31 downto 0);

    begin -- DV_CMPR

    if( NCLR = '1' ) then

        if ( DV_CNT = DV_STOP ) then
            STOPDV <= '1' AFTER 1 ns ;    -- PUTLINE("turn stop on");
        end if;
    end if;
end process;
```

```

        else
            STOPDV <= '0' AFTER 1 ns ;    - PUTLINE("turn stop off");
        end if;

    end if;

    if ( NCLR = '0' ) then
        STOPDV <= '0' AFTER 1 ns ;    - PUTLINE("turn stop off clear");
    end if;

end process; - DV_CMPR

- THIS PROCESS IS A RISING EDGE D-FLIP-FLOP WITH SYNCHRONOUS CLEAR
- WITH GATE LOGIC TO CONTROL TOGGLEDDV

DV_D_FF:process

begin - DV_D_FF

    wait until ( prising(CLK) );

    if ( NCLR = '1' ) then

        if( STOPDV = '1' ) then
            TOGGLEDDV <= not(TOGGLEDDV) AFTER 1 ns ;
        else
            TOGGLEDDV <= TOGGLEDDV AFTER 1 ns ;
        end if;

    end if;

    if ( NCLR = '0' ) then
        TOGGLEDDV <= '1' ;
    end if;

end process ; - DV_D_FF

- *****
- *****      FRAME VALID GENERATOR      *****
- *****

- 10_BIT 2_TO_1 SELECTOR

FV_2T01: process( TOGGLEFV, FV_STOP, FV_ON, FV_OFF )

begin - FV_2T01

    if( TOGGLEFV = '0' ) then

        FV_STOP <= FV_ON AFTER 1 ns ;
        - PUTLINE("FV_STOP = FV_ON");
    end if;

    if( TOGGLEFV = '1' ) then

```

Appendix B

```
        FV_STOP <= FV_OFF AFTER 1 ns ;
    - PUTLINE("FV_STOP = FV_OFF");
    end if;

    end process;

FV_CNTR: process - 10 BIT RISING EDGE COUNTER WITH SYNCHRONOUS CLEAR

    variable fcnt : INTEGER;
    variable temp : VLBIT_VECTOR(31 downto 0);

    begin - FV_COUNTER

        wait until ( prising(TOGGLEDV) );

        if( (NCLR = '1') and (STOPFV = '0') ) then
            fcnt := fcnt + 1 ;          - PUTLINE("increment count ");
        end if;

        if ( (NCLR = '0') or (STOPFV = '1')) then
            fcnt := 0 ;                - PUTLINE("reset count ");
        end if;

        temp := int2vld(fcnt);         - pass count out to FV_CNT
        FV_CNT <= temp(9 downto 0) AFTER 1 ns ;

        end process; - FV_COUNTER

- 10 BIT IDENTITY COMPARITOR

FV_CMPR: process( FV_CNT, FV_STOP, NCLR )

    variable cnt : INTEGER;
    variable temp : VLBIT_VECTOR(31 downto 0);

    begin - FV_CMPR

        if( NCLR = '1' ) then

            if ( FV_CNT = FV_STOP ) then
                STOPFV <= '1' AFTER 1 ns ;    - PUTLINE("turn stop on");
            else
                STOPFV <= '0' AFTER 1 ns ;    - PUTLINE("turn stop off");
            end if;

        end if;

        if ( NCLR = '0' ) then
            STOPFV <= '0' AFTER 1 ns ;        - PUTLINE("turn stop off clear");
        end if;

        end process; - FV_CMPR
```

- THIS PROCESS IS A RISING EDGE D-FLIP-FLOP WITH SYNCHRONOUS CLEAR
- WITH GATE LOGIC TO CONTROL TOGGLEFV

```

FV_D_FF:process
begin -- FV_D_FF

    wait until ( prising(TOGGLEDV) );

    if ( NCLR = '1' ) then

        if( STOPFV = '1' ) then
            TOGGLEFV <= not(TOGGLEFV)  AFTER 1 ns ;
        else
            TOGGLEFV <= TOGGLEFV  AFTER 1 ns ;
        end if;

    end if;

    if ( NCLR = '0' ) then
        TOGGLEFV <= '1' ;
    end if;

end process ; -- FV_D_FF

```

```

- *****
- *****          OUTPUT GENERATOR          *****
- *****

```

- THIS PROCESS IS A 14 BIT RISING EDGE COUNTER WITH SYNCHRONOUS CLEAR

