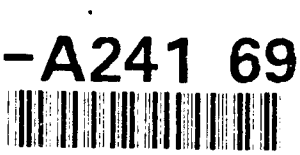


ANNUAL REPORT

VOLUME 6

TASK 6: GN&C PROCESSOR TEST & EVALUATION

REPORT NO. AR–0142–91–002

September 24, 1991

GUIDANCE, NAVIGATION AND CONTROL

DIGITAL EMULATION TECHNOLOGY LABORATORY

Contract No. DASG60–89–C–0142

Sponsored By

The United States Army Strategic Defense Command

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30322–0540

Contract Data Requirements List Item A005

Period Covered: FY 91

Type Report: Annual

REPORT DOCUMENTATION PAGE

Form Approved
OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT 1)Approved for public release; distribution is unlimited 2) continued on reverse side
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S) AR-0142-91-002	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION School of Electrical Eng. Georgia Tech 6b OFFICE SYMBOL (If applicable)	7a NAME OF MONITORING ORGANIZATION U.S. Army Strategic Defense Command
6c ADDRESS (City, State, and ZIP Code) Atlanta, Georgia 30332	7b ADDRESS (City, State, and ZIP Code) P.O. Box 1500 Huntsville, AL 35807-3801
8a NAME OF FUNDING/SPONSORING ORGANIZATION 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER DASG60-89-C-0142
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO / PROJECT NO / TASK NO / WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
Guidance, Navigation and Control Digital Emulation Technology Laboratory
Volume 6 (Unclassified)

12 PERSONAL AUTHOR(S)
C. O. Alford, Wei Siong Tan

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Annual	FROM 9/28/90 TO 9/27/91	9/27/91	86

16 SUPPLEMENTARY NOTATION

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

1. Introduction
2. Strategy
3. Testing and Evaluation
   - 3.1 GT-DP Module
   - 3.2 GT-VSM8 Test Board
   - 3.3 GT-VFPU Test Board
   - 3.4 GT-EP Evaluation Board
   - 3.5 GT-SPOBJ
   - 3.6 GT-DP/PFP Test Board
4. Summary & Assessment
5. Test Schedule

Appendix A: GAL Listing
Appendix B: GAL Listings
Appendix C: Test Monitor Source Code
Appendix D: CUPL Listing
Appendix E: Board Schematics
Appendix F: Board Schematics
Appendix G: GT-DP/PFP Board Schematics

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code) / 22c OFFICE SYMBOL

DD Form 1473, JUN 86 — Previous editions are obsolete — SECURITY CLASSIFICATION OF THIS PAGE

Distribution statement continued
2) This material may be reproduced by or for the U.S. Government pursuant to the copy license under the clause at DFARS252.227-7013, October 1988.

ANNUAL REPORT

VOLUME 6

TASK 6: GN&C PROCESSOR TEST & EVALUATION

September 24, 1990

Author

Wei Siong Tan

COMPUTER ENGINEERING RESEARCH LABORATORY

Georgia Institute of Technology

Atlanta, Georgia 30322–0540

Eugene L. Sanders

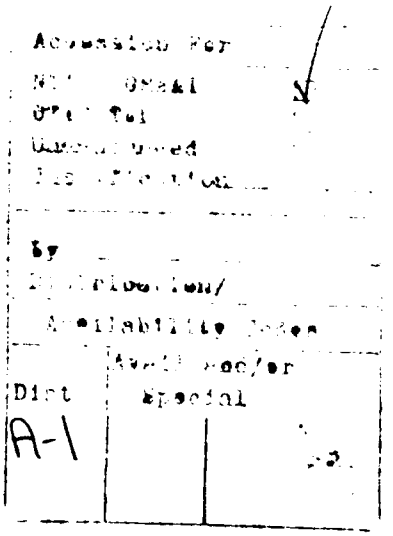
USASDC

Contract Monitor

Cecil O. Alford

Georgia Tech

Project Director



DISCLAIMER

The views, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other official documentation.

DISTRIBUTION CONTROL

(1) This material is approved for public release unlimited distribution.

TABLE OF CONTENTS

1. Introduction

CERL has developed a set of VLSI chip set for the guidance, navigation, and control of high the next generation interceptors for the Strategic Defense Intiative applications. The processor is called the GN&C processor and collectively the VLSI chip set is called the GN&C chip set.

The GN&C processor consists of three functional types: executive processor, signal processor, data processor, and interconnection network. The VLSI chips that had been developed for the executive processor are the GT–VIAG and GT–VDAG. The VLSI chips that had been developed for the signal processor are the GT–VNUC, GT–VTF, GT–VTHR, GT–VSF, GT–VCLS, and GT–VCTR. The VLSI chips that had been developed for the data processor are the GT–VSEQ, GT–VDR, and GT–VFPU. The VLSI chips that had been developed for the interconnection network are the GT–VSM8 and GT–VSNI. The GT–VFPU chip developed for the data processor is also used on the executive processor.

All of the GN&C chip set except the GT–VNUC and GT–VTF had been fabricated and tested. The GT–VNUC and the GT–VTF are currently being fabricated.

This document presents the test and evaluation methodology for the GN&C chip set, the GN&C processor board, and the development of special test board.

2. Strategy

As illustrated in Figure 1, the testing and evaluation of the GN&C chip set consists of four stages: VLSI testing, board testing, functional testing, and system testing.

The VLSI testing consists of two phases: simulation testing and packaged part testing. Before a chip is sent out for fabrication, test vectors are generated and applied to the simulation model of the VLSI chip. Two types of test vectors are generated. The functional test vectors, the first type, consists of a set of test patterns devised to test the functional modules in the VLSI chip under

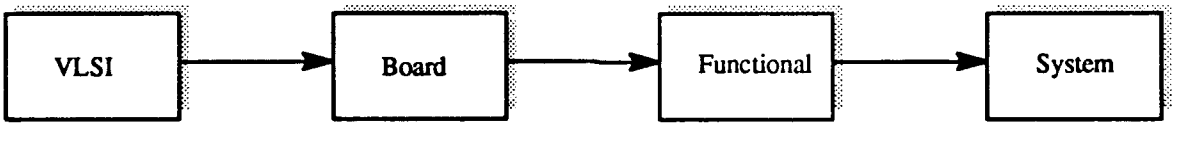


Figure 1. Testing and Evaluation Strategy

a normal functional operating condition. It also includes the testing of the interaction between the functional modules. The second type is the manufacturing test vectors. The manufacturing test vectors are designed to exercise all the transistor nodes so that manufacturing defects can be detected. The strategy for generating the manufacturing test vectors is to exercise all the functional modules of in a orderly fashion so that each changing is a state is can be detected in the output pins of the chip. There is no strict requirement of the fault coverage level. Typically, the manufacturing test vectors achieve close to a 90% coverage for the VLSI chips developed at CERL. The amount functional vectors depends on what the designer think is sufficient to cover the chip. Some kind of system level test requirement is imposed on the functional test vectors. This requirement is derived from system level simulation of the GN&C chip set and in some cases derived from the multichip simulation of some subset of the GN&C chip set.

The simulation model of the VLSI chip consists of a functional model and a gate level model. During the development stage, all simulation are performed on the functional model because the gate level simulation runs a lot slower. Prior to sending a chip out for fabrication, all the test vectors are run through the gate level simulation.

After a chip is fabricated, the same set of test vectors were applied to the packaged parts. The chip tester looks up the value of the input signal from the vector files and applies the appropriate stimuli to the input pins of the chip. At the proper interval dictated by the test vectors, the chip tester samples the signals on the output pins and compared them against the values in the test vector files. The passing of test vectors on the chip tester constitutes the condition of acceptance for the GN&C chipset from the manufacturer.

A printed circuit board typically contains several VLSI chips. The board testing verifies that the VLSI chips in a printed board operate correctly as a functional unit. Each printed circuit board for the GN&C chip typically represents a functional module of the GN&C processor. In some cases, an evaluation test board is assembled to specifically test the functionality a VLSI chip or chip set before the final printed circuit board module is developed.

The functional testing involves the integration and testing of several printed circuit board modules. The testing at this level is usually very rudimentary. For the executive and data processor, this step involves the loading and execution of simple program to verify the capability to perform basic computations and to interact with the operating environment through the I/O commands. For the signal processor, this step involves the programming and checking of the host port for programming the coeficients and the processing of simple images through the signal processing chip set.

The system level test consists of general application tests, diagnostic tests, and the mission specific application tests. The general application tests consist of a collection of numerically inten-

sive application and benchmark test to verify and evaluate the performance of the processor. The first objective of the diagnostic test is to determine whether the GN&C processor is functional. The second objective is to identify the faulty board and the faulty component. In many cases, the diagnostic test may not be able to identify the exact faulty part. It may simply identify a list of possible faulty components. The mission specific application test involves running a representative GN&C program. This includes interfacing the GN&C processor to a simulation testbed which simulates the interceptor dynamics and its surrounding environment.

3. Testing and Evaluation

As indicated earlier, all the VLSI chips except the GT–VNUC and GT–VTF chips had been fabricated and tested. The GT–VNUC and GT–VNUC chips are currently being fabricated. The latest two chips that had finished fabrication is the GT–VIAG and GT–VDAG chips. Both chips do not pass all the test vectors. The GT–VIAG chip experienced some problems with the RF[25] signal line. The errors on the signal line do not exhibit a deterministic patterns. The GT–VDAG chip experienced some problems with the R_eq_f_1, R_eq_f_2, S_eq_f_1, and S_eq_f_2 signals. The error patterns indicated that there may have been an error in the Genesil comparator block. Instead of the propagational, the error patterns suggest that there might be a PHASEB latch in the comparator block. Fortunately, this does not cause those signals are available in PHASEA and it should not matter that the signals are sampled on PHASEA. One possible concern is that this constraint might stretch out the timing of PHASEA. Both of these problems are currently being investigated.

To date 6 GN&C processor and test boards had been built. In the following sections, the test and evaluation of each of these board will be given.

3.1. GT–DP Module

As shown in Figure 2, the GT–DP module consists of four VLSI chips: GT–VDR, GT–VSEQ, GT–VFPU, and GT–VSNI. The board is 6.3" x 1.8" in size. The board is a generic module that can be plugged into a mother board with a connection to a host computer. The GT–DP module had been tested on two separate mother boards. The first mother board has a Multbus I host port and a GT–VSM8 chip (see section 3.2). It is can accomodate two GT–DP modules. The second mother board has a GT–EP host port and a PFP crossbar interface port (see section 3.6). It can accomodate one GT–EP module.

The initial testing of the GT–EP module was done using the mother board with the Multibus I port. A PC–AT host computer is used to communicate with the GT–DP module(s). Two off–the–shelf interface boards are used to connect the PC–AT bus and the Mtibus I bus.

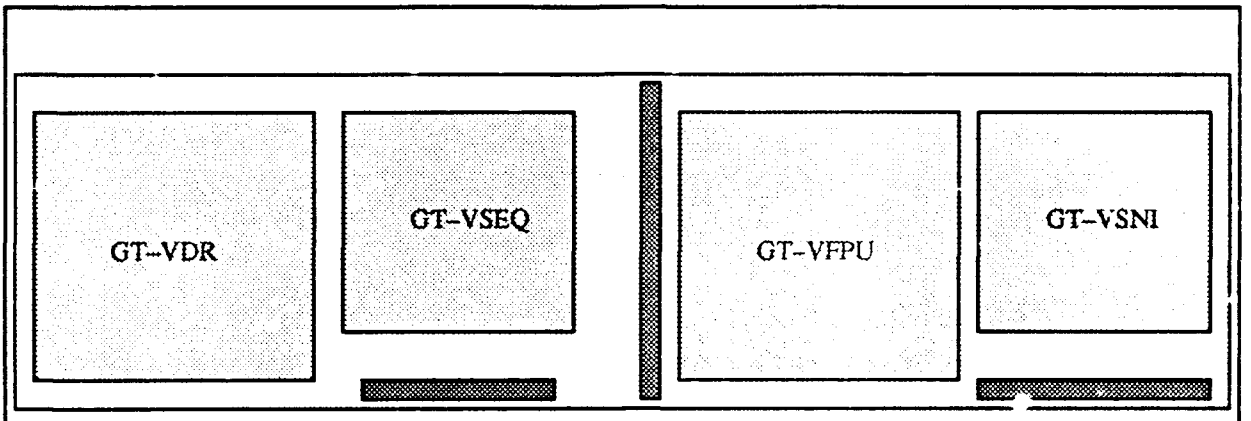


Figure 2. GT-DP Module Layout

The software that had been developed for the GT-DP module are a compiler, a loader, and some host utilities. The compiler supports a subset of Pascal language. The data types supported are boolean, integer, real, and array of integer and real. The language constructs supported are if ... then ... else ... , repeat ... until ..., while ... do, begin ... end , procedure, and function. Arbitrary arithmetic and boolean expressions are supported for the above data types and language constructs.

The loader accepts the output produced by the compiler and down load code to the GT-DP instruction and data memory. For the most part, the loader is independent of the interface hardware used to connect the GT-DP processor. The loader invokes procedure calls in the host utility library to interact with the interface hardware.

The host utility library consists of a set of subroutines for the communication between the GT-DP processor and the host. For example, the *start_system* and *stop_system* procedures allows a program to stop and start the GT-DP processors in the system. The test_DR and test_SEQ invokes a sequence of tests to verify the functionality of the GT-VDR and GT-VSEQ chips. The loader also includes functions/procedures for the host to communicate interactively with the GT-DP processor. Interactive communication means that the host processor is running a program, the GT-DP processor is running a program, and the two processors are exchanging data over the interface board.

The application programs that had run on the GT-DP processor are a 7th order differential equation system for satellite attitude control, a Mandelbrot fractal image generation program, and a simple target selection algorithm in hardware-in-the-loop simulation with the Parallel Function Processor testbed.

All four chips in the GT-DP have been functioning as expected. The processor is tested to run at 10 Mhz. The Genesil Timing Analysis estimates the operating frequency at 6.6 Mhz.

3.2. GT–VSM8 Test Board

The GT–VSM8 test board is a Multibus I wire–wrapped board with a GT–VSM8 chip, 3 buffer chips, two EPLDs, 1 hex switch, and 2 TTL clock generators. It has two slots for the GT–DP processor modules. One TTL clock generator sets the transfer rate for the GT–VSM8 chip. The other TTL clock generator sets the operating frequency for the GT–DP processors. The hex switch selects board space based the address value of /ADR13..0 for the Multibus I. The block diagram of the GT–VSM8 test board is shown in Figure 3. This test setup is referred to as the GT–DP Multibus I system.

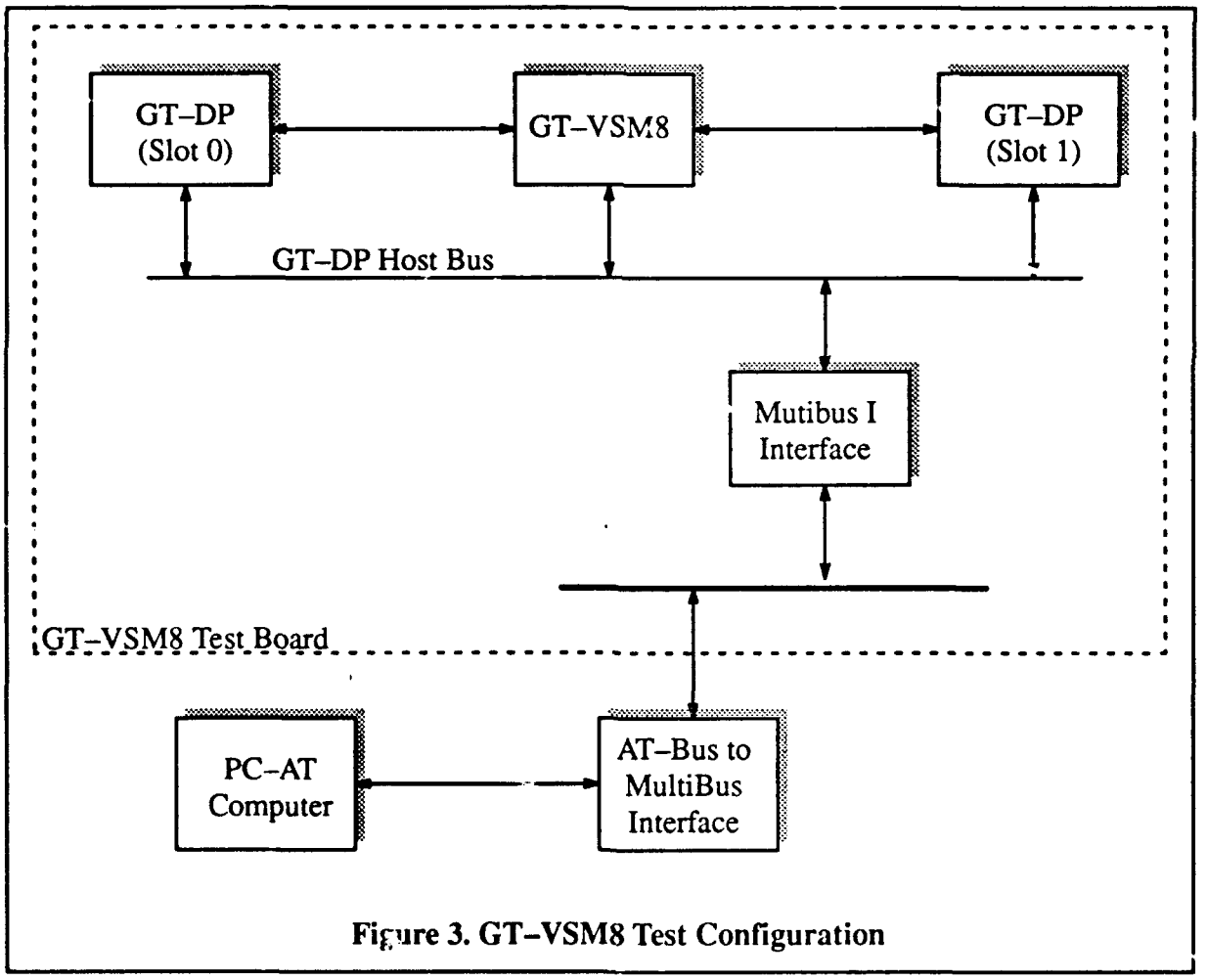


Figure 3. GT–VSM8 Test Configuration

The board had been used successfully to download programs to the GT–DP modules. The communication between two GT–DP modules have also been tested through the on–board GT–VSM8 chip.

3.3. GT–VFPU Test Board

The GT–VFPU test board was developed to characterize the operating speed of the GT–VFPU chip. This document presents the design of the test board, the testing strategy, and the test results.