```

PIX_COUNTER:process -- 14 BIT RISING EDGE COUNTER WITH SYNCHRONOUS CLEAR

```

```

    variable pcnt : INTEGER;
    variable temp : VLBIT_VECTOR(31 downto 0);

```

```

begin -- PIX_COUNTER

```

```

    wait until ( prising(CLK) );

```

```

-- OUTPUT COUNT VALUE, FRAME VALID AND DATA VALID

```

```

if( (NCLR = '1') and (TOGGLEDV = '0') and (TOGGLEFV = '0') ) then
    PIX <= temp(13 downto 0)  AFTER 7 ns ;
else
    PIX <= "XXXXXXXXXXXXXXXX" AFTER 7 ns ;
end if;

```

```

NDV <= TOGGLEDV or TOGGLEFV AFTER 7 ns ;
NFV <= TOGGLEFV  AFTER 7 ns ;

```

```

-- UPDATE COUNT AND PASS TO TEMP

```

```

if( (NCLR = '1') and (TOGGLEDV = '0') and (TOGGLEFV = '0') ) then
    pcnt := pcnt + 1 ;          -- PUTLINE("increment count ");
end if;

```

Appendix B

```
    if ( NCLR = '0' or (TOGGLEFV = '1') ) then
        pcnt      := 0 ;                      - PUTLINE("reset count ");
    end if;

    temp      := int2vld(pcnt);              - pass count out to FV_CNT

end process; - PIX_COUNTER

- THIS PROCESS GENERATES FOUR 16 BIT OUTPUT PIXEL CHANNELS

PIX_OUTPUT:process( PIX )

begin - PIX_OUTPUT

    CH0(13 downto 00) <= PIX    AFTER 7 ns ;
    CH0(15 downto 14) <= "00"  AFTER 7 ns ;
    CH1(13 downto 00) <= PIX    AFTER 7 ns ;
    CH1(15 downto 14) <= "01"  AFTER 7 ns ;
    CH2(13 downto 00) <= PIX    AFTER 7 ns ;
    CH2(15 downto 14) <= "10"  AFTER 7 ns ;
    CH3(13 downto 00) <= PIX    AFTER 7 ns ;
    CH3(15 downto 14) <= "11"  AFTER 7 ns ;

end process; - PIX_OUTPUT

end archICON_RECO;
```

Appendix C.—Viewsim Command Files

C-1.—tbench.cmd


```

||||| VECTOR DECLARATION
|||||

```

```
| INPUT TEST VECTORS ICON_RECO
```

```

vector ch0in      ch0in[15:00]
vector chl1in     chl1in[15:00]
vector ch2in      ch2in[15:00]
vector ch3in      ch3in[15:00]
vector dv_on      dv_on[7:0]
vector dv_off     dv_off[7:0]
vector fv_on      fv_on[9:0]
vector fv_off     fv_off[9:0]

vector dcnt       $1i64\dcnt1 $1i64\dcnt0
vector xcnt       $1i64\xcnt1 $1i64\xcnt0

```

```
| OUTPUT DATA IM_RECO SHEET 1 OF 7
```

```

vector ch0das     $1i64\ch0da[07:00]
vector ch0db      $1i64\ch0db[07:00]
vector ch0dc      $1i64\ch0dc[07:00]

vector ch0da      $1i64\ch0da[15:14] $1i64\ch0da[07:00]
vector ch0db      $1i64\ch0db[15:14] $1i64\ch0db[07:00]
vector ch0dc      $1i64\ch0dc[15:14] $1i64\ch0dc[07:00]

vector chl1da     $1i64\chl1da[15:14] $1i64\chl1da[07:00]
vector chl1db     $1i64\chl1db[15:14] $1i64\chl1db[07:00]
vector chl1dc     $1i64\chl1dc[15:14] $1i64\chl1dc[07:00]

vector ch2da      $1i64\ch2da[15:14] $1i64\ch2da[07:00]
vector ch2db      $1i64\ch2db[15:14] $1i64\ch2db[07:00]
vector ch2dc      $1i64\ch2dc[15:14] $1i64\ch2dc[07:00]