3.3.1. Board Design

The architecture of the GT–VFPU test board is shown in Figure 5. The board was designed on a Multibus I board. A Multibus I to PC–AT interface board is used to connect the test board to a PC–AT host. The clock that drives the GT–VFPU chip is connected externally to a function generator. The power to the GT–VFPU chip is decoupled from the Multibus power plane. It is connected to an external power supply. The board design schematics are shown in Appendix F. The source code listing for the programmable logic chips are shown in shown in Appendix A and the test monitor program source code listing is shown in Appendix B.

The test software running on the PC Host generates a set of test vectors for each opcode category and downloads them into the memory unit that stores the stimuli for the GT–VFPU. Once instructed to execute, a run–time controller fetches the stimuli sequentially from each of the memory locations. For each stimuli, GT–VFPU computes the result and stores it in a second memory unit. The test software on the PC computes the correct results to be expected and compares this result with the result generated by the GT–VFPU. The memory units are capable of generating and capturing 4096 vectors in a single run.

The test board schematic design is included in Appendix A. The source programs for the on–board GALs are listed in Appendix B. The source code listing of the test software is listed in Appendix C.

3.3.2. Test Monitor

A test monitor running on the PC–AT host was developed to control the GT–VFPU test board. It consists of 1,530 lines of Turbo Pascal source code. The available commands are:

tmem : test memory;
tlog : test xor/and/or/passR/not R/not S;
tiadd : test integer add/sub/rsub;
timult : test integer mult
tfadd : test floating point add
tfmult : test floating point mult
tshift : test ROR/ROL/SHR/SHL
tspec0 : test pack exp & float

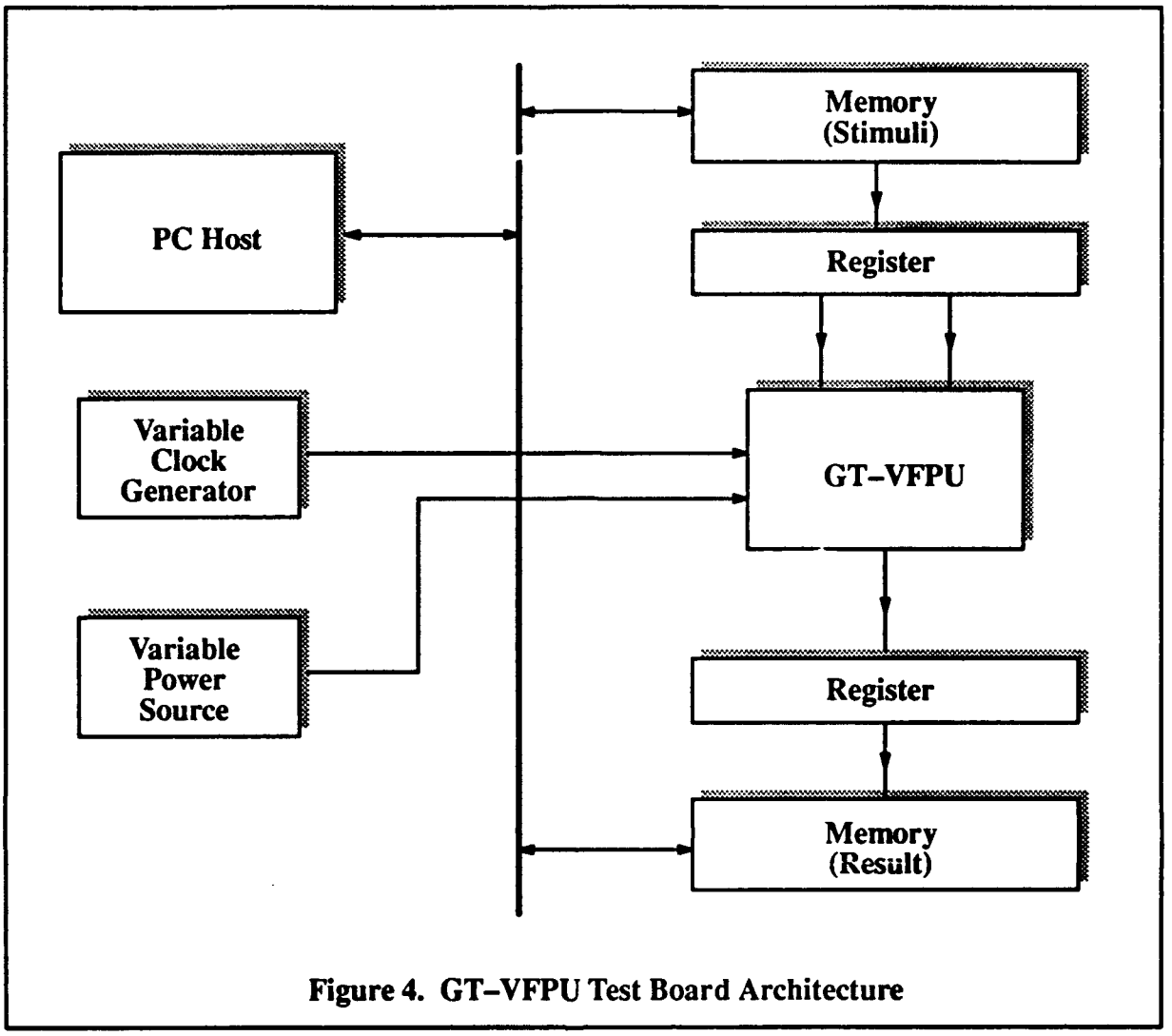


Figure 4. GT–VFPU Test Board Architecture

tspec1 : test seed, unp_exp, unp_man, root_exp, & root_man

tspec2 : test round & trunc

tspec3 : test sign manipulation

tall : test all of the above

dsoe : do not stop on error

soe : stop on error

start : start testing

stop : stop testing

sb : select memory bank

dw : display memory word

sw : substitute memory word

cont : set testing mode to continuous

single : set testing mode to single
debug : toggle debug setting
quit : quit FPU Test Monitor.

Each of the above commands can be invoked from the monitor.

3.3.3. Test Strategy

The GT–VFPU opcodes are divided into five broad categories: logical, shift, fixed–point, floating–point, and special. The stimuli for the GT–VFPU consists of the signals R[31..0], S[31:0], and Opcode[4:0]. Two vector sets were used to provide a combination of stimuli for each opcode category.

The first set consists of test vectors generated from a fixed set of patterns devised for each opcode category. Three arrays are used to store the primary test patterns for R[31:0], S[31:0], and Opcode[4:0]. Three, three–level–nested loops are used to generate the test patterns for different combinations of R[31:0], S[31:0]. amd Opcode[4:0]. Each nested loop places the index of the R[31:0], S[31:0], and Opcode[4:0] arrays in the inner loop. The purpose is to generate a sequential set of patterns that toggle the different sources of input stimuli to the GT–VFPU on a per cycle basis. To insure that the opcode bit fields are toggling every cycle, each odd storage element for the Opcode[4:0] array is stored with an inverted value of the opcode of the preceeding even storage element.

The second vector set consists of random patterns for R[31:0] and S[31:0].

The fixed patterns for each opcode categories are given in the following sections.

3.3.3.1. Logical Operations

The logical operations are passR, and, or, xor, not R, and not S. The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are $00000000, $ffffffff, $555555555, $aaaaaaaa, $ffffffff, $000000000, $12345678, and $9abcdef0. The pattern $00000000 is repeated twice to create $ffffffff to $00000000 and $00000000 to $ffffffff transistions.

3.3.3.2. Shift Operations

The shift operations are shift left, shift right, rotate left, and rotate right. The fixed patterns used for this test for the R[31:0] array are $00000000, $ffffffff, $555555555, $aaaaaaaa, $ffffffff, $000000000, $12345678, and $9abcdef0. Tne fixed patterns for the S[31:0] array are $00000000, $00000001, $00000010, ..., $0000000f.

3.3.3.3. Fixed–point Operations

The fixed–point operations are addition, subtraction, reverse subtraction (–R[31:0] + S[31:0]), and multiplication. The Multiplication is tested separately.

3.3.3.3.1. Addition/Subtraction/Reverse Substraction

The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are $00000000, $00ffffff, $005555555, $00aaaaaa, $00ffffff, $000000001, $00123456, $00abcdef, $80ffffff, $805555555, $80aaaaaa, $80ffffff, $800000001, $80123456, and $80abcdef.

3.3.3.3.2. Multiplication

The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are $000000000, $00ffffff, $00aaaaaa, $00555555, $00000fff, $00000aaa, $00000555, $00ffffff, $80000000, $80ffffff, $80aaaaaa, $80555555, $80000fff, $80000aaa, $80000555, $80ffffff.

3.3.3.4. Floating–Point Operations

The floating–point operations are addition, subtraction, reverse subtraction, and multiplication. The multiplication is tested separately.

3.3.3.4.1. Addition/Subtraction/Reverse Subtraction

The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are $3f000000, $3fffffff, $3faaaaaa, $3f555555, $3f000001, $7f000000, $2a800000, $55000000, $00800000, $bf000000, $bfffffff, $bfaaaaaa, $bf555555, $bf00001, $ff000000, $aa800000, $d5000000, $80800000.

3.3.3.4.2. Multiplication

The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are $00000000, $3fffffff, $3faaaaaa, $3f555555, $00000fff, $00000aaa, $7f000555, $7fffffff, $08000000, $bfffffff, $bfaaaaaa, $bf555555, $80000fff, $80000aaa, $ff000555, $ffffffff.

3.3.3.5. Special Operations

The special operations are pack exponent, float, inverse seed, unpack exponent, unpack mantissa, square root exponent seed, square root mantissa seed, round, trunc, sign of sine, odd to negative, change sign, and sign of tan. The testing of these operations are separated into four groups.

3.3.3.5.1. Pack exponent/Float

The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are $00ffffff, $80000001, $80000004, $80000010, $80000040, $80000100, $80ffffff.

3.3.3.5.2. Inverse Seed/ Unpack Exponent/ Unpack Mantissa/ Sqrt Seed

The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are $00000000, $3fffffff, $00000fff, $3f000555, $08000000, $bfffffff, $80000fff, $ff000555.

3.3.3.5.3. Round/Trunc

The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are $00000000, $3fffffff, $00000fff, $3f000555, $4b000000, $bfffffff, $c6000fff, $bf000555.

3.3.3.5.4. Sign of Sine/ Odd to Negative/Change Sign/ Sign of Tan

The fixed patterns used for this test for the R[31:0] and S[31:0] arrays are $00000000, $00000001, $80000010, $80000001, $3f800000, $3f800001, $bf800010, and $bf800001.

3.3.4. Test Result

All Tests were done at room temperature (~75 deg F). Three voltage test conditions were applied to the GT–VFPU chip. The test results are shown in Table 1.

Table 1. Maximum Operating Frequency Characterized by Opcode Category

Opcode Category	4.5 V @ 150 mA	5.0 V @ 190 mA	5.5 V @ 230 mA
logical	19	19	19
Fixed–Point Add	18	19	19
Fixed–Point Mult	19	19	19
Floating–Point Add	16	18	18
Floating–Point Mult	17	19	18
Shift	19	19	18
Special 0	17	18	18
Special 1	17	18	19
Special 2	15	17	19
Special 3	19	19	19
Max/Min	19/15	19/17	19/18

3.3.5. GT–VFPU Test Summary

The GT–VFPU test board demonstrated that the Genesil timing and power analysis were very conservative. Genesil predicted a maximum operating frequency of 6.6 Mhz and a power consumption of 5.1 W for the GT–VFPU chips. The test result shows that the GT–VFPU operates at 17 Mhz and consumes only 950 mW with a 5.0 V supply. The highest operating frequency attained is 19 Mhz. This limit may have been imposed by the test board.

3.4. GT–EP Evaluation Board

The GT–EP evaluation test board was developed as a platform for :

1. evaluating and testing the GT–VIAG and GT–VDAG chips at the board level,

2. developing and testing the GT–EP software, and
3. measuring and evaluating the performance of the GT–EP processor.

3.4.1. Board Design

3.4.1.1. Host Interface

The GT–EP evaluation board is connected to a Sun host through a GT–APVME interface board (see Figure 5). The GT–EP communicates with the GT–APVME board through a high speed 32–bit synchronous AP–Bus. The GT–APVME in turn communicates with the Sun host through the VME bus. The GT–EP is a board module on top of the GT–APVME mother board. This arrangement isolates the host from the processor module and simplifies the hardware and software testing of the host interface. Appendix E provides the schematics of theboard design.

Figure 5. GT–EP Host Interface

3.4.1.2. Processor Functional Blocks

Figure 6 shows the high level block diagram of the GT–EP evaluation board.

The EP–Bus connects the various functional components of the GT–EP processor. Electrically, the GT–EP bus is not strong enough to interface directly with GT–APVME board. The AP–Bus buffers the EP–Bus to the GT–APVME board.

The following sections describe the functionality of each block. Appendix C provides the source code listings of the programmable logic chips.

3.4.1.2.1. Instruction Memory

The instruction memory uses four CYM1831 or CYM1841 memory modules. Each CYM1831 consists of 64k x 32 bits and each CYM1841 256kx32 bits. Initially the CYM1831 memory modules will be used. They can be replaced with CYM1841 memory modules if additional instruction memory is required.

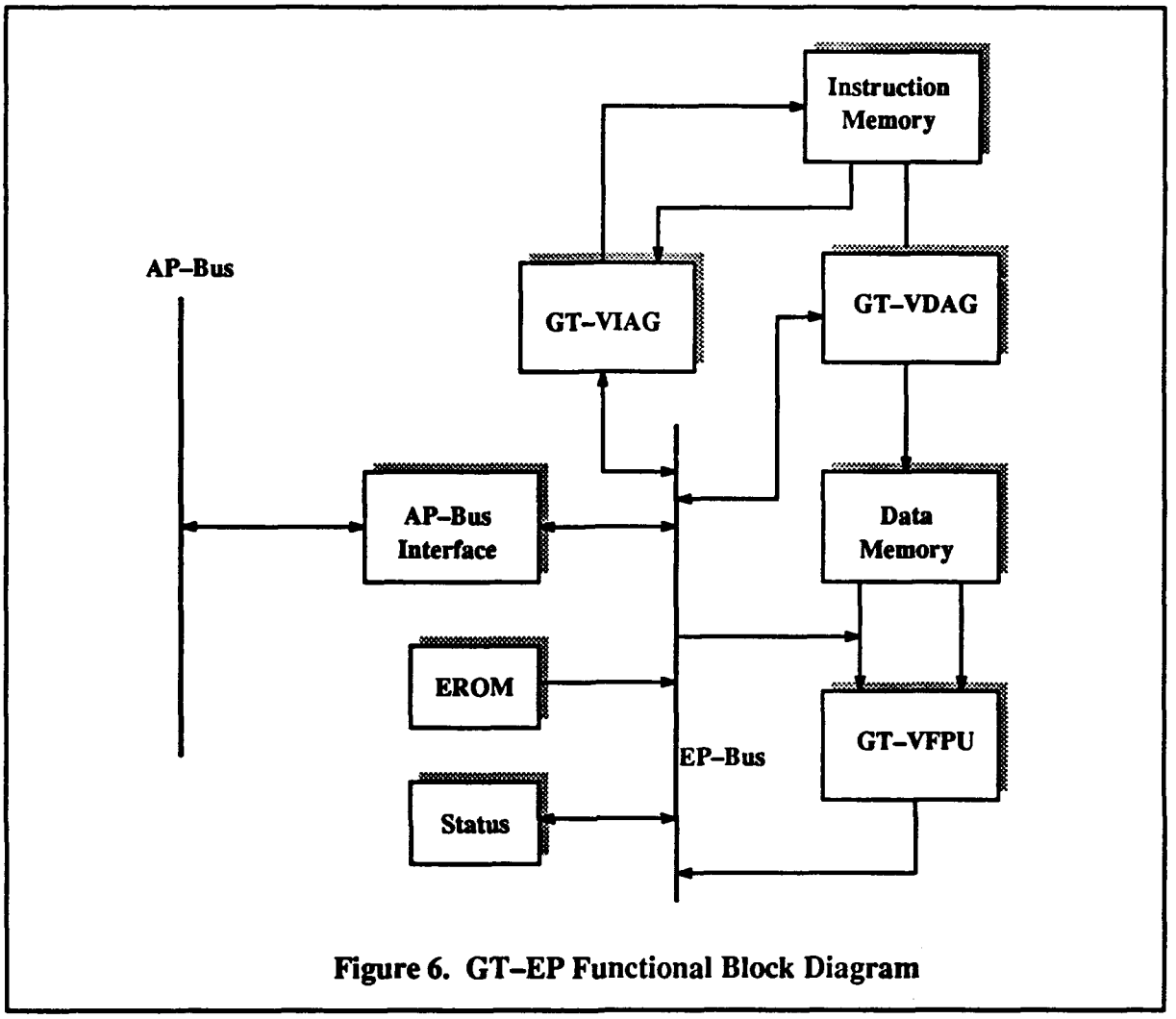


Figure 6. GT–EP Functional Block Diagram

A set of buffers are used between the GT–VIAG and the memory modules for the control and address lines. This is necessary because of the capacitive loading on the address lines and control lines to the memory modules.

The instruction memory supplies only 22 of 26 bits for the pc_min_op field, 24 of 26 bits for the r_off field, 25 of 26 bits for the f_off field, and 22 of 26 bits of the s_off field. The remaining bits of the instruction fields are connected to ground through a set of pull–down resistors. This poses a problem when a negative offset with respect to the address pointer is required. In this case, the f_off, r_off, and s_off fields need to be sign extended by connecting f_off[23] to f_off[25:24], r_off[23] to r_off[25:24], and s_off[21] to s_off[25:22]. To reduce the capacitive loading on f_off[23], r_off[23], and s_off[21], the sign extend can be spread across f_off[22], f_off[23], r_off[22], r_off[23], s_off[18], s_off[19], s_off[20], and s_off[21]. Each of the signals would then have to drive only two signal loads. This scheme works only if 17 bits of address information are

needed as is the case for the CYM1831 or CYM1841 memory modules. The address mapping to the 24–bit AP–Bus is not affected because relative addressing is not used on the AP–Bus.

Also in the instruction memory schematic section are three LEDs for conveying the GT–EP status information. They indicate the kernel status, the freeze condition, and the GT–EP bus activities on the AP–Bus.

3.4.1.2.2. GT–VIAG

The GT–VIAG chip is primarily responsible for generating the instruction address for the instruction memory.

The Intr1 signal to the GT–VIAG is connected to the Bus_err signal on the AP–Bus so that an interrupt can be generated when an access is performed on the AP–Bus memory address that is not mapped onto a physical device. Intr0 and Intr7..2 are mapped directly to Intr0 and Intr7..2 on the AP–Bus.

3.4.1.2.3. GT–VDAG

The GT–VDAG VLSI chip generates data addresses, SF_adr and RF_adr, for the data memory.

The SF_addr field is 17 bits and controls only the data memory. The RF_addr field is 24 bits and controls the data memory and the AP–Bus.

3.4.1.2.4. Data Memory

The data memory uses two memory modules that are of the same type as the ones used for the instruction memory. The data memory size is either 64kx32 or 256kx32 depending on which type of memory module is used. The two memory modules are written simultaneously. During a read, separate values can be fetched from the two memory modules.

Four 8–bit buffers are used to map the RF data bus onto the S data bus during a write. A 20–pin GAL is used to generate the control signals for the memory modules. Also coming out of this GAL are FPU_FRZ and FPU_OE signals which are used to control the GT–VFPU freeze and output enable signals.

3.4.1.2.5. EPROM

Four 2kx8 CY7C293 EPROM chips provide startup code for the GT–EP processor. The boot EPROMs in the GT–VIAG and the GT–VDAG chips enable the GT–EP processor to begin fetching data from device channel 3 and down loading them to the instruction memory for execution after a reset.

One 20–pin GAL and a 74393 counter chip supply the address to the EPROM chips. Each time a read is performed on device channel 3, the address value is incremented by one. The address is initialized to zero on a reset.

3.4.1.2.6. AP–Bus Interface

The schematics for the AP–Bus interface spread across four sheets: control, data, address, and connector.

The control section consists of three 20–pin GAL chips. They are primarily used to generate control signals for the AP–Bus interface. The CLK and FPU_CLK signals are generated here. They are used to drive the GT–VIAG/GT–VDAG and GT–VFPU respectively. The CLK signal is one half the frequency of the AP_CLK. The FPU_CLK is also one half the frequency except during a freeze. If the GT_VIAG and the GT–VDAG chips are about to enter a freeze condition, the FPU_CLK is held low so that the internal states of the GT–VFPU do not change. This is necessary because the GT–VFPU expects a freeze signal to be valid on the falling edge of the clock whereas the GT–VIAG is not able to resolve the freeze signal until after the falling edge of the clock.

One of the GAL chips provides signals to drive three LEDs. The three signals are mapped onto the GT–EP write memory space on device channel 3. The signal, BUS_LCK, is passed onto the AP–Bus. It is set active when the GT–EP intends to retain master control of the AP–Bus.

The GT–EP interfaces with the AP–Bus with four control signals (Rd, Wr, EP_Brq, and EP_Mstr), 24–bit address, and 32–bit data. On a write operation, the GT–EP posts the write on the data register and proceeds before the AP–Bus actually completes the write cycle. On a consecutive write, however, the GT–EP will pause if the value on the data register has not been written out to the AP–Bus. On a read operation, the GT–EP waits until the data from the AP–Bus is valid.

The interrupt signals on the AP–Bus are active low. They are inverted before connection to the GT–VIAG chip. Intr 1 is generated by the GT–EP processor. The Intr1 to the GT–VIAG chip is connected to the Bus_err signal on the AP–Bus instead.

3.4.1.3. Device Mapping

The GT–EP device mapping is shown in Table 2. The write only registers in the last four entries are each a 1–bit register. Their contents are toggled each time a write is performed on the register. Upon reset, their values are initialized to zero.

Table 2. GT–EP Device Mapping

DS3..0	RF_ADR24	RF_ADR1..0	Device	Operations	Description
0	x	xx	GT–VIAG & GT–VDAG	Read & Write	Accessing load and store instructions
1	x	xx	Data Memory	Read & Write	Primary storage device
2	x	xx	AP–Bus	Read	AP–Bus Interface
2	1	xx	AP–Bus	Write	AP–Bus Interface
3	x	xx	ROM	Read only	Used during booting
3	1	00	LED0	write only	First general status indicator
3	1	01	LED1	write only	Second general status indicator
3	1	10	Intr1	write only	Generate an interrupt on the AP–Bus
3	1	11	Bus_lock	write only	Lock the AP–Bus for use by the GT–EP only

3.4.2. AP–Bus Interface Timing

Figure 7, Figure 8, Figure 9, and Figure 10 illustrate the interface timing between the EP–Bus and the AP–Bus. The AP–Bus operates at twice the frequency of the EP–Bus. Since the EP–Bus performs a write and a read operation in a single cycle, the bandwidth between the two buses matches almost exactly.

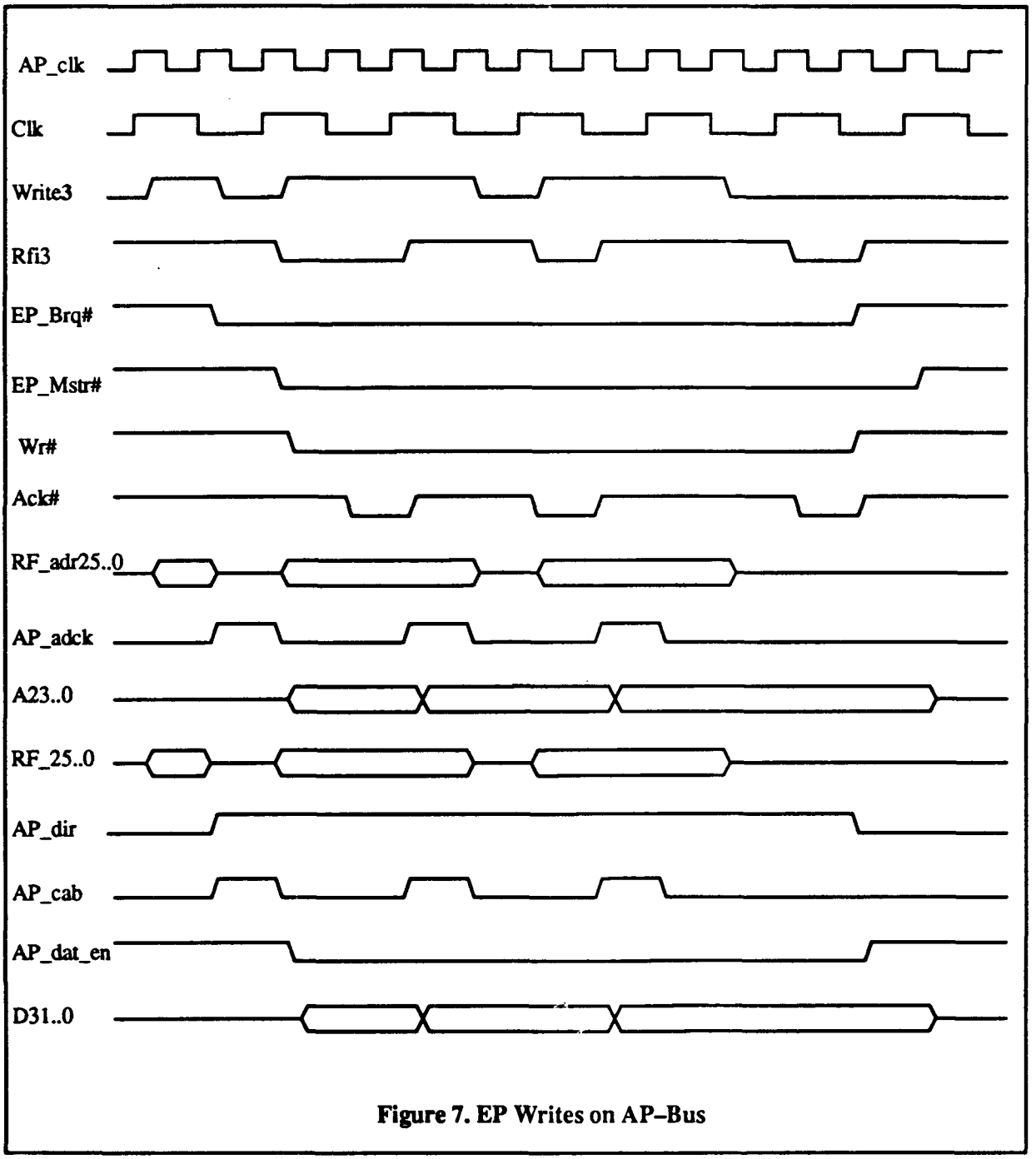


Figure 7. EP Writes on AP–Bus

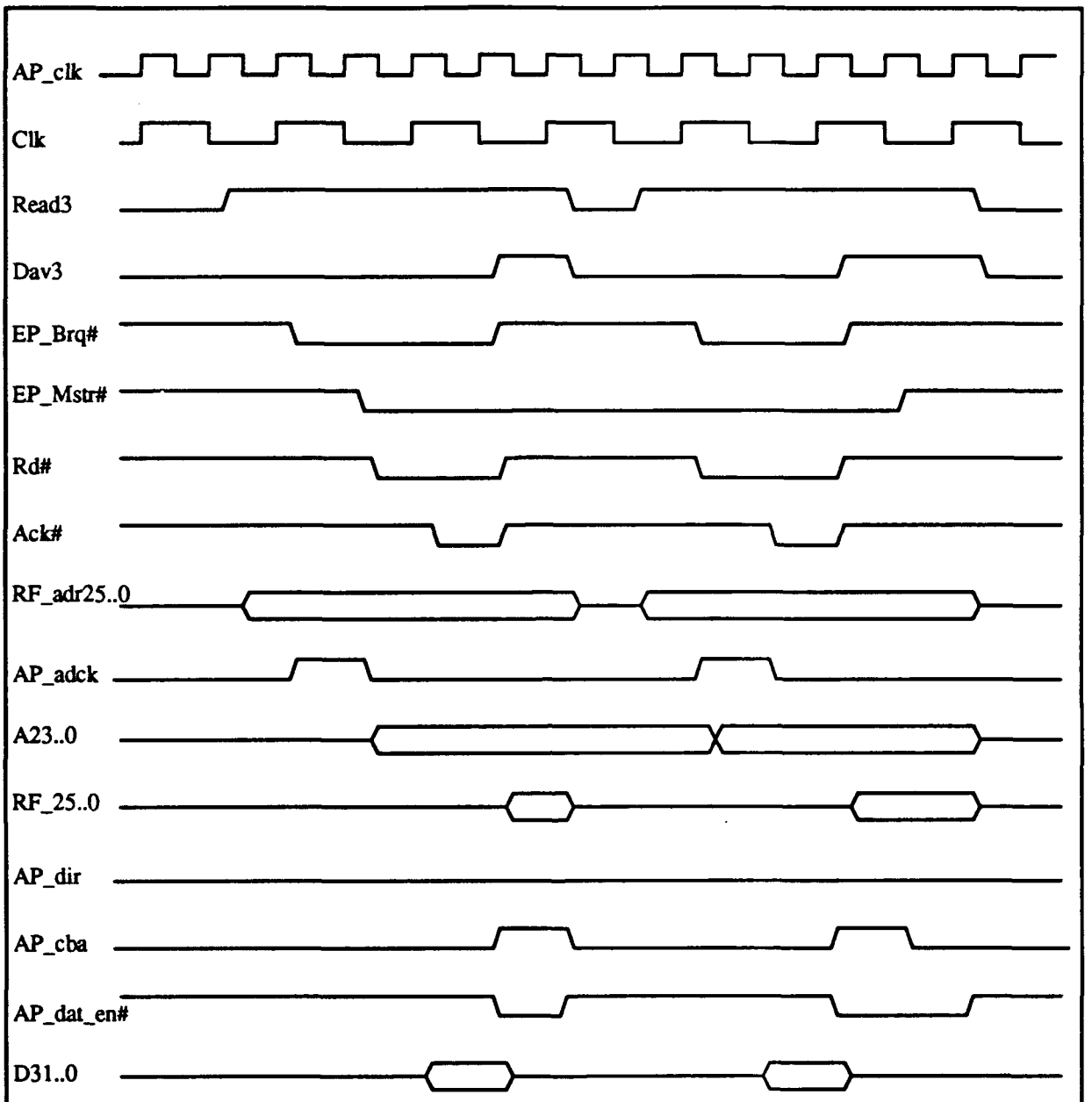


Figure 8. EP Reads on AP-Bus

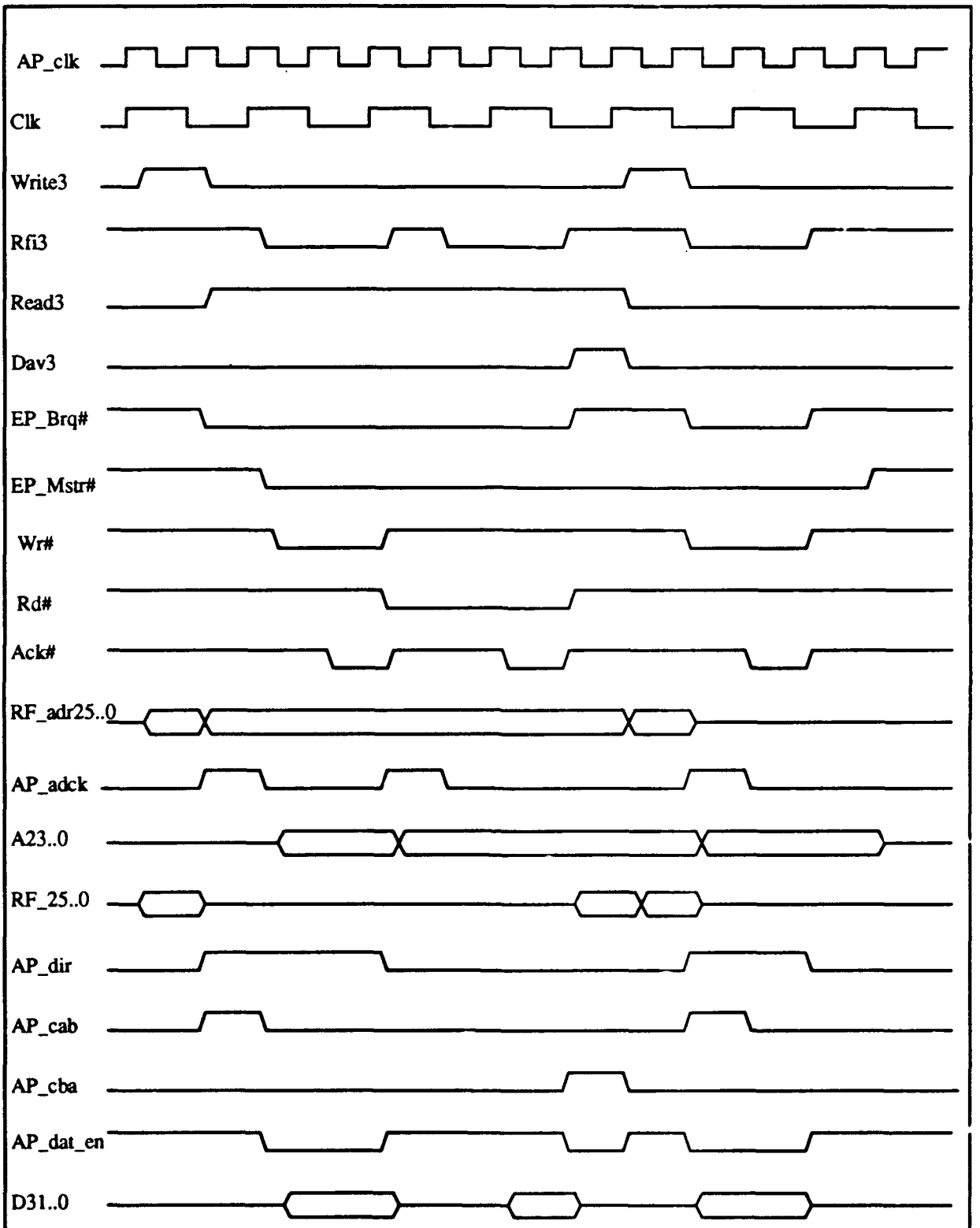


Figure 9. EP Write–Read–Write on AP–Bus

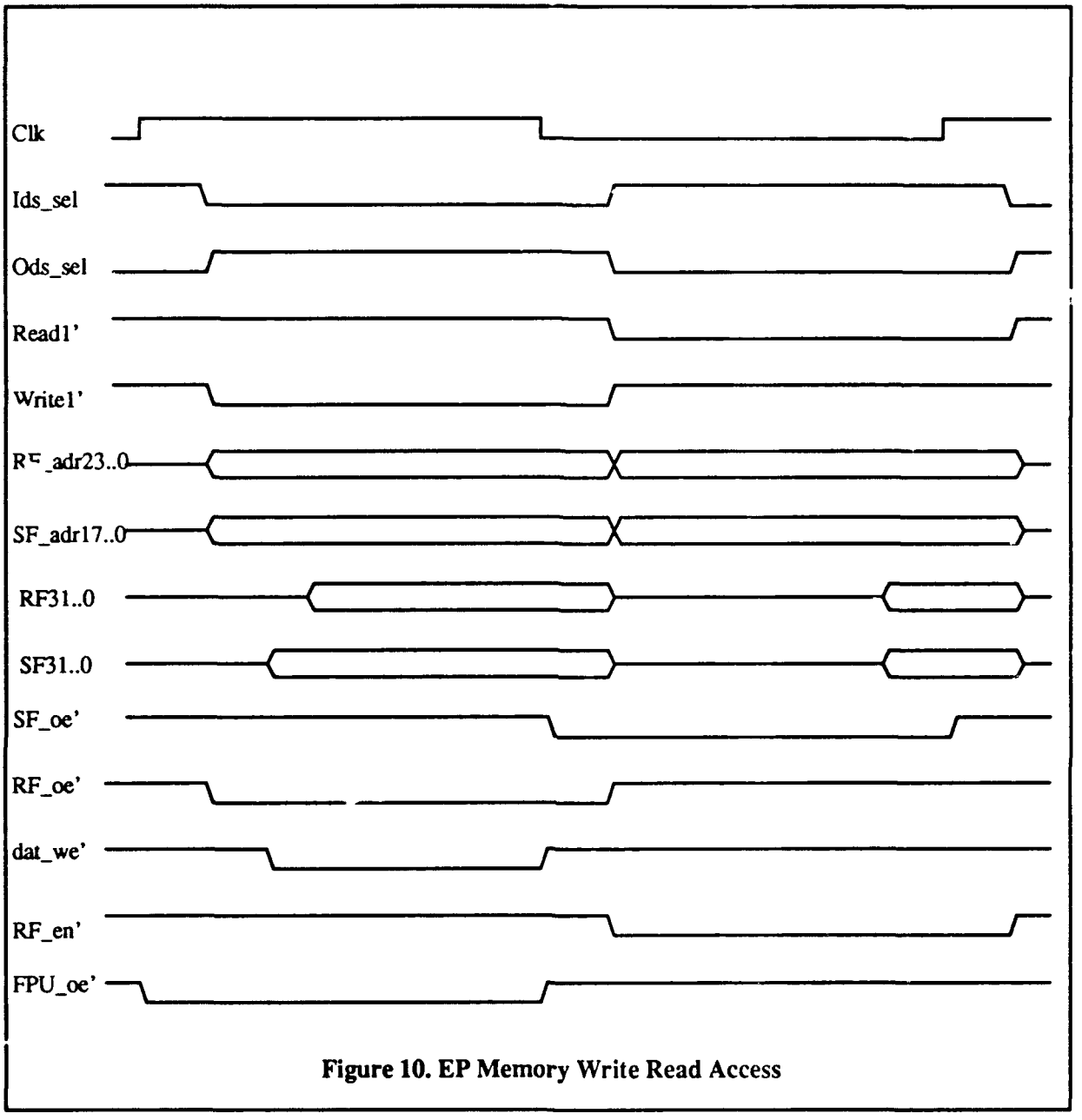


Figure 10. EP Memory Write Read Access

3.4.3. GT–EP Testing

Hitachi Multiwire had fabricated four multiwired boards. Their engineer(s) failed to connect the AP–Bus power and ground connectors to the power and ground planes, respectively on the boards. To begin testing immediately, Georgia Tech accepted one board and requested Hitachi to refabricate the other three to fix the problem. Testing on the GT–EP evaluation had been started. The folowing sections provides the details of the tests that had been done and outline an approach for the tests that are planned.