vector ch3da      $1i64\ch3da[15:14] $1i64\ch3da[07:00]
vector ch3db      $1i64\ch3db[15:14] $1i64\ch3db[07:00]
vector ch3dc      $1i64\ch3dc[15:14] $1i64\ch3dc[07:00]

vector dvcnt      $1i64\dvcnt1 $1i64\dvcnt0

vector c0data     c0t[15:14] c0t[07:00]
vector c1data     c1t[15:14] c1t[07:00]
vector c2data     c2t[15:14] c2t[07:00]
vector c3data     c3t[15:14] c3t[07:00]

vector c0sync     c0t[17:16]
vector c1sync     c1t[17:16]
vector c2sync     c2t[17:16]
vector c3sync     c3t[17:16]

vector plcnt      $1i64\p1cnt6 $1i64\p1cnt5 +
                  $1i64\p1cnt4 $1i64\p1cnt3 +
                  $1i64\p1cnt2 $1i64\p1cnt1 +
                  $1i64\p1cnt0

vector p2cnt      $1i64\p2cnt6 $1i64\p2cnt5 +
                  $1i64\p2cnt4 $1i64\p2cnt3 +
                  $1i64\p2cnt2 $1i64\p2cnt1 +

```

Appendix C

```
$1i64\6i2\p2cnt0

vector led          $1i64\led3 $1i64\led2 $1i64\led1 $1i64\led0

||||| CONVERTING ALL THE VECTORS TO HEX |||||

radix hex ch0in
radix hex chl1n
radix hex ch2in
radix hex ch3in
radix hex dv_on
radix hex dv_off
radix hex fv_on
radix hex fv_off

radix hex ch0da
radix hex ch0db
radix hex chl1da
radix hex ch2da
radix hex ch3da

radix hex plcnt
radix hex p2cnt

radix hex c0data
radix hex c1data
radix hex c2data
radix hex c3data

radix hex led
||||| VECTOR ASSIGNMENT |||||

assign dv_on 03\h
assign dv_off 03\h
assign fv_on 02\h
assign fv_off 02\h

||||| CREATING A PLOT ON VIEW TRACE - IM_RECO1.WFM |||||

wave input.wfm inclk ~indv ~inlv +
                    ch0in chl1n ch2in ch3in +
                    $1i64\ckd10 +
                    $1i64\6i2\awrite $1i64\6i2\aw1

wave rcycle.wfm $1i64\6i2\CLK      $1i64\6i2\NDV0  $1i64\6i2\NFV0 +
                $1i64\6i2\NRESET  +
                $1i64\6i2\CH0RES  $1i64\6i2\CH1RES  +
                $1i64\6i2\CH2RES  $1i64\6i2\CH3RES  +
                $1i64\6i2\res0    $1i64\6i2\res1    +
                $1i64\6i2\rstop   $1i64\6i2\resb1   $1i64\6i2\resb2 +
                $1i64\6i2\resb3   $1i64\6i2\resb4   $1i64\6i2\resb5 +
                $1i64\6i2\glbres  $1i64\6i2\awrite plcnt +
                $1i64\~if0rs      $1i64\6i2\rcycle
```

```

wave xcycle.wfm $1i64\6i2\CLK      $1i64\6i2\nfv2    $1i64\6i2\rcycle +
                $1i64\6i2\glbres    plcnt           $1i64\6i2\xcycle +
                $1i64\6i2\xstop     p2cnt           +
                $1i64\6i2\ceql      $1i64\6i2\holdeq $1i64\6i2\holdeql +
                $1i64\6i2\xcnt      $1i64\6i2\awrite +
                $1i64\6i2\xtradv    $1i64\6i2\bread  $1i64\6i2\nc1r

| wave pline.wfm  $1i64\6i2\CLK      $1i64\6i2\glbres    $1i64\6i2\xcycle +
|                $1i64\6i2\awrite    $1i64\6i2\xtradv    $1i64\6i2\xcnt +
|                $1i64\6i2\xstop     $1i64\6i2\bread    $1i64\6i2\bval0  +
|                $1i64\6i2\bval1     $1i64\6i2\bval2     $1i64\6i2\bval3  +
|                $1i64\6i2\awl       $1i64\6i2\xcnt      $1i64\6i2\dcnt  +
|                $1i64\6i2\cshort3   $1i64\6i2\bextra3   $1i64\6i2\cwrite3