3.4.3.1. Functional Testiing

The first test is to fetch and execute instructions from the on–board EPROMs. Three programs had been successfully run and executed. Several netlist errors were found and corrected using jumper wires. The three test programs are described in the following sections.

3.4.3.1.1. LEDBAS

LEDBAS is a simple program that performs simple computations using the data memory and simple I/O operations on channel 3. This test verified that the GT–EP invokes the internal boot–ROM, fetches instruction from the EPROM, and executes the program correctly. The behavior of the GT–EP was observed using a logic analyzer.

3.4.3.1.2. LED

The LED program starts up the GT–EP processor and toggles the two on–board status LEDs at a two million count interval. This program confirms the GT–EP steady state operation. The behavior can be observed visually from the value of the LEDs, e.g. 00, 01, 10, 11. This program had ran succesfully.

3.4.3.1.3. APVME

APVME tests the basic write and read operations on the AP–Bus interface. This programs starts up the GT–EP processor, executes six writes on the AP–VME board memory, and executes six reads on the AP–VME board memory. The program execution is observed and verified on the logic analyzer.

3.4.3.1.4. LOADER

LOADER enables the GT–EP processor to communicate with the host through the GT–APVME board. It understands three basic messages from the host: load_inst, load_data, and start_program. With this three basic messages, the host can completely control the GT–EP processor including loading a more sophisticated loader if necessary. The LOADER program had been written and compiled. It has not been tested.

3.4.3.2. System Testing

3.4.3.2.1. Applications

It is desirable to run as many application programs as possible on the GT–EP processors. The programs that will definitely be run on the processor are the 7th order satellite attitude control system, Mandelbrot fractal image generation, solution of linear system, and π integration. Many

routines from the SPECMARK benchmark that can run at single precision will be tested on the GT–EP. Most importantly a collection of representative GN&C flight algorithms will be tested on the GT–EP.

3.4.3.2.2. Diagnostic

The GT–EP processor needs a diagnostic program to verify its full functionality. The major components of the GT–EP processor are the GT–VIAG, GT–VDAG, GT–VFPU, instruction memory, data memory, and I/O interface. The diagnostic program will consist of test routines to exercise each of the major components.

3.4.3.2.3. Mission Specific

The ultimate test for the GT–EP processor is the mission specific test. Under this test, the GT–EP needs to receive centroid data from GT–SPOBJ board, process the data, performs all necessary GN&C computations, and interacts with the surrounding environment simulated on the Parallel Function Processor.

3.5. GT–SPOBJ

The GT–SPOBJ board consists of a GT–VSF chip, a GT–VTHR chip, a GT–VCLS chip, and two GT–VCTR chips. The processing elements of the GT–SPOBJ and the sequence of operations are shown in Figure 11. The GT–SPOBJ is not a test board but an actual prototype board for the GN&C processor. Since the actual GT–EP processor board is not available yet, a PC–AT computer is used to host the testing through a custom–designed AT–Bus to EP–Bus interface board.

3.5.1. Board Design

The detail of the board design is described in Volume 5.

3.5.2. Testing

3.5.2.1. Host Port

The GT–VSF and GT–VTHR host port is used to program the chip coefficients and to control the chip operating mode. The testing of the host port are acomplished by writing and reading the patterns aaaa, 5555, ffff, 0000, and random patterns to all the coefficient registers.

The GT–VCLS chip does not require a host port. The host port of the GT–VCTR chip is used for reading object centroids detected on the pixel data streams. The GT–VCTR host port is tested in conjuction with the pixel port testing in the next section.

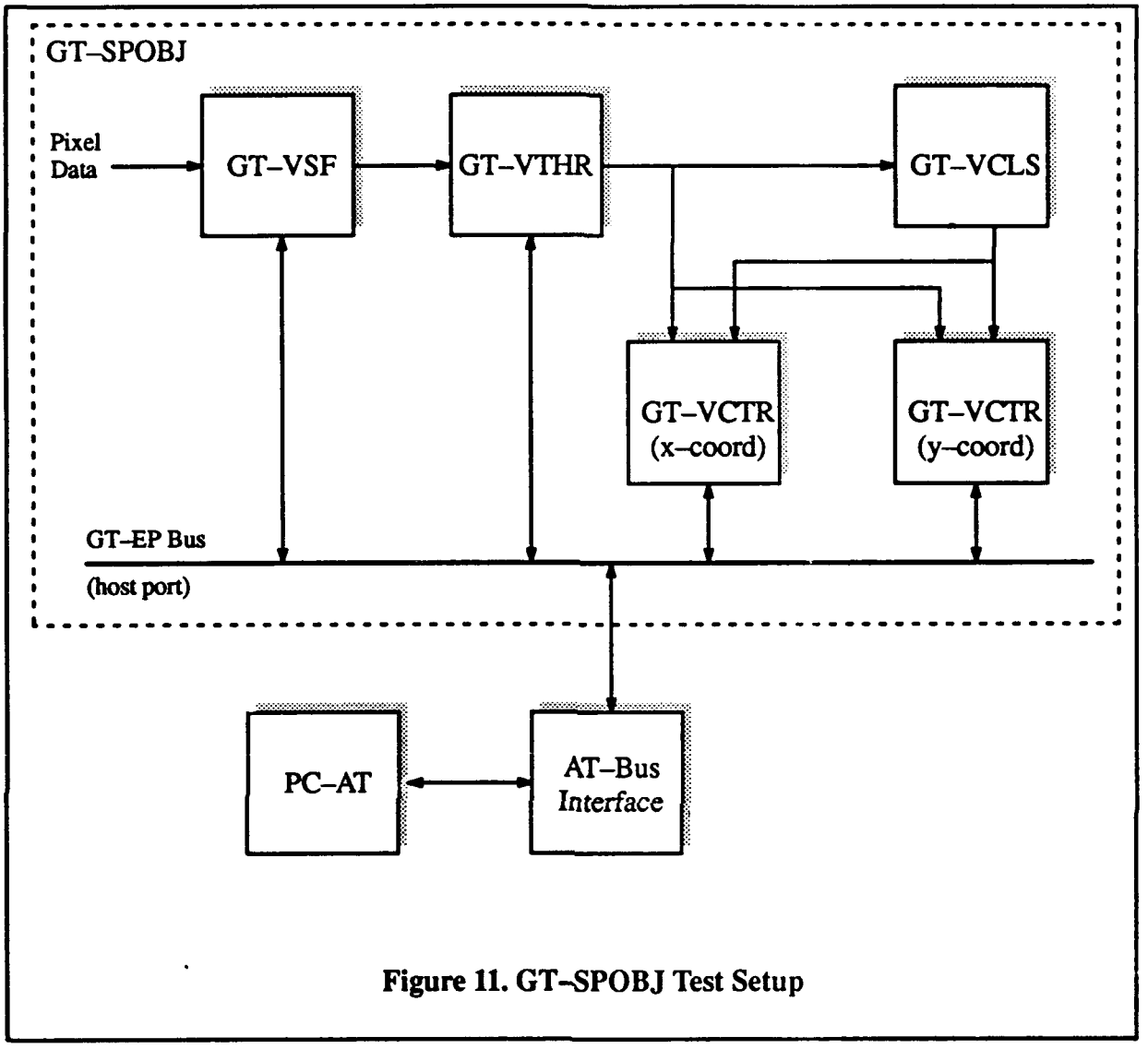


Figure 11. GT–SPOBJ Test Setup

3.5.2.2. Pixel Port

The pixel port is used to receive pixel data from the Focal Plane Array. In the test mode, the Seeker Scene Emulator is used to emulate the Focal Plane Array by generating pixel data for the GT–VSF chip. The processed pixel data from the GT–VSF is passed to the GT–VTHR which in turn passes processed data to the GT–VCLS and GT–VCTR chips. The functionality of the processing chain is verified by reading the centroids of each object in the Focal Plane Array image from the GT–VCTR host ports.

The GT–SPOBJ board had been tested in a hardware–in–loop simulation with the Parallel Function Processor, Seeker Scene Emulator, and the GT–DP processor. The simulation is running the terminal phase of EXOSIM. The Parallel Function Processor sends the missile line of sight information to Seeker Scene Emulator. The Seeker Scene Emulator generates Focal Plane Array

images to the GT–SPOBJ. The GT–DP processor selects a target and sends its centroid to the Parallel Function Processor.

3.6. GT–DP/PFP Test Board

The GT–DP/PFP board interfaces the EP–Bus to the GT–DP processor module and the PFP Crossbar. Two sets of FIFOs, one for each direction, are used for the communication between the EP–Bus and the PFP crossbar. The interface with the GT–DP processor is through memory mapping on the EP–Bus. The basic functional blocks of the GT–DP/PFP board are shown in Figure 12.

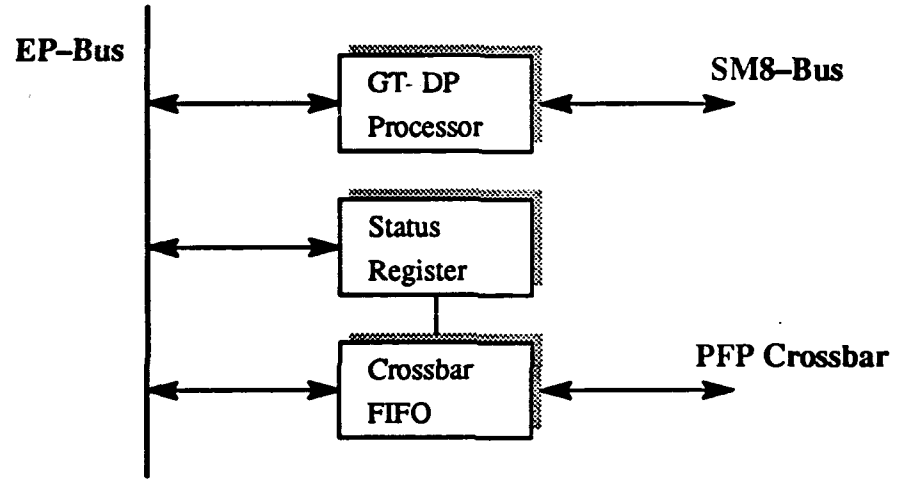


Figure 12. GT–DP/PFP Basic Functional Blocks

3.6.1. Design

3.6.1.1. EP–Bus Address Mapping

The GT–DP/PFP board is mapped as device 12 on the EP–Bus, e.g. DS3..0=12. The decoding of the EP–Bus address on the board is shown in Table 3.

Table 3. EP–Bus Address Mapping

A6	A5	A4	Device Selected	Valid Operations	Data Width
0	0	0	GT–VSNI chip	Read and Write	8 bits
0	0	1	GT–VDR chip	Read and Write	16 bits
0	1	0	GT–VSEQ chip	Read and Write	8 bits
0	1	1	PFP Crossbar	Read and Write	8 bits
1	0	0	Status Register	Read Only	14 bits

3.6.1.2. Interrupt Assignment

The interrupt assignment is shown in Table 4.

Table 4. EP–Bus Interrupt Assignment

Source	Interrupt	Description
APXB_EF#	Intr2	Data had arrived from the PFP

3.6.1.3. Status Register

The mapping of the status register is shown in Table 5.

Table 5. Status Register Listing

Source	AP–Bus	Description
APXB_FF#	D0	0 indicates that the outgoing FIFO to the PFP is full
APXB_EF#	D1	0 indicates that the incoming FIFO contains data from PFP

3.6.1.4. EPLD Control

Three G16V8 EPLD chips are used to control the GT–DP/PFP test board. A general description of each of the function of the EPLD is given in the following sections. The source code listings of the EPLD are given in Appendix D.

3.6.1.4.1. Xbar.pld

Xbar.pld is used to generate control signals for the PFP Crossbar interface.

3.6.1.4.2. Dec1.pld

Dec1.pld generates the chip select signals for the GT–DP processor. The chip select signals select the GT–VSEQ, GT–VDR, or GT–VSNI chip. In addition, Dec1.pld generates the read and

write control signals for the interface FIFO with the PFP and the device ready signal for the EP–Bus. Dec1.pld also allows the status of the interface FIFO to be read from the EP–Bus.

3.6.1.4.3. Dec2.pld

Dec2.pld controls the GT–DP/PFP board select signal and generates the read and write control signals for the GT–DP processor.

3.6.1.5. Board Schematics

The GT–DP/PFP board schematics is shown in Appendix G.

3.6.2. Testing

The functional modules of the GT–DP is mapped onto the EP–Bus address space. Programs were downloaded to the GT–DP board from a PC–AT computer, through the AT–Bus to EP–Bus interface board. The loader and utilities developed for the GT–DP Multibus I system (see sections 3.1 and 3.2) were ported to the GT–DP EP–Bus system. All the application programs that were run on the GT–DP Multibus I system were executed on the GT–DP/PFP board to verify its functionality.

The GT–DP/PFP serves as an interface point for GT–SPOBJ, GT–DP, GT–EP (emulated by a PC–AT computer through an interface board), and the Parallel Function Processor in the EXOSIM hardware–in–the–loop simulation. Prior to the hardware–in–the–loop simulation, patterns of aaaa, 5555, ffff, 000, and random numbers were transfered between the GT–DP/PFP and the Parallel Function Processor to verify that the interface hardware is working properly.

4. Summary and Assesment

The testing of the different modules of the GN&C processor is underway. The primary test focus right now is on the GT–EP evaluation board. Testing on the board shall proceed according to the plan outlined in section 3.4.3. Once the test result is available, the actual GT–EP board for the GN&C processor will be designed.

A printed circuit board for the GT–VTF and GT–VNUC chips had been designed. The board is now out for bid. It should be available when the two chips come out the chip fabrication and testing.

The testing of the GN&C chip set had been proceeding smoothly. All four chips for the GT–DP processor work functionally as expected. The operating frequency is actually much better than expected (from 6.6 Mhz to 10 Mhz). The GT–VSM8 network chip is working functionally as expected. The test, however, reveals an anomaly state on the GT–VSF chip when given an im-

proper sequence frame control signals. A reset does not entirely clear the chip to its power–up condition after the GT–VSF chip received a sequence of improper frame control signals. The test also reveals that the GT–VCTR chip requires the centroid data to be read consecutively without interruption. This is an unnecessary constraint that can be easily corrected. Both of these findings had been reported to AHAT and simple design modifications had been made on the GT–VSF and the GT–VCTR chips.

The GT–VIAG and GT–VDAG chips both reported errors while undergoing chip test. The error on the GT–VIAG chip had been verified on the GT–EP evaluation board to be not true. The chip is actually functioning properly. The problem might be due to an incorrect wiring on the IC test fixture. The error on the GT–VDAG had not been confirmed on the GT–VDAG chip because the problem would not occur under a normal operating condition. This issue is expected to be resolved as the testing on the GT–EP evaluation board is proceeding as planned.

A Focal Point Array frame buffer that interface directly to the EP_Bus had been developed. The frame buffer can hold up to four frames of 128x128 pixel data. The board had been fabricated and tested. The GT–SPOBJ board will be tested more extensively using the frame buffer.

In all, ten VLSI chips had been designed, fabricated, and tested. None requires a redesign and refabrication. Not all testings had been completed. Testing is still proceeding. The problem on the GT–VSF arised when an improper sequence of frame control signals was encountered. This situation does not occur under a normal operating condition. Nonetheless, a design fixed had been forwarded to the AHAT project. The problem on the GT–VCTR can be overcome in software by disabling interrupts while reading the centroid data from the GT–VCTR chips. A design fix had also been inserted into the AHAT chip design.

5. Test Schedule

Figure 13 shows the test schedule for the VLSI chip set that had been developed, namely, the GT–VSEQ, GT–VDR, GT–VFPU, GT–VSNI, GT–VSM8, GT–VSF, GT–VTHR, GT–VCTR, GT–VCLS, GT–VIAG, GT–VDAG, GT–VTF, and GT–VNUC. The next generation chip set is too early in the development phase to call for a detail test schedules to be included in this volume. A rough schedule is shown in volume 7.

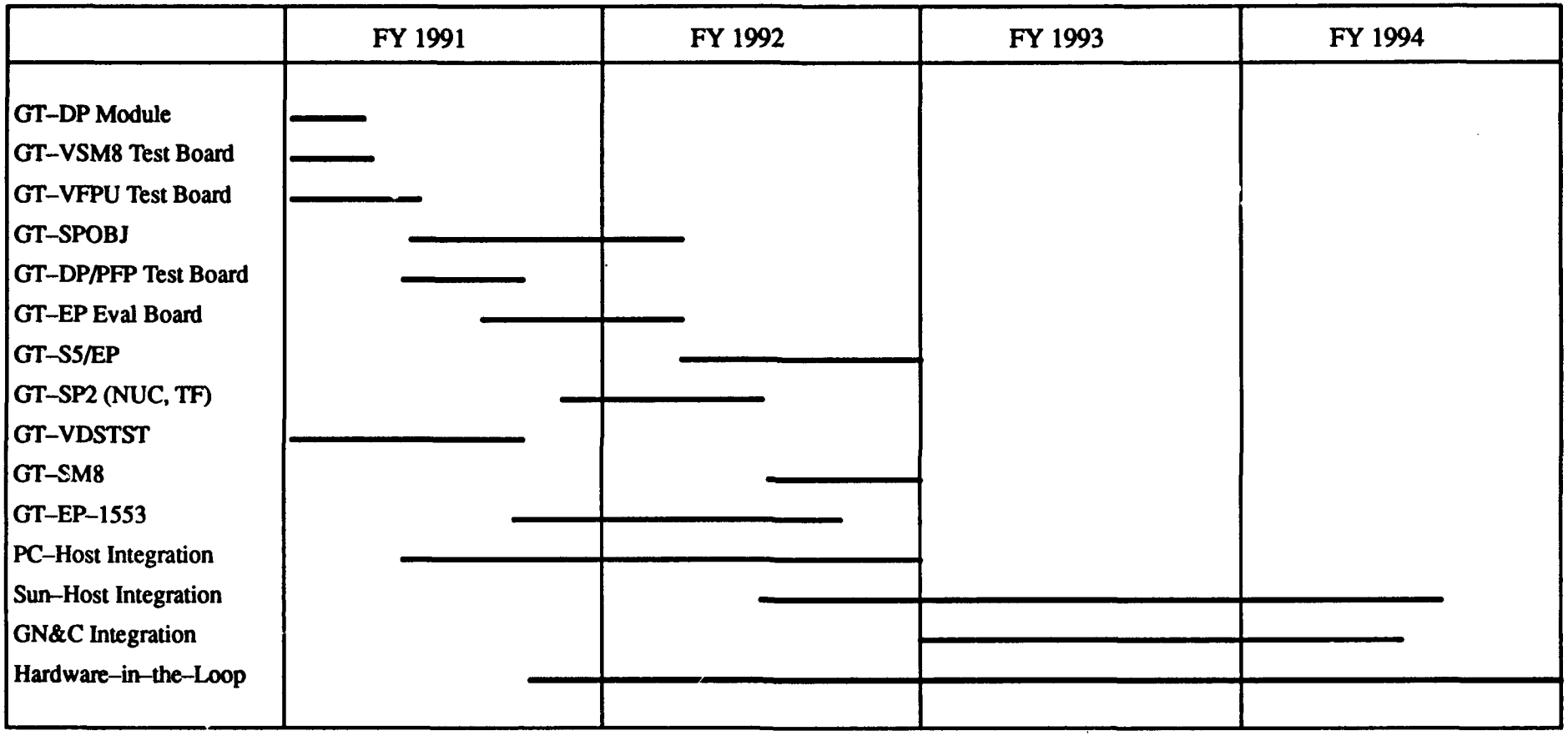


Figure 13. GN&C Processor Test and Evaluation Schedule

Appendix A : GAL Listings

```
NAME      Runctrl;
Partno    000;
Date      08/30/90;
Revision  0.0;
Designer  Dr. Tan;
Company   Cerl;
Assembly  FPU test board;
Location  Pal4;
Device    G16V8;

/* Input */

Pin 1  = clk_src;   /* osc */
Pin 2  = osc;
Pin 3  = !MWTC;
Pin 4  = !ADR_16;
Pin 5  = BD_0;
Pin 6  = !BS;

/* Output */

Pin 12 = !we0;
Pin 13 = clk;
Pin 14 = osc_1;
Pin 15 = run;
Pin 16 = n_run;
Pin 17 = osc_2;
Pin 18 = clk1;
Pin 19 = !we1;

/* Logic Equations */

run.d    =  (BS & MWTC & ADR_16 & BD_0) # ((!BS # !MWTC # !ADR_16) & run);
we0      =  MWTC & !run;
we1      =  (MWTC & !run) # ((!osc) & run );
osc_1    =  osc;
osc_2    =  osc_1;
n_run    = !run;
clk      =  osc_1;
clk1     =  osc_1;

NAME      Csctrl;
Partno    000;
Date      08/30/90;
Revision  0.0;
Designer  Dr. Tan;
Company   Cerl;
Assembly  FPU test board;
Location  Pal1;
Device    G16V8;  /* EP310 */

/* Inputs */

Pin 1  = !ADR_13;
Pin 2  = !ADR_14;
Pin 3  = !ADR_15;
Pin 4  = !ADR_16;
Pin 5  = clk1;
Pin 6  = !BS;
Pin 7  = run;

/* Outputs */

Pin 12 = !CS_0;
Pin 13 = !CS_1;
Pin 14 = !CS_2;
Pin 15 = !CS_3;
Pin 16 = !CS_4;
Pin 17 = !CS_5;
Pin 18 = !CS_6;
Pin 19 = !CS_7;
```

```
/* Logic Equations */

CS_7 = BS & !ADR_16 &  ADR_15 &  ADR_14 &  ADR_13 & !run # run;
CS_6 = BS & !ADR_16 &  ADR_15 &  ADR_14 & !ADR_13 & !run # run;
CS_5 = BS & !ADR_16 &  ADR_15 & !ADR_14 &  ADR_13 & !run # run;
CS_4 = BS & !ADR_16 &  ADR_15 & !ADR_14 & !ADR_13 & !run # run;
CS_3 = BS & !ADR_16 & !ADR_15 &  ADR_14 &  ADR_13 & !run # run;
CS_2 = BS & !ADR_16 & !ADR_15 &  ADR_14 & !ADR_13 & !run # run;
CS_1 = BS & !ADR_16 & !ADR_15 & !ADR_14 &  ADR_13 & !run # run;
CS_0 = BS & !ADR_16 & !ADR_15 & !ADR_14 & !ADR_13 & !run # run;
```

```
Name      Counter1;
Partno    000;
Date      08/30/90;
Revision  0.0;
Designer  Dr. Tan;
Company   Cer1;
Assembly  FPU test board;
Location  Pal6;
Device    G16V8;

/* Inputs */

Pin 1  = clk;
Pin 2  = cout;
Pin 3  = run;

/* Output */

Pin 12 = cout0;
Pin 13 = pc_0;
Pin 14 = pc_1;
Pin 15 = pc_2;
Pin 16 = pc_3;
Pin 17 = pc_4;
Pin 18 = pc_5;

/* Logic Equations */

pc_0.d  = (!pc_0 #  cout)  &  run;      /* original: cout1 */
pc_1.d  = ( pc_0 & !pc_1  #  !pc_0  &   pc_1  #  cout) &  run;
pc_2.d  = ( pc_0 &  pc_1 & !pc_2  # !(pc_0  &  pc_1) &  pc_2  # cout) & run;
pc_3.d  = ( pc_0 &  pc_1 &  pc_2  &  !pc_3  #
          !(pc_0 &  pc_1 &  pc_2) &   pc_3  #  cout) &  run;
pc_4.d  = ( pc_0 &  pc_1 &  pc_2  &   pc_3  & !pc_4  #
          !(pc_0 &  pc_1 &  pc_2  &   pc_3) &  pc_4  #  cout) & run;
pc_5.d  = ( pc_0 &  pc_1 &  pc_2  &   pc_3  &  pc_4  & !pc_5  #
          !(pc_0 &  pc_1 &  pc_2  &   pc_3  &  pc_4) &  pc_5  # cout) & run;

cout0   =   pc_0 &  pc_1   &  pc_2  &   pc_3  &  pc_4  &  pc_5;

Name      Counter2;
Partno    000;
Date      09/30/90;
Revision  0.0;
Designer  Dr. Tan;
Company   Cer1;
Assembly  FPU test board;
Location  PAL5;
Device    G16V8;

/* Input */

Pin 1  = clk;
Pin 2  = cout0;
Pin 3  = run;

/* Output */

Pin 12 = cout;
Pin 13 = pc_6;
Pin 14 = pc_7;
Pin 15 = pc_8;
Pin 16 = pc_9;
Pin 17 = pc_10;
Pin 18 = pc_11;

/* Logic Equations */

pc_6.d  =  (cout0 & !pc_6          # !cout0 &   pc_6          #  cout) & run;
pc_7.d  =  (cout0 &  pc_6  & !pc_7 # !(cout0 &  pc_6) & pc_7  #  cout) & run;
pc_8.d  =  (cout0 &  pc_7  &  pc_6 &  !pc_8  #
          !(cout0 &  pc_7  &  pc_6) &   pc_8  #  cout) & run;
```

```
pc_9.d  =  (cout0 &  pc_8  &  pc_7  &   pc_6  & !pc_9  #
          !(cout0 &  pc_8  &  pc_7  &   pc_6) &  pc_9  # cout)  &  run;
pc_10.d =  (cout0 &  pc_9  &  pc_8  &   pc_7  &  pc_6  & !pc_10 #
          !(cout0 &  pc_9  &  pc_8  &   pc_7  &  pc_6) &  pc_10 #  cout) & run;
pc_11.d =  (cout0 &  pc_10 &  pc_9  &   pc_8  &  pc_7  &  pc_6  & !pc_11 #
          !(cout0 &  pc_10 &  pc_9  &   pc_8  &  pc_7  &  pc_6) &  pc_11 # cout) &
           run;
cout    =   cout0 &  pc_11 &  pc_10  &  pc_9  &  pc_8  &  pc_7 & pc_6;
```

```
NAME     Bectrl;
Partno   000;
Date     08/30/90;
Revision 0.0;
Designer Dr. Tan;
Company  Cerl;
Assembly FPU test board;
Location Pal2;
Device   G16V8;

/* Input */

Pin 1  = !ADR_13;
Pin 2  = !ADR_14;
Pin 3  = !ADR_15;
Pin 4  = !ADR_16;
Pin 5  = clk1;
Pin 6  = !BS;
Pin 7  = run;

/* Output */

Pin 12 = !Be_0;
Pin 13 = !Be_1;
Pin 14 = !Be_2;
Pin 15 = !Be_3;
Pin 16 = !Be_4;
Pin 17 = !Be_5;
Pin 18 = !Be_6;
Pin 19 = !Be_7;

/* Logic Equations */

Be_7 = BS & !ADR_16 &  ADR_15 & !ADR_14 &  ADR_13 & !run;
Be_6 = BS & !ADR_16 &  ADR_15 &  ADR_14 & !ADR_13 & !run;
Be_5 = BS & !ADR_16 &  ADR_15 &  ADR_14 &  ADR_13 & !run;
Be_4 = BS & !ADR_16 & !ADR_15 &  ADR_14 & !ADR_13 & !run;
Be_3 = BS & !ADR_16 & !ADR_15 &  ADR_14 &  ADR_13 & !run;
Be_2 = BS & !ADR_16 & !ADR_15 & !ADR_14 & !ADR_13 & !run;
Be_1 = BS & !ADR_16 & !ADR_15 & !ADR_14 &  ADR_13 & !run;
Be_0 = BS & !ADR_16 &  ADR_15 & !ADR_14 & !ADR_13 & !run;
```

```
NAME      Mbctrl;
Partno    000;
Date      08/30/90;
Revision  0.0;
Designer  Dr. Tan;
Company   Cerl;
Assembly  FPU test board;
Location  Pal3;
Device    G16V8;

/* Input */

Pin 1  = osc;
Pin 2  = !MWTC;
Pin 3  = !MRDC;
Pin 4  = !ADR_16;
Pin 5  = !ADR_17;
Pin 6  = !ADR_18;
Pin 7  = !ADR_19;

/* Output */

Pin 12 =  MBdir;
Pin 13 = !Xack;
Pin 14 =  Xack1;
Pin 15 =  Xack0;
Pin 17 =  MBdir1;
Pin 18 = !BS;
Pin 19 = !MBen;

/* Logic Equations */

Xack0.D  = !Xack  &  Xack1 & (MWTC # MRDC);
Xack1.D  = !Xack  & !Xack0 & (MWTC # MRDC) #
            Xack1 &  Xack0 & (MWTC # MRDC);

/* if BS */
Xack.OE =  BS;

Xack.D  = (Xack1 &  Xack0 & (MWTC # MRDC) #
           Xack  & !Xack0 & (MWTC # MRDC));

MBen   =  MWTC # MRDC;
MBdir  =  MWTC;
MBdir1 =  MWTC;

BS     = !ADR_19 & !ADR_18 & !ADR_17;
```

```
NAME     Runctrl;
Partno   000;
Date     08/30/90;
Revision 0.0;
Designer Dr. Tan;
Company  Cerl;
Assembly FPU test board;
Location Pal4;
Device   G16V8;

/* Input */

Pin 1  = clk_src;   /* osc */
Pin 2  = osc;
Pin 3  = !MWTC;
Pin 4  = !ADR_16;
Pin 5  = BD_0;
Pin 6  = !BS;

/* Output */

Pin 12 = !we0;
Pin 13 = clk;
Pin 14 = osc_1;
Pin 15 = run;
Pin 16 = n_run;
Pin 17 = osc_2;
Pin 18 = clk1;
Pin 19 = !we1;

/* Logic Equations */

run.d    =  (BS & MWTC & ADR_16 & BD_0) # ((!BS # !MWTC # !ADR_16) & run);
we0      =  MWTC & !run;
we1      =  (MWTC & !run) # ((!osc) & run );
osc_1    =  osc;
osc_2    =  osc_1;
n_run    = !run;
clk      =  osc_1;
clk1     =  osc_1;
```