| wave abfifo.wfm $1i64\6i2\CLK      $1i64\~if0rs    $1i64\ckwif0    $1i64\ckrif0  +
|                $1i64\~if0ef        $1i64\~if1ef    $1i64\~if2ef    $1i64\~if3ef  +
|                $1i64\~if0we        $1i64\~if1we    $1i64\~if2we    $1i64\~if3we  +
|                ch0da               ch1da           ch2da           ch3da       +
|                $1i64\~if0re        $1i64\~if1re    $1i64\~if2re    $1i64\~if3re  +
|                ch0db               ch1db           ch2db           ch3db

| wave bcfifo.wfm $1i64\6i2\CLK      $1i64\~of0rs    $1i64\ckrif0    $1i64\ckwof0  +
|                $1i64\~if0re        $1i64\~if1re    $1i64\~if2re    $1i64\~if3re  +
|                ch0db               ch1db           ch2db           ch3db       +
|                dvcnt               +
|                $1i64\~of0we        $1i64\~of1we    $1i64\~of2we    $1i64\~of3we  +
|                ch0dc               ch1dc           ch2dc           ch3dc

wave input.wfm inclk ~indv ~invf +
                ch0in ch1in ch2in ch3in +
                $1i64\ckd10 +
                $1i64\6i2\awrite $1i64\6i2\awl

wave output.wfm resc0 rdec0 clkc0 +
                $1i64\ckwof0 $1i64\~of0we $1i64\~of0rs +
                $1i64\ckrof0 $1i64\~of0re $1i64\~of0ef $1i64\~of0ff +
                c0ory c0dav c0clk c0iry c0data

wave dataout.wfm $1i64\6i2\CH0RES rdec0 ~invf $1i64\6i2\glbres +
                c0clk c0ory c0dav c0sync c0data +
                c1clk c1ory c1dav c1sync c1data +
                c2clk c2ory c2dav c2sync c2data +
                c3clk c3ory c3dav c3sync c3data +
                $1i64\192clr led

wave bval.wfm $1i64\6i2\clk $1i64\6i2\ndv0 $1i64\6i2\nfv0 +
                $1i64\6i2\awrite $1i64\6i2\awl $1i64\6i2\aw2 +
                $1i64\6i2\aw12 +
                $1i64\6i2\clrx $1i64\6i2\nfv1 $1i64\6i2\nfv2 +
                $1i64\6i2\xcycle $1i64\6i2\xtradv $1i64\6i2\bread0 +
                $1i64\6i2\bread $1i64\6i2\bval0 $1i64\6i2\bval1 +
                $1i64\6i2\bval2 $1i64\6i2\bval3 $1i64\6i2\bval1 +

wave cwri.wfm  $1i64\6i2\clk      $1i64\6i2\xtradv +
                dcnt xcnt +
                $1i64\6i2\axextra0 +
                $1i64\6i2\axextra1 +
                $1i64\6i2\axextra2 +
                $1i64\6i2\axextra3 +
                $1i64\6i2\cshort0 +

```

Appendix C

```
$1i64\56i2\cshort1 +
$1i64\56i2\cshort2 +
$1i64\56i2\cshort3

wave led.wfm $1i64\56i2\clk $1i64\56i2\ndv0 $1i64\56i2\nfv0 +
$1i64\~if0we $1i64\~if0re +
$1i64\~if0ef $1i64\~if1ef $1i64\~if2ef $1i64\~if3ef +
$1i64\~of0ef $1i64\~of1ef $1i64\~of2ef $1i64\~of3ef +
led

||||| INITIALIZATION |||||
|||||
clock clk 1 0
stepsize 36.32ns
| MRSR FREQUENCY OF 13.767 MHZ

| RISING EDGE OUTPUT DATA CLOCK
h $1i64\56i2\MODE0

| FIX MINUS DIFFERENTIAL INPUTS
h $1i64\MRESCH0 $1i64\MRDECH0 $1i64\MCLKCH0
h $1i64\MRESCH1 $1i64\MRDECH1 $1i64\MCLKCH1
h $1i64\MRESCH2 $1i64\MRDECH2 $1i64\MCLKCH2
h $1i64\MRESCH3 $1i64\MRDECH3 $1i64\MCLKCH3