Appendix B : Test Monitor Source Code

```
program testfpu;
uses ieee_cnv,
     {$u e:\dp\dp_comp\hex_conv} hex_conv,
                          io,dos;
const
    max_test_vector = 4092;
    { logical }
    fpu_and = $8;
    fpu_or = $9;
    fpu_xor = $a;
    fpu_notR = $b;
    fpu_notS = $c;
    fpu_passR = $14;
    { fixed }
    fpu_add = $0;
    fpu_sub = $1;
    fpu_mult = $2;
    fpu_rsub = $3;
    { float }
    fpu_fadd = $10;
    fpu_fsub = $11;
    fpu_fmult = $12;
    fpu_frsub = $13;
    { shift }
    fpu_ror = $4;
    fpu_rol = $5;
    fpu_shr = $6;
    fpu_shl = $7;
    { special }
    fpu_pack = $0d;
    fpu_float = $15;
    fpu_seed = $e;
    fpu_unp_exp = $18;
    fpu_unp_man = $19;
    fpu_rootexp = $1a;
    fpu_rootman = $1b;
    fpu_round = $f;
    fpu_int_r = $16;
    fpu_int_s = $17;
    fpu_sin_sgn = $1c;
    fpu_odd_neg = $1d;
    fpu_chg_sgn = $1e;
    fpu_tan_sgn = $1f;

    { test cases }
    logical = 0;
    iadd = 1;
    imult = 2;
    fadd = 3;
    fmult = 4;
    shift = 5;
    special = 6;

var bank,offset,i,address : word;
    start_bank, end_bank : word;
    pattern : array[0..20] of word;
    ch : char;
    command : string;
    debug : boolean;
    rsign : integer;
    ssign : integer;
    test_case : integer;
    last_op,last_s,last_r : integer;
    r,s : array[0..4095] of longint;
    op : array[0..31] of word;
    cflag, zflag : integer;
    continuous : char;
    stop_on_error : char;
    count : integer;
    test_cycle : longint;
    no_error : longint;

procedure write_error(procedure_name,message:string);
begin
  writeln('Error at procedure ',procedure_name,' !!!');
  writeln('  ',message);
  halt;
end;
```

```
procedure mwrite(bank,offset,pattern:word);
begin
  memw[segment:(bank shl 13) + offset] := pattern;
end;

function mread(bank,offset:word):word;
begin
  mread := memw[segment:(bank shl 13) + offset];
end;

procedure verify(bank,offset,data,expdata:word);
begin
  if data <> expdata then
  begin
    writeln('error at bank ',bank,' at location ',word_to_hex(offset));
    writeln('  written: ',word_to_hex(expdata));
    write  ('  read   : ',word_to_hex(data),' <CR> '); readln;
    writeln;
  end;
end;

procedure test_memory;
begin
  stop_processor;
  pattern[0] := $1234;
  pattern[1] := $0000;
  pattern[2] := $5555;
  pattern[3] := $aaaa;
  pattern[4] := $ffff;
  pattern[5] := $ff00;
  pattern[6] := $00ff;
  write('test all banks ? '); readln(ch);
  if (ch = 'y') or (ch = 'Y') then
  begin
    start_bank := 0; end_bank := 7;
  end
  else
  begin
     write('Which bank to test ? ');readln(start_bank);
    end_bank := start_bank;
  end;
  begin
    for bank := start_bank to end_bank  do
    begin
      for i := 0 to 6 do
      begin
        address := 0;
        writeln('testing pattern ',word_to_hex(pattern[i]),' on bank ',bank);
        while address <= $1ffe do
        begin
          mwrite(bank,address,pattern[i]);
          verify(bank,address,mread(bank,address),pattern[i]);
          address := address + 2;
        end;
      end;
      address := 0;
      writeln('writing address on bank ',bank);
      while address <= $1ffe do
      begin
        mwrite(bank,address,address);
        address := address + 2;
      end;
      address := 0;
      writeln('reading address on bank ',bank);
      while address <= $1ffe do
      begin
        verify(bank,address,mread(bank,address),address);
        address := address + 2;
      end;
    end;
    if (ch='y') or (ch='Y') then
    begin
      address := 0;
      writeln('writing address to all banks');
      repeat
        mwrite(0,address,address);
        address := address + 2;
      until address = $fffe;
      mwrite(0,address,address);
      address := 0;
      writeln('reading address from all banks');
      repeat
```

```
        verify(0,address,mread(0,address),address);
        address := address + 2;
      until address = $fffe;
      verify(0,address,mread(0,address),address);
    end;
    if (ch <> 'y') and (ch <> 'Y') then exit;
  end;
  writeln('memory testing completed');
end; { of test_memory }

procedure write_fpu_vector(address:word;r,s:longint;op:word);
var lsw,msw : longint;
begin
  address := address shl 1;
  lsw := $0000ffff and r;
  msw := r shr 16;
  mwrite(0,address,lsw);
  mwrite(1,address,msw);
  verify(0,address,mread(0,address),lsw);
  verify(1,address,mread(1,address),msw);
  lsw := $0000ffff and s;
  msw := s shr 16;
  mwrite(2,address,lsw);
  mwrite(3,address,msw);
  verify(2,address,mread(2,address),lsw);
  verify(3,address,mread(3,address),msw);
  op := op or $0200; { set proc_run to 1 }
  mwrite(4,address,op);
  verify(4,address,mread(4,address),op);
end; { of write_fpu_vector }

procedure check_fpu_vector(address:word;r,s:longint;op:word);
var lsw,msw : longint;
begin
  address := address shl 1;
  lsw := $0000ffff and r;
  msw := r shr 16;
  verify(0,address,mread(0,address),lsw);
  verify(1,address,mread(1,address),msw);
  lsw := $0000ffff and s;
  msw := s shr 16;
  verify(2,address,mread(2,address),lsw);
  verify(3,address,mread(3,address),msw);
  op := op or $0200; { set proc_run to 1 }
  verify(4,address,mread(4,address),op);
  mwrite(6,address+8,0);
  mwrite(7,address+8,0);
  verify(6,address,mread(6,address+8),0);
  verify(7,address,mread(7,address+8),0);
end; { of write_fpu_vector }

function sm2twosc(d:longint):longint;
begin
  if (d and $80000000) <> 0 then
    d := -(d and $00ffffff)
  else
    d := d and $00ffffff;
  sm2twosc := d;
end;

function twosc2sm(d:longint):longint;
begin
  if (d < 0) then
    d := (-d and $7fffffff) or $80000000;
  if (d and $01000000) <> 0 then cflag := 1;
  twosc2sm := d and $80ffffff;
end;

function compute_fadd(r s:longint):longint;
var
   addsubsel,expdiff,bgta,explarge : longint;
   output,mantlarge,intermediate,i : longint;
   sexp,rexp,smant,rmant,ssign,rsign : longint;
   overflow,underflow : longint;
begin
      sexp  := (s and $7f800000) shr 23;
      rexp  := (r and $7f800000) shr 23;
      smant := s and $007fffff;
      if (s<>0) then smant := smant or $00800000;
      rmant := r and $007fffff;
      if (r<>0) then rmant := rmant or $00800000;
```

```
ssign := (s and $80000000) shr 31;
rsign := (r and $80000000) shr 31;

addsubsel := (1 xor ssign xor rsign);
expdiff := rexp - sexp;
bgta := 0;
explarge := rexp;
output := rsign;

if (expdiff < 0) then
begin
   expdiff := -expdiff;
   bgta := 1;
   explarge := sexp;
   output := ssign;
end;

explarge := explarge + 1;

if (bgta = 0) then
begin
   if (expdiff>=1) then
      intermediate := smant shr (expdiff-1)
   else
      intermediate := smant shl 1;
   mantlarge    := rmant shl 1;
end
else
begin
   if (expdiff>=1) then
      intermediate := rmant shr (expdiff-1)
   else
      intermediate := rmant shl 1;
   intermediate := rmant shr (expdiff-1);
   mantlarge    := smant shl 1;
end;
if (expdiff > 23) then intermediate := 0;

if (addsubsel = 1) then
   intermediate := mantlarge + intermediate
else
   intermediate := mantlarge - intermediate;

if (intermediate < 0) then
begin
   intermediate := -intermediate;
   output := output xor 1;
end;
intermediate := intermediate shr 1;

{writeln('intermed  ',longint_to_hex(intermediate));}

i := 0;
while (  ((intermediate and $01000000)=0) and (i<25) ) do
begin
   intermediate := intermediate shl 1;
   explarge := explarge - 1;
   i := i + 1;
end;
intermediate := intermediate shr 1;

overflow  := (explarge and $100) shr 8;
underflow := (explarge and $80 ) shr 7;

if ( (underflow and overflow) <> 0 ) then
begin
   output := 0;
   zflag := 1;
   overflow := 0;
end
else
begin
   output := output shl 31;
   output := output or ( (explarge and $ff) shl 23 );
   output := output or (  intermediate and $7fffff );
end;

if (intermediate = 0) then
begin
   output := 0;
   zflag := 1;
   overflow := 0;
end;
```

```
      cflag := 0;

{     if (output <> 0) then
         writeln('fmant',longint_to_hex( (output and $7fffffff) or $80000000 ))
      else
         writeln('fmant',longint_to_hex( 0 ));
      writeln('f    ',longint_to_hex(output));
      writeln('fexp ',longint_to_hex( (output and $7f800000) shr 23 ));
 }

      compute_fadd := output;

end;{ of compute_fadd }

function compute_fmult(r,s:longint):longint;
var
   rmanthi,rmantlo,smanthi,smantlo : longint;
   output,w,x,y,z,reshi,reslo,intermediate : longint;
   sexp,rexp,smant,rmant,ssign,rsign : longint;
   overflow,underflow : longint;
begin
      sexp  := (s and $7f800000) shr 23;
      rexp  := (r and $7f800000) shr 23;
      smant := s and $007fffff;
      if (s<>0) then smant := smant or $00800000;
      rmant := r and $007fffff;
      if (r<>0) then rmant := rmant or $00800000;
      ssign := (s and $80000000) shr 31;
      rsign := (r and $80000000) shr 31;

      rmantlo := rmant and $ffff;
      rmanthi := rmant shr 16;
      smantlo := smant and $ffff;
      smanthi := smant shr 16;
      w := rmantlo * smantlo;
      x := rmanthi * smantlo;
      y := rmantlo * smanthi;
      z := rmanthi * smanthi;
      intermediate := ( (w shr 16) and $ffff ) + x + y;
      reslo := ( (intermediate shl 16) and $ffff0000 ) or (w and $ffff);
      reshi := ( (intermediate shr 16) and $ffff ) + z;

{     writeln('rmant ',longint_to_hex(rmant));
      writeln('smant ',longint_to_hex(smant));
      writeln('reshi ',longint_to_hex(reshi));
      writeln('reslo ',longint_to_hex(reslo));
      }

      intermediate := rexp + sexp;
      if ( (reshi and $8000) <> 0 ) then
      begin
         output := (reshi shl 8) or ( (reslo shr 24) and $ff );
         intermediate := intermediate - 126;
      end
      else
      begin
         output := (reshi shl 9) or ( (reslo shr 23) and $1ff );
         intermediate := intermediate - 127;
      end;

      overflow := (intermediate and $100) shr 8;
      cflag := 0;
      zflag := 0;
      underflow := ( ( (not rexp) and (not sexp) ) shr 7 ) and 1;
      if ( (overflow and underflow) <> 0 ) then
      begin
         output := 0;
         zflag := 1;
         overflow := 0;
      end
      else
      begin
         output := output and $7fffff;
         output := output or ( (ssign xor rsign) shl 31);
         output := output or ( (intermediate and $ff) shl 23 );
      end;

      if ( (reshi or reslo) = 0 ) then
      begin
         output := 0;
         zflag := 1;
         overflow := 0;
      end;
```

```
      compute_fmult := output;

end;{ of compute_fmult }

function compute_float(r:longint):longint;
var
   explarge : longint;
   output,intermediate,i : longint;
   rexp,rmant,rsign : longint;
   overflow,underflow : longint;
begin
      rmant := r and $00ffffff;
      rsign := (r and $80000000) shr 31;

      explarge := $97;
      intermediate := rmant;
      i := 0;
      while (  ((intermediate and $01000000)=0) and (i<25) ) do
      begin
         intermediate := intermediate shl 1;
         explarge := explarge - 1;
         i := i + 1;
      end;
      intermediate := intermediate shr 1;

      overflow  := (explarge and $100) shr 8;
      underflow := (explarge and $80 ) shr 7;

      if ( (underflow and overflow) <> 0 ) then
      begin
         output := 0;
         zflag := 1;
         overflow := 0;
      end
      else
      begin
         output := rsign shl 31;
         output := output or ( (explarge and $ff) shl 23 );
         output := output or (  intermediate and $7fffff );
      end;

      if (intermediate = 0) then
      begin
         output := 0;
         zflag := 1;
         overflow := 0;
      end;

      cflag := 0;
      compute_float := output;

end;{ of compute_float }

function compute_unp_exp(r:longint):longint;
var
   explarge : longint;
   output,intermediate,i : longint;
   rexp,rmant,rsign : longint;
   overflow,underflow : longint;
begin
      rexp := (r shr 23) and $ff;
      rsign := 0;
      rexp := rexp - 127;
      if (rexp < 0) then
         begin
         rexp := -rexp;
         rsign := 1;
         end;

      explarge := $97;
      intermediate := rexp;
      i := 0;
      while (  ((intermediate and $01000000)=0) and (i<25) ) do
      begin
         intermediate := intermediate shl 1;
         explarge := explarge - 1;
         i := i + 1;
      end;
      intermediate := intermediate shr 1;

      output := rsign shl 31;
      output := output or ( (explarge and $ff) shl 23 );
      output := output or (  intermediate and $7fffff );
      zflag := 0;
```

```
      if (intermediate = 0) then
      begin
         output := 0;
         zflag := 1;
         overflow := 0;
      end;

      cflag := 0;
      compute_unp_exp := output;

end;{ of compute_unp_exp }

function compute_int(r:longint;round:integer):longint;
var
   expdiff,bgta,reshi : longint;
   output,intermediate,carryin : longint;
   rexp,rmant,rsign : longint;
   overflow,underflow : longint;
begin
      rexp  := (r and $7f800000) shr 23;
      rmant := r and $007fffff;
      if (r<>0) then rmant := rmant or $00800000;
      rsign := (r and $80000000) shr 31;

      expdiff := $00000096 - rexp;
      reshi := $96;
      intermediate := rmant;
      bgta := 1;
      if (expdiff < 0) then
      begin
         reshi := rexp;
         expdiff := -expdiff;
         bgta := 0;
      end;

      if (expdiff > 0) then
         carryin := ( intermediate shr (expdiff - 1) ) and 1
      else
         carryin := 0;

      intermediate := intermediate shr (expdiff * bgta);
      if ( (round=1) and (bgta=1) ) then intermediate := intermediate + carryin;
      if ( (bgta=1) and (expdiff>24) ) then intermediate := 0;
      reshi := reshi + 1;

      overflow := 0;
      cflag := 0;
      zflag := 0;
      if (reshi <> $97) then overflow := 1;
      if (intermediate = 0) then zflag := 1;

      if ( zflag=1 ) then
         output := 0
      else
      begin
         output := rsign shl 31;
         output := output or (  intermediate and $ffffff );
      end;

      compute_int := output;

end;{ of compute_int }

procedure check_fpu_result(address:word;r,s:longint;op:word);
var f,msw,lsw,readf : longint;
    tr,ts : longint;
    rsign,ssign : longint;
    sr,ss : single;
    readcflag,readzflag : integer;
    i : integer;
    sexp,rexp,fexp,smant,rmant,fmant : longint;
    Year,Month,Day,DayOfWeek : word;
    Hour,Minute,Second,Sec100 : word;
begin
  cflag := 0;
  zflag := 0;
  address := (address+4) shl 1;
  case op of
    fpu_and:  f := r and s;
    fpu_or :  f := r or s;
    fpu_xor : f := r xor s;
    fpu_notR: f := not r;
    fpu_notS: f := not s;
```

```
fpu_passR: f := r;
fpu_add : f := twosc2sm(sm2twosc(r) + sm2twosc(s));
fpu_sub : f := twosc2sm(sm2twosc(r) - sm2twosc(s));
fpu_rsub : f := twosc2sm(sm2twosc(s) - sm2twosc(r));
fpu_mult :
begin
  ssign := s and $80000000;
  rsign := r and $80000000;
  tr := r and $ffffff;
  ts := s and $ffffff;
  f := 0;
  for i := 0 to 23 do
  begin
    if (ts and 1) = 1 then f := f + tr;
    tr := tr shl 1; ts := ts shr 1;
  end;
  if (f and $01000000) <> 0 then cflag := 1;
  if f = 0 then
  begin
    rsign := 0;
    ssign := 0;
  end;
  f := (rsign xor ssign) or (f and $00ffffff);
end;
{ float }
fpu_fadd : f := compute_fadd(r,s);
fpu_fsub : f := compute_fadd(r,(s xor $80000000));
fpu_fmult: f := compute_fmult(r,s);
fpu_frsub: f := compute_fadd((r xor $80000000),s);
fpu_ror : f := ((r shl (32 - (s and $1f))) or (r shr (s and $1f)));
fpu_rol : f := ((r shr (32 - (s and $1f))) or (r shl (s and $1f)));
fpu_shr : f := r shr (s and $1f);
fpu_shl : f := r shl (s and $1f);
fpu_pack :
begin
   f := r and $ffffff;
   if ( (r and $80000000)=0 ) then
      f := 127 + f
   else
      f := 127 - f;
   cflag := 0;
   zflag := 0;
   if ( (f < 0) or ((f and $ff)=0) )then
      begin
      f := 0;
      zflag := 1;
      end;
   f:= ( (f shl 23) and $7f800000 );
end;
fpu_float : f := compute_float(r);
fpu_seed :
begin
   f := (r and $7f800000) shr 23;
   f := f + 2;
   cflag := 0;
   zflag := 0;
   if ( (r and $7fffffff)=0 ) then
   begin
      zflag := 1;
      f := 0;
   end
   else
   begin
      f := (r and $80000000) or ( (f and $ff) shl 23 ) or $400000;
      f := f xor $7f800000;
   end;
end;
fpu_unp_exp : f:= compute_unp_exp(r);
fpu_unp_man :
begin
   f := (s and $807fffff) or $3f800000;
   zflag := 0;
   cflag := 0;
   if ( (s and $7fffffff) = 0 ) then
   begin
      zflag := 1;
      f := 0;
   end;
end;
fpu_rootexp :
begin
```

```
   if ((r and $80000000)=0) then
      if ( (r and $00800000) <> 0 ) then
         f := ( (r shr 24) and $7f ) + 64
      else
         f := ( (r shr 24) and $7f ) + 63
   else
      if ( (r and $00800000) <> 0 ) then
         f := -( (r shr 24) and $7f ) + 64
      else
         f := -( (r shr 24) and $7f ) + 63;
   if (f<0) then f := -f;
   f := (f and $ff) shl 23;
   cflag := 0;
   zflag := 0;
   if ((r and $7fffffff)=0) then
   begin
      zflag := 1;
      f := 0;
   end;
end;
fpu_rootman :
begin
   if ( (s and $00800000) = 0) then
      f := (s and $007fffff) or $40000000
   else
      f := (s and $007fffff) or $3f800000;
   cflag := 0;
   zflag := 0;
   if ((s and $7fffffff)=0) then
   begin
      zflag := 1;
      f := 0;
   end;
end;
fpu_round : f:= compute_int(r,1);
fpu_int_r : f:= compute_int(r,0);
fpu_int_s : f:= compute_int(s,0);
fpu_sin_sgn :
begin
   f := (s xor (r shl 31)) and $80000000;
   f := f or (s and $7fffffff);
   zflag := 0;
   cflag := 0;
   if ((f and $7fffffff)=0) then
   begin
      zflag := 1;
      f := 0;
   end;
end;
fpu_odd_neg :
begin
   f := (s shl 31) or (s and $ffffff);
   zflag := 0;
   cflag := 0;
   if ((s and $ffffff)=0) then
   begin
      zflag := 1;
      f := 0;
   end;
end;
fpu_chg_sgn :
begin
   f := (r and $80000000) or (s and $7fffffff);
   zflag := 0;
   cflag := 0;
   if ((s and $7fffffff)=0) then
   begin
      zflag := 1;
      f := 0;
   end;
end;
fpu_tan_sgn :
begin
   f := (r shr 31) xor (s shr 31) xor r;
   f := f shl 31;
   f := f or (s and $7fffffff);
   cflag := 0;
   zflag := 0;
   if ((f and $7fffffff)=0) then
   begin
      zflag := 1;
```

```
          f := 0;
        end;
      end;
    end; { of opcode case }
    if (f=0) then zflag := 1 else zflag := 0;
    lsw := mread(6,address);
    msw := mread(7,address);
    readf := (msw shl 16) or lsw;
    lsw := mread(5,address);
    readcflag := (lsw and 1);
    readzflag := (lsw and 2) shr 1;
    if (readf <> f) or (readcflag <> cflag) or (readzflag <> zflag) or (debug) then
    begin
      if (readf <> f) or (readcflag <> cflag) then no_error := no_error + 1;
      write(word_to_hex(address shr 2),':',longint_to_hex(r),' ');
      case op of
        fpu_and : write('and  ');
        fpu_or  : write('or   ');
        fpu_xor : write('xor  ');
        fpu_add : write('fix+ ');
        fpu_sub : write('fix- ');
        fpu_mult: write('fix* ');
        fpu_rsub: write('fixr-');
        fpu_passr: write('passR');
        fpu_pack    : write('pack exp R');
        fpu_float   : write('float R');
        fpu_seed    : write('inv seed R');
        fpu_unp_exp : write('unpack exp R');
        fpu_unp_man : write('unpack mant S');
        fpu_rootexp : write('root exp R');
        fpu_rootman : write('root mant S');
        fpu_round   : write('round R');
        fpu_int_r   : write('truncate R');
        fpu_int_s   : write('truncate S');
        fpu_sin_sgn : write('sine sign');
        fpu_odd_neg : write('odd negative S');
        fpu_chg_sgn : write('change sign');
        fpu_tan_sgn : write('tangent sign');
        fpu_fadd : write('float+ ');
        fpu_fsub : write('float- ');
        fpu_fmult: write('float* ');
        fpu_frsub: write('floatr-');
      end;
      write(' ',longint_to_hex(s),' -> ');
      write(longint_to_hex(f),'[',zflag,',',cflag,'] (pc) ');
      write(longint_to_hex(readf),'[',readzflag,',',readcflag,'] (fpu) ');
      if (readf <> f) or (readcflag <> cflag) then
      begin
        GetTime(Hour,Minute,Second,Sec100);
        GetDate(Year,Month,Day,DayOfWeek);
        write('Date: ',Month,'/',Day,'/',Year,' at ',Hour,':',Minute,':',Second,' ');
        if (command = 'tall') then writeln('Error at cycle ',count,'.')
                              else writeln('Error at cycle ',test_cycle,'.');
      end;
      if (stop_on_error = 'y') then readln else writeln;
    end; { of readf <> f}
  end; { of check_fpu_result }

  procedure generate_vectors(phase : integer);
  var i,j,k : integer;
  begin
    address := 2;
    writeln('generating test vectors phase ',phase);
    case phase of
    0:
      for k := 0 to last_op do
        for j := 0 to last_s do
          for i := 0 to last_r do
          begin
            if address >= max_test_vector then write_error('','too many test vectors');
            write_fpu_vector(address,r[i],s[j],op[k]);
            address := address + 1;
          end;
    1:
      for k := 0 to last_op do
        for i := 0 to last_r do
          for j := 0 to last_s do
          begin
            if address >= max_test_vector then write_error('','too many test vectors');
            write_fpu_vector(address,r[i],s[j],op[k]);
            address := address + 1;
```

```
        end;
  2:
    for j := 0 to last_s do
      for i := 0 to last_r do
        for k := 0 to last_op do
        begin
          if address >= max_test_vector then write_error('','too many test vectors');
          write_fpu_vector(address,r[i],s[j],op[k]);
          address := address + 1;
        end;
  end;
end; { of generate_vectors }

procedure wait;
var i : integer;
begin
  for i := 0 to 4095 do;
end; { of wait }

Procedure check_vectors(phase : integer);
var i,j,k : integer;
begin
  writeln('checking test vectors phase ',phase);
  stop_processor;
  address := 2;
  case phase of
  0:
    for k := 0 to last_op do
      for j := 0 to last_s do
        for i := 0 to last_r do
        begin
          check_fpu_vector(address,r[i],s[j],op[k]);
          address := address + 1;
      end;
  1:
    for k := 0 to last_op do
      for i := 0 to last_r do
        for j := 0 to last_s do
        begin
          check_fpu_vector(address,r[i],s[j],op[k]);
          address := address + 1;
      end
  2:
    for j := 0 to last_s do
      for i := 0 to last_r do
        for k := 0 to last_op do
        begin
          check_fpu_vector(address,r[i],s[j],op[k]);
          address := address + 1;
      end;
  end;
end; { of check_vectors }

procedure check_results(phase : integer);
var i,j,k : integer;
begin
  writeln('checking test result');
  start_processor;
  wait;
  address := 2;
  stop_processor;
  case phase of
  0 :
    for k := 0 to last_op do
      for j := 0 to last_s do
        for i := 0 to last_r do
        begin
          if not odd(k) then
            check_fpu_result(address,r[i],s[j],op[k]);
          address := address + 1;
        end;
  1 :
    for k := 0 to last_op do
      for i := 0 to last_r do
        for j := 0 to last_s do
        begin
          if not odd(k) then
            check_fpu_result(address,r[i],s[j],op[k]);
          address := address + 1;
        end;
  2 :
    for j := 0 to last_s do
```

```
      for i := 0 to last_r do
        for k := 0 to last_op do
        begin
          if not odd(k) then
            check_fpu_result(address,r[i],s[j],op[k]);
          address := address + 1;
        end;
  end;
end; { of check_result }

procedure test_random;
var i,j : integer;
begin
  writeln('-- random pattern test');
  stop_processor;
  writeln('generate vectors');
  j := 0;
  for i := 2 to 4090 do
  begin
    r[i] := random($ffff);
    r[i] := (r[i] shl 16) or random($ffff);
    s[i] := r[i];
    write_fpu_vector(i,r[i],s[i],op[j]);
    if j = last_op then j := 0 else j := j+1;
  end;
  writeln('checking vectors');
  j := 0;
  for i := 2 to 4090 do
  begin
    check_fpu_vector(i,r[i],s[i],op[j]);
    if j = last_op then j := 0 else j := j+1;
  end;
  start_processor;
  wait;
  writeln('checking fpu results');
  stop_processor;
  j := 0;
  for i := 2 to 4090 do
  begin
    if not odd(j) then
      check_fpu_result(i,r[i],s[i],op[j]);
    if j = last_op then j := 0 else j := j+1;
  end;
end; { of test_random }

procedure test_logical;
var
   phase : integer;
begin
  writeln('--  xor/and/or/passR tests  --');
  test_case := logical;
  op[0] := fpu_passR; op[1] := not op[0];
  op[2] := fpu_and;   op[3] := not op[2];
  op[4] := fpu_or;    op[5] := not op[4];
  op[6] := fpu_xor;   op[7] := not op[6];
  op[8] := fpu_xor;   op[9] := not op[6];
  last_op := 9;
  test_cycle := 0;
  repeat
    writeln('----  logical test cycle ',test_cycle,'  ---');
    test_cycle := test_cycle + 1;
    writeln('-- fixed pattern test');
    r[0] := $00000000; r[1] := $ffffffff; r[2] := $55555555; r[3] := $aaaaaaaa;
    r[4] := $ffffffff; r[5] := $00000000; r[6] := $12345678; r[7] := $9abcdef0;
    last_r := 7;
    s[0] := $00000000; s[1] := $ffffffff; s[2] := $55555555; s[3] := $aaaaaaaa;
    s[4] := $ffffffff; s[5] := $00000000; s[6] := $12345678; s[7] := $9abcdef0;
    last_s := 7;
    for phase := 0 to 2 do
    begin
      generate_vectors(phase);
      check_vectors(phase);
      check_results(phase);
    end;
    test_random;
  until (continuous <> 'y') and (continuous <> 'Y');
end; { of test_logical }

procedure test_iadd;
var
    phase : integer;
begin
```

```
  test_case := iadd;
  test_cycle := 1;
  writeln('begin integer add/sub/rsub tests');

  op[0] := fpu_add;  op[1] := not op[0];
  op[2] := fpu_sub;  op[3] := not op[2];
  op[4] := fpu_rsub; op[5] := not op[4];
  last_op := 5;

  test_cycle := 1;
  repeat
    writeln('----  integer add test cycle ',test_cycle,'  ---');
    test_cycle := test_cycle + 1;
    writeln('-- fixed pattern test');
    r[0] := $00000000; r[1] := $00ffffff; r[2] := $00555555; r[3] := $00aaaaaa;
    r[4] := $00000001; r[5] := $00123456; r[6] := $00abcdef; r[7] := $80ffffff;
    r[8] := $80555555; r[9] := $80aaaaaa; r[10]:= $80000001; r[11]:= $80123456;
    r[12]:= $80abcdef;
    last_r := 12;
    s[0] := $00000000; s[1] := $00ffffff; s[2] := $00555555; s[3] := $00aaaaaa;
    s[4] := $00000001; s[5] := $00123456; s[6] := $00abcdef; s[7] := $80ffffff;
    s[8] := $80555555; s[9] := $80aaaaaa; s[10]:= $80000001; s[11]:= $80123456;
    s[12]:= $80abcdef;
    last_s := 12;
    for phase := 0 to 2 do
    begin
      generate_vectors(phase);
      check_vectors(phase);
      check_results(phase);
    end;
    test_random;
  until (continuous <> 'y') and (continuous <> 'Y');
end; { of test_fix }

procedure test_imult;
var
    phase : integer;
begin
  test_case := imult;
  test_cycle := 1;
  writeln('begin integer mult tests');

  op[0] := fpu_mult; op[1] := not op[0];
  last_op := 1;

  test_cycle := 1;
  repeat
    writeln('----  integer mult test cycle ',test_cycle,'  ---');
    test_cycle := test_cycle + 1;
    writeln('-- fixed pattern test');
    r[0] := $00000000;
    r[1] := $00ffffff; r[2] := $00aaaaaa; r[3] := $00555555;
    r[4] := $00000fff; r[5] := $00000aaa; r[6] := $00000555; r[7] := $00ffffff;
    r[8] := $8000000;
    r[9] := $80ffffff; r[10] := $80aaaaaa; r[11] := $80555555;
    r[12] := $80000fff; r[13] := $80000aaa; r[14] := $80000555; r[15] := $80ffffff;
    last_r := 15;
    s[0] := $00000000;
    s[1] := $00ffffff; s[2] := $00aaaaaa; s[3] := $00555555;
    s[4] := $00000fff; s[5] := $00000aaa; s[6] := $00000555; s[7] := $00ffffff;
    s[8] := $8000000;
    s[9] := $80ffffff; s[10] := $80aaaaaa; s[11] := $80555555;
    s[12] := $80000fff; s[13] := $80000aaa; s[14] := $80000555; s[15] := $80ffffff;
    last_s := 15;
    for phase := 0 to 2 do
    begin
      generate_vectors(phase);
      check_vectors(phase);
      check_results(phase);
    end;
    test_random;
  until (continuous <> 'y') and (continuous <> 'Y');
end; { of test_imult }

procedure test_fmult;
var
    phase : integer;
begin
  test_case := fmult;
  test_cycle := 1;
  writeln('begin float mult tests');
```

```
  op[0] := fpu_fmult; op[1] := not op[0];
  last_op := 1;

  test_cycle := 1;
  repeat
    writeln('----  floating mult test cycle ',test_cycle,'  ---');
    test_cycle := test_cycle + 1;
    writeln('-- fixed pattern test');
    r[0] := $00000000;
    r[1] := $3fffffff; r[2] := $3faaaaaa; r[3] := $3f555555;
    r[4] := $00000fff; r[5] := $00000aaa; r[6] := $7f000555; r[7] := $7fffffff;
    r[8] := $08000000;
    r[9] := $bfffffff; r[10] := $bfaaaaaa; r[11] := $bf555555;
    r[12] := $80000fff; r[13] := $80000aaa; r[14] := $ff000555; r[15] := $ffffffff;
    last_r := 15;
    s[0] := $00000000;
    s[1] := $3fffffff; s[2] := $3faaaaaa; s[3] := $3f555555;
    s[4] := $00000fff; s[5] := $00000aaa; s[6] := $7f000555; s[7] := $7fffffff;
    s[8] := $8000000;
    s[9] := $bfffffff; s[10] := $bfaaaaaa; s[11] := $bf555555;
    s[12] := $80000fff; s[13] := $80000aaa; s[14] := $ff000555; s[15] := $ffffffff;
    last_s := 15;
    for phase := 0 to 2 do
    begin
      generate_vectors(phase);
      check_vectors(phase);
      check_results(phase);
    end;
    test_random;
  until (continuous <> 'y') and (continuous <> 'Y');
end; { of test_fmult }

procedure test_fadd;
var
    phase : integer;
begin
  test_case := fadd;
  test_cycle := 1;
  writeln('begin floating point add tests');

  op[0] := fpu_fadd; op[1] := not op[0];
  op[2] := fpu_fsub; op[3] := not op[2];
  op[4] := fpu_frsub; op[5] := not op[4];
  last_op := 5;

  test_cycle := 1;
  repeat
    writeln('----  floating add test cycle ',test_cycle,'  ---');
    test_cycle := test_cycle + 1;
    writeln('-- fixed pattern test');
    r[0] := $3f000000;
    r[1] := $3fffffff; r[2] := $3faaaaaa; r[3] := $3f555555; r[4] := $3f000001;
    r[5] := $7f000000; r[6] := $2a800000; r[7] := $55000000; r[8] := $00800000;
    r[9] := $bf000000;
    r[10] := $bfffffff; r[11] := $bfaaaaaa; r[12] := $bf555555; r[13] := $bf000001;
    r[14] := $ff000000; r[15] := $aa800000; r[16] := $d5000000; r[17] := $80800000;
    last_r := 17;
    s[0] := $3f000000;
    s[1] := $3fffffff; s[2] := $3faaaaaa; s[3] := $3f555555; s[4] := $3f000001;
    s[5] := $7f000000; s[6] := $2a800000; s[7] := $55000000; s[8] := $00800000;
    s[9] := $bf000000;
    s[10] := $bfffffff; s[11] := $bfaaaaaa; s[12] := $bf555555; s[13] := $bf000001;
    s[14] := $ff000000; s[15] := $aa800000; s[16] := $d5000000; s[17] := $80800000;
    last_s := 17;
    for phase := 0 to 2 do
    begin
      generate_vectors(phase);
      check_vectors(phase);
      check_results(phase);
    end;
    test_random;
  until (continuous <> 'y') and (continuous <> 'Y');
end; { of test_fadd }

procedure test_shift;
var
    phase : integer;
    var i : integer;
begin
  test_case := shift;
  test_cycle := 1;
  writeln('begin ror/rol/shr/shl tests');
```

```
  op[0] := fpu_ror; op[1] := not op[0];
  op[2] := fpu_rol; op[3] := not op[2];
  op[4] := fpu_shr; op[5] := not op[4];
  op[6] := fpu_shl; op[7] := not op[6];
  last_op := 7;

  test_cycle := 1;
  repeat
    writeln('----  ror/rol/shr/shl test cycle ',test_cycle,'  ---');
    test_cycle := test_cycle + 1;
    writeln('-- fixed pattern test');
    r[0] := $00000000;
    r[1] := $ffffffff; r[2] := $aaaaaaaa; r[3] := $55555555; r[4] := $01234567;
    r[5] := $89abcdef;
    last_r := 5;
    for I := 0 to 31 do s[i] := i;
    last_s := 31;
    for phase := 0 to 2 do
    begin
      generate_vectors(phase);
      check_vectors(phase);
      check_results(phase);
    end;
    test_random;
  until (continuous <> 'y') and (continuous <> 'Y');
end; { of test_shift }

procedure test_special0;
var
    phase : integer;
begin
    test_case := special;
    test_cycle := 1;
    writeln('Testing Pack Exp and Float');
    op[0] := fpu_pack; op[1] := not op[0];
    op[2] := fpu_float; op[3] := not op[0];
    last_op := 3;
    r[0] := 1; r[1] := 4; r[2] := 16; r[3] := 64; r[4] := 256;
    r[5] := $ffffff;
    r[6] := $80000001; r[7] := $80000004; r[8] := $80000010;
    r[9] := $80000040; r[10] := $80000100; r[11] := $80ffffff;
    last_r := 11;
    s[0] := 0; s[1] := not s[0];
    last_s := 1;
  repeat
    writeln('----  special test 0 cycle ',test_cycle,'  ---');
    test_cycle := test_cycle + 1;
    for phase := 0 to 2 do
    begin
      generate_vectors(phase);
      check_vectors(phase);
      check_results(phase);
    end;
    test_random;
  until (continuous <> 'y') and (continuous <> 'Y');
end;  { of test_special0 }

procedure test_special1;
var
    phase : integer;
begin
    test_case := special;
    test_cycle := 1;
    writeln('Testing seed, unp_exp and unp_man, rootexp, rootman');
    op[0] := fpu_seed; op[1] := not op[0];
    op[2] := fpu_unp_exp; op[3] := not op[0];
    op[4] := fpu_unp_man; op[5] := not op[0];
    op[6] := fpu_rootexp; op[7] := not op[0];
    op[8] := fpu_rootman; op[9] := not op[0];
    last_op := 9;
    r[0] := $00000000;
    r[1] := $3fffffff;
    r[2] := $00000fff;
    r[3] := $7f000555;
    r[4] := $08000000;
    r[5] := $bfffffff;
    r[6] := $80000fff;
    r[7] := $ff000555;
    last_r := 7;
    s[0] := $00000000;
    s[1] := $3fffffff;
    s[2] := $00000fff;
```

```
    s[3] := $7f000555;
    s[4] := $08000000;
    s[5] := $bfffffff;
    s[6] := $80000fff;
    s[7] := $ff000555;
    last_s := 7;
  repeat
    writeln('----  special test 1 cycle ',test_cycle,'  ---');
    test_cycle := test_cycle + 1;
    for phase := 0 to 2 do
    begin
      generate_vectors(phase);
      check_vectors(phase);
      check_results(phase);
    end;
    test_random;
  until (continuous <> 'y') and (continuous <> 'Y');
end;  { of test_special1 }

procedure test_special2;
var
    phase : integer;
begin
    test_case := special;
    test_cycle := 1;
    writeln('Testing Round and Trunc/Int');
    op[0] := fpu_round; op[1] := not op[0];
    op[2] := fpu_int_r;  op[3] := not op[0];
    op[4] := fpu_int_s;  op[5] := not op[0];
    last_op := 5;
    r[0] := $00000000;
    r[1] := $3fffffff;
    r[2] := $46000fff;
    r[3] := $3f000555;
    r[4] := $4b000000;
    r[5] := $bfffffff;
    r[6] := $c6000fff;
    r[7] := $bf000555;
    last_r := 7;
    s[0] := $00000000;
    s[1] := $3fffffff;
    s[2] := $46000fff;
    s[3] := $3f000555;
    s[4] := $4b000000;
    s[5] := $bfffffff;
    s[6] := $c6000fff;
    s[7] := $bf000555;
    last_s := 7;
  repeat
    writeln('----  special test 2 cycle ',test_cycle,'  ---');
    test_cycle := test_cycle + 1;
    for phase := 0 to 2 do
    begin
      generate_vectors(phase);
      check_vectors(phase);
      check_results(phase);
    end;
    test_random;
  until (continuous <> 'y') and (continuous <> 'Y');
end;  { of test_special2 }

procedure test_special3;
var
    phase : integer;
begin
    test_case := special;
    test_cycle := 1;
    writeln('Testing Special Sign Manipulation');
    op[0] := fpu_sin_sgn; op[1] := not op[0];
    op[2] := fpu_odd_neg; op[3] := not op[0];
    op[4] := fpu_chg_sgn; op[5] := not op[0];
    op[6] := fpu_tan_sgn; op[7] := not op[0];
    last_op := 7;
    r[0] := $00000000;
    r[1] := $00000001;
    r[2] := $80000010;
    r[3] := $80000001;
    r[4] := $3f800000;
    r[5] := $3f800001;
    r[6] := $bf800010;
    r[7] := $bf800001;
    last_r := 7;
```

```
    s[0] := $00000000;
    s[1] := $00000001;
    s[2] := $80000010;
    s[3] := $80000001;
    s[4] := $3f800000;
    s[5] := $3f800001;
    s[6] := $bf800010;
    s[7] := $bf800001;
    last_s := 7;
  repeat
    writeln('----  special test 3 cycle ',test_cycle,'  ---');
    test_cycle := test_cycle + 1;
    for phase := 0 to 2 do
    begin
      generate_vectors(phase);
      check_vectors(phase);
      check_results(phase);
    end;
    test_random;
  until (continuous <> 'y') and (continuous <> 'Y');
end;  { of test_special3 }

procedure substitute_word;
var start_address, data : word;
    xd : string;
begin
  write('start address ? '); readln(start_address);
  if odd(start_address) then start_address := start_address - 1;
  repeat
    write(word_to_hex(start_address),':',word_to_hex(mread(bank,start_address)));
    write(' -> '); readln(xd);
    if xd <> 'quit' then
    begin
      data := hex_to_word(xd);
      mwrite(bank,start_address,data);
      verify(bank,address,mread(bank,start_address),data);
      start_address := start_address + 2;
    end;
  until xd = 'quit';
end; { of substitute_word }

procedure display_word;
var i,start_address, no_word : word;
begin
  write('start address ? ');readln(start_address);
  if odd(start_address) then start_address := start_address - 1;
  write('no of words ? ');readln(no_word);
  writeln('memory bank ',bank);
  i := start_address;
  while i <= (start_address + no_word*2) do
  begin
    writeln(word_to_hex(i),':',word_to_hex(mread(bank,i)));
    i := i + 2;
  end;
end; { of display_word }

procedure select_bank;
begin
  write('bank : ',bank,' -> ');readln(bank);
end; { of select_bank }

procedure t;
begin
  stop_processor;
  for i := 0 to 32 do
  begin
    mwrite(4,i shl 1,i);
    verify(4,i shl 1,mread(4,i shl 1),i);
  end;
  start_processor;
end;

procedure toggle_debug;
begin
  if debug = true then
  begin
    debug := false;
    writeln('debugging off');
 end
 else
 begin
   debug := true;
```

```
   writeln('debugging on');
 end;
end;

Procedure test_all;
var local_cont : char;
begin
  local_cont := continuous;
  continuous := 'n';
  count := 1;
  repeat
    writeln('*** Test all functions *** Cycle ',count,' (errors = ',no_error,') ***');
    test_logical;
    writeln('*******************************************************');
    test_iadd;
    writeln('*******************************************************');
    test_fadd;
    writeln('*******************************************************');
    test_imult;
    writeln('*******************************************************');
    test_fmult;
    writeln('*******************************************************');
    test_shift;
    writeln('*******************************************************');
    test_special0;
    writeln('*******************************************************');
    test_special1;
    writeln('*******************************************************');
    test_special2;
    writeln('*******************************************************');
    test_special3;
    writeln('*******************************************************');
    count := count + 1;
  until (local_cont <> 'y') and (local_cont <> 'Y');
  continuous := local_cont;
end;

begin
  debug := false;
  continuous := 'n';
  stop_on_error := 'y';
  no_error := 0;
  writeln('FPU Test Monitor');
  stop_processor;
  while true do
  begin
    write('>'); readln(command);
    if command = 'help' then
    begin
      writeln('tmem   : test memory');
      writeln('tlog   : test xor/and/or/passR');
      writeln('tiadd  : test integer add/sub/rsub');
      writeln('timult : test integer mult');
      writeln('tfadd  : test floating point add');
      writeln('tfmult : test floating point mult');
      writeln('tshift : test ROR/ROL/SHR/SHL');
      writeln('tspec0 : test pack exp & float ');
      writeln('tspec1 : test seed, unp_exp, unp_man, root_exp, & root_man');
      writeln('tspec2 : test round & trunc');
      writeln('tspec3 : test sign manipulation');
      writeln('tall   : test all of the above');
      writeln('dsoe   : do not stop on error');
      writeln('soe    : stop on error');
      writeln('start  : start testing');
      writeln('stop   : stop testing');
      writeln('sb     : select memory bank');
      writeln('dw     : display memory word');
      writeln('sw     : substitute memory word');
      writeln('cont   : set testing mode to continuous');
      writeln('single : set testing mode to single');
      writeln('debug  : toggle debug setting');
      writeln('quit   : quit FPU Test Monitor');
    end
    else
    if command = 'tmem' then test_memory
    else
    if command = 'tlog' then test_logical
    else
    if command = 'tiadd' then test_iadd
    else
    if command = 'tfadd' then test_fadd
    else
```

```
    if command = 'timult' then test_imult
    else
    if command = 'tfmult' then test_fmult
    else
    if command = 'tshift' then test_shift
    else
    if command = 'tspec0' then test_special0
    else
    if command = 'tspec1' then test_special1
    else
    if command = 'tspec2' then test_special2
    else
    if command = 'tspec3' then test_special3
    else
    if command = 'tall' then test_all
    else
    if command = 'dsoe' then stop_on_error := 'n'
    else
    if command = 'soe' then stop_on_error := 'y'
    else
    if command = 'start' then start_processor
    else
    if command = 'stop' then stop_processor
    else
    if command = 'sw' then substitute_word
    else
    if command = 'dw' then display_word
    else
    if command = 'sb' then select_bank
    else
    if command = 't' then t
    else
    if command = 'debug' then toggle_debug
    else
    if command = 'quit' then exit
    else
    if command = 'cont' then
    begin
      continuous := 'y';
      writeln('Next test specified will be repeated indefinitely');
    end
    else
    if command = 'single' then
    begin
      continuous := 'n';
      writeln('Next test specified will not be repeated');
    end
    else
    if command = 'exit' then exit
  end;
end.
```