| SET INPUT DELAY FOR INCLK ~INDV AND ~INLV

| Ha0 Ha1 La2
h $1i64\dlcka0
h $1i64\dlcka1
l $1i64\dlcka2

l $1i64\dldva0
l $1i64\dldva1
l $1i64\dldva2

l $1i64\dlfva0
l $1i64\dlfva1
l $1i64\dlfva2

| ALTERA COMPILED - DRIVE GND AND VDD FOR ALTERA CNTRL.
h $1i64\56i2\vdd
l $1i64\56i2\gnd

| INITIALIZE VHDL INPUT TEST VECTOR GENERATOR ( ICON_RECO )
| AND PULL POWERUP PIN FOR ALTERA-CNTRL.VHD (NCLR) LOW.

l $1i64\192clr

h ~pre
l ~clr
c 5
h ~clr

r $1i64\192clr

| INITIALIZE OUTPUT READ CIRCUIT OSCILLATOR

l $1i206\probe
```

```

s 1000
r $1i206\probe

| ISSUE A RESET TO BOARD THEN BEGIN READING DATA.

| VHDL
| h $1i64\6i2\rstop
| h $1i64\6i2\stopeq

| ALTERA COMPILED

l $1i64\6i2\P1CNT0.CLRN
l $1i64\6i2\P1CNT1.CLRN
l $1i64\6i2\P1CNT2.CLRN
l $1i64\6i2\P1CNT3.CLRN
l $1i64\6i2\P1CNT4.CLRN
l $1i64\6i2\P1CNT5.CLRN
l $1i64\6i2\P1CNT6.CLRN

l $1i64\6i2\P2CNT0.CLRN
l $1i64\6i2\P2CNT1.CLRN
l $1i64\6i2\P2CNT2.CLRN
l $1i64\6i2\P2CNT3.CLRN
l $1i64\6i2\P2CNT4.CLRN
l $1i64\6i2\P2CNT5.CLRN
l $1i64\6i2\P2CNT6.CLRN

h $1i64\6i2\xstop

c 15

r $1i64\6i2\P1CNT0.CLRN
r $1i64\6i2\P1CNT1.CLRN
r $1i64\6i2\P1CNT2.CLRN
r $1i64\6i2\P1CNT3.CLRN
r $1i64\6i2\P1CNT4.CLRN
r $1i64\6i2\P1CNT5.CLRN
r $1i64\6i2\P1CNT6.CLRN

r $1i64\6i2\P2CNT0.CLRN
r $1i64\6i2\P2CNT1.CLRN
r $1i64\6i2\P2CNT2.CLRN
r $1i64\6i2\P2CNT3.CLRN
r $1i64\6i2\P2CNT4.CLRN
r $1i64\6i2\P2CNT5.CLRN
r $1i64\6i2\P2CNT6.CLRN

r $1i64\6i2\xstop

| l $1i64\6i2\_4I31_276.CLRN $1i64\6i2\XSTOP_27__Q_474.CLRN
| l $1i64\6i2\P2CNT_17__Q_463.CLRN $1i64\6i2\CEXTRA2_47__Q_323.CLRN
| l $1i64\6i2\P1CNT_9__Q_461.CLRN $1i64\6i2\P2CNT_18__Q_464.CLRN
| l $1i64\6i2\P2CNT_16__Q_462.CLRN $1i64\6i2\P1CNT_11__Q_456.CLRN
| l $1i64\6i2\STOPEQ_24__Q_470.CLRN $1i64\6i2\P2CNT_19__Q_465.CLRN
| l $1i64\6i2\CWRITE3_52__Q_332.CLRN $1i64\6i2\CWRITE1_50__Q_330.CLRN
| l $1i64\6i2\CWRITE2_51__Q_331.CLRN $1i64\6i2\CH1DAV__DIN.CLRN
| l $1i64\6i2\BVAL3_34__Q_320.CLRN $1i64\6i2\BVAL3__DIN.CLRN
| l $1i64\6i2\P2CNT_20__Q_466.CLRN $1i64\6i2\P1CNT_8__Q_460.CLRN
| l $1i64\6i2\P1CNT_13__Q_458.CLRN $1i64\6i2\RES1__CLR
| l $1i64\6i2\P1CNT_10__Q_455.CLRN $1i64\6i2\RESB2__CLR

```