Appendix C : GAL Listing

```
Name      rom_ctrl;
Partno    000;
Date      03/8/91;
Revision  0.0;
Designer  Dr. Tan;
Company   Cerl;
Assembly  FPU test board;
Location  Pal6;
Device    G16V8;

/* Inputs */

Pin 1  = clk;
Pin 2  = !reset;
Pin 3  = read3;
Pin 4  = freeze;
Pin 5  = guard;
Pin 6  = clk_adv;

/* Output */

Pin 12 = rom_a0;
Pin 13 = rom_a1;
Pin 14 = rom_a2;
Pin 15 = !rom_cs;
Pin 16 = dav3;
Pin 17 = fpu_frz;
Pin 18 = clear;

/* Logic Equations */

rom_a0.d  = !rom_a0 & dav3 & !freeze & !guard & !reset
          # rom_a0 & (guard # freeze # !dav3) & !reset;
rom_a1.d  = ((rom_a0 & !rom_a1 # !rom_a0 & rom_a1)
          & dav3 & !freeze & !guard & !reset)
          #  rom_a1 & (guard # freeze # !dav3) & !reset;
rom_a2.d  = (( rom_a0 & rom_a1 & !rom_a2  # !(rom_a0  & rom_a1) & rom_a2)
          & dav3 & !freeze & !guard & !reset)
          # (rom_a2 & (freeze # guard # !dav3) & !reset);
dav3.d    = read3 & !dav3 # dav3 & (freeze # guard) & read3;
rom_cs    = read3;
clear     = reset;
fpu_frz   = 'b'0;
```

```
NAME      inst_ctl;
Partno    000;
Date      02/20/91;
Revision  0.0;
Designer  Dr. Wei Siong Tan;
Company   Cerl, Georgia Tech;
Assembly  GT-EP Evaluation Board;
Location  GAL2;
Device    G16V8;

/* Input */

Pin 2  = clk_adv;
Pin 3  = !inst_wr;
Pin 4  = !cs_pha;
Pin 5  = !cs_phb;
Pin 6  = inst_en;

/* Output */

Pin 12 = !cs_phb0;
Pin 13 = !cs_pha0;
Pin 14 = !cs_pha1;
Pin 15 = !cs_pha2;
Pin 16 = !inst_oe0;
Pin 17 = !inst_oe1;
Pin 18 = !inst_we0;
Pin 19 = !inst_we1;

/* Logic Equations */

cs_phb0 = cs_pha;
cs_pha0 = cs_phb; /* iag instruction is actually accessed on phase B */
cs_pha1 = cs_phb; /* iag instruction is actually accessed on phase B */
cs_pha2 = cs_phb;
inst_oe0 = !inst_en;
inst_oe1 = !inst_en;
inst_we0 = inst_wr & clk_adv;
inst_we1 = inst_wr & clk_adv;
```

```
NAME      dm_ctrl;
Partno    000;
Date      02/20/91;
Revision  0.0;
Designer  Dr. Wei Siong Tan;
Company   Carl, Georgia Tech;
Assembly  GT-EP Evaluation Board;
Location  GAL2;
Device    G16V8;

/* Input */
/* pin 1 = osc */
Pin 2  = write2;
Pin 3  = !write1;
Pin 4  = booting;
Pin 5  = read3;
Pin 6  = clk;
Pin 7  = ods_sel;
Pin 8  = ids_sel;
Pin 9  = write3;

/* Output */

Pin 12 = !dat_we0;
Pin 13 = !dat_we1;
Pin 14 = !dat_oe0;
Pin 15 = !rf_en;
Pin 16 = clk_adv;
Pin 17 = !fpu_oe;
Pin 18 = fpu_frz;
Pin 19 = rf_off24;

$define false   (clk & !clk)

/* Logic Equations */

dat_we0      = write1 & clk_adv & !write2 & !write3;
dat_we1      = write1 & clk_adv & !write2 & !write3;
dat_oe0      = !clk_adv & !read3;
rf_en        = ods_sel # read3;
clk_adv.d    = !clk;
fpu_oe       = ods_sel & !read3;
fpu_frz      = 'b'0;
rf_off24.oe  = !clk;
```

```
NAME      ap_ctrl1;
Partno    000;
Date      03/21/91;
Revision  0.0;
Designer  Dr. Wei Siong Tan;
Company   Cerl, Georgia Tech;
Assembly  GT-EP Evaluation Board;
Location  GAL3;
Device    G16V8;

/* Input */

/* Pin 1 = AP_clk; */
Pin 2  = !reset;
Pin 3  = write2;
Pin 4  = read2;
Pin 5  = !cp_ack;
Pin 6  = !ntc_ack;
Pin 7  = !bus_error;
Pin 8  = !ep_mstr;
Pin 9  = a23;
Pin 19 = rf_adr24;

/* Output */

Pin 12 = !ack;
Pin 13 = !EP_Brq;
Pin 14 = AP_dir;
Pin 15 = rfi2;
Pin 16 = !wr;
Pin 17 = ap_cab;
Pin 18 = apstate;

/* Logic Equations */
$DEFINE idle      'b'11
$DEFINE writing   'b'01
$DEFINE reading0  'b'00
$DEFINE reading1  'b'10

$DEFINE idle_s     ( apstate &  AP_dir)
$DEFINE writing_s  (!apstate &  AP_dir)
$DEFINE reading0_s (!apstate & !AP_dir)
$DEFINE reading1_s ( apstate & !AP_dir)

$DEFINE tiw   (write2 & rf_adr24)
$DEFINE twi   (ack & !(write2 & rf_adr24) & !read2 # reset)
$DEFINE twr0  (ack & read2)
$DEFINE tir0  (read2)
$DEFINE tr0r1 (ack # reset)
$DEFINE tr1i  (!read2 & !(write2 & rf_adr24))
$DEFINE tr1w  (write2 & rf_adr24)

   ack = ((ntc_ack & !a23 # cp_ack & a23 # bus_error) & ep_mstr # reset);
   rfi2.d = idle_s # writing_s & ack # reading1_s # rfi2 & write2 & rf_adr24;
   wr.oe = ep_mstr;
   wr.d = idle_s & write2 & rf_adr24
        # writing_s & (!ack # write2 & rf_adr24)
        # reading1_s & (write2 & rf_adr24);
```

```
   ap_cab.d  = idle_s & write2 & rf_adr24
             # writing_s & ack & write2 & rf_adr24
             # reading1_s & write2 & rf_adr24;
   EP_Brq.d = idle_s & (read2 # write2 & rf_adr24) #
              writing_s # reading0_s # reading1_s;

field state_machine = [apstate,AP_dir];

SEQUENCE state_machine {
PRESENT idle     IF      tiw   NEXT writing;
                 IF      tir0  NEXT reading0;
                 DEFAULT       NEXT idle;
PRESENT writing  IF      twi   NEXT idle;
                 IF      twr0  NEXT reading0;
                 DEFAULT       NEXT writing;
PRESENT reading0 IF      tr0r1 NEXT reading1;
                 DEFAULT       NEXT reading0;
PRESENT reading1 IF      tr1i  NEXT idle;
                 IF      tr1w  NEXT writing;
                 DEFAULT       NEXT reading1;
}
```

```
NAME      ap_ctrl2;
Partno    000;
Date      03/21/91;
Revision  0.0;
Designer  Dr. Wei Siong Tan;
Company   Cerl, Georgia Tech;
Assembly  GT-EP Evaluation Board;
Location  GAL3;
Device    G16V8;

/* Input */

/* Pin 1 = AP_clk; */
Pin 2  = !reset;
Pin 3  = write2;
Pin 4  = read2;
Pin 5  = !ack;
Pin 6  = !EP_Brq;
Pin 7  = AP_dir;
Pin 8  = !ep_mstr;
Pin 9  = adv_clk;
Pin 18 = apstate;
Pin 19 = rf_adr24;

/* Output */

Pin 12 = dav2;
Pin 13 = !rd;
Pin 14 = ap_adck;
Pin 15 = ap_cba;
Pin 16 = !ap_dat_en;

/* Logic Equations */
$DEFINE idle_s     ( apstate &  AP_dir)
$DEFINE writing_s  (!apstate &  AP_dir)
$DEFINE reading0_s (!apstate & !AP_dir)
$DEFINE reading1_s ( apstate & !AP_dir)

   dav2.d = (ack & reading0_s) # dav2 & read2 & reading1_s;
   rd.oe = ep_mstr;
   rd.d = idle_s & read2 # reading0_s & !ack # writing_s & read2 & ack;
   ap_adck.d = idle_s & (read2 # write2 & rf_adr24)
             # writing_s & ack
             # reading1_s & write2 & rf_adr24;
   ap_cba.d  = reading0_s & ack;
   ap_dat_en = AP_dir & ep_mstr # (!AP_dir & read2);
```

```
NAME      ap_ctrl3;
Partno    000;
Date      03/21/91;
Revision  0.0;
Designer  Dr. Wei Siong Tan;
Company   Carl, Georgia Tech;
Assembly  GT-EP Evaluation Board;
Location  GAL4;
Device    G16V8;

/* Input */

/* Pin 1 = AP_clk; */
Pin 2  = rf_adr0;
Pin 3  = rf_adr1;
Pin 4  = write3;
Pin 5  = kernel_mode;
Pin 6  = !reset_out;
Pin 7  = !ep_mstr;
Pin 8  = freeze;
Pin 9  = rf_adr24;

/* Output */

Pin 12 = !intr1;
Pin 13 = !Bus_lock;
Pin 14 = !reset;
Pin 16 = !led1;
Pin 15 = !led0;
Pin 17 = !block;
Pin 18 = fpu_clk;
Pin 19 = clk;

/* Logic Equations */
$define ep_led0  (!rf_adr1 & !rf_adr0 & rf_adr24 & write3)
$define ep_led1  (!rf_adr1 &  rf_adr0 & rf_adr24 & write3 & kernel_mode)
$define ep_intr1 ( rf_adr1 & !rf_adr0 & rf_adr24 & write3 & kernel_mode)
$define ep_block ( rf_adr1 &  rf_adr0 & rf_adr24 & write3 & kernel_mode)

led0.d   = (ep_led0 & !led0 & !reset_out)
         # (!ep_led0 & led0 & !reset_out);
led1.d   = (ep_led1 & !led1)
         # (!ep_led1 & led1 & !reset_out);
intr1.d  = (ep_intr1 & !intr1 & !reset_out)
         # (!ep_intr1 & intr1 & !reset_out);
Bus_lock.oe = ep_mstr;
Bus_lock = block;
block.d  = (ep_block & !block)
         # (!ep_block & block & !reset);
reset.oe  = 'b'0;
reset     = reset_out;
clk.d     = !clk;
fpu_clk.d = !clk & !freeze;
```

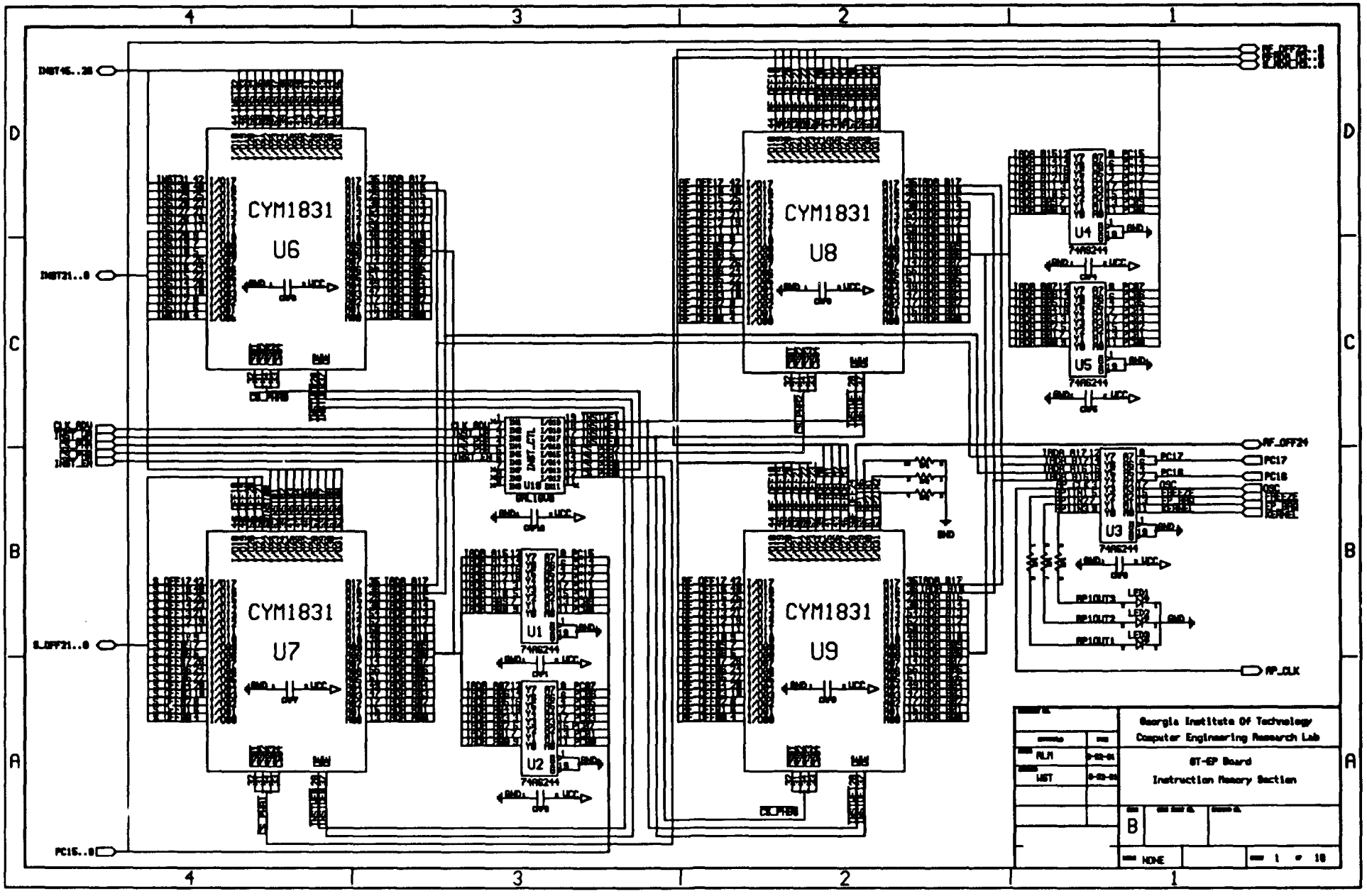
Appendix D : CUPL Listing

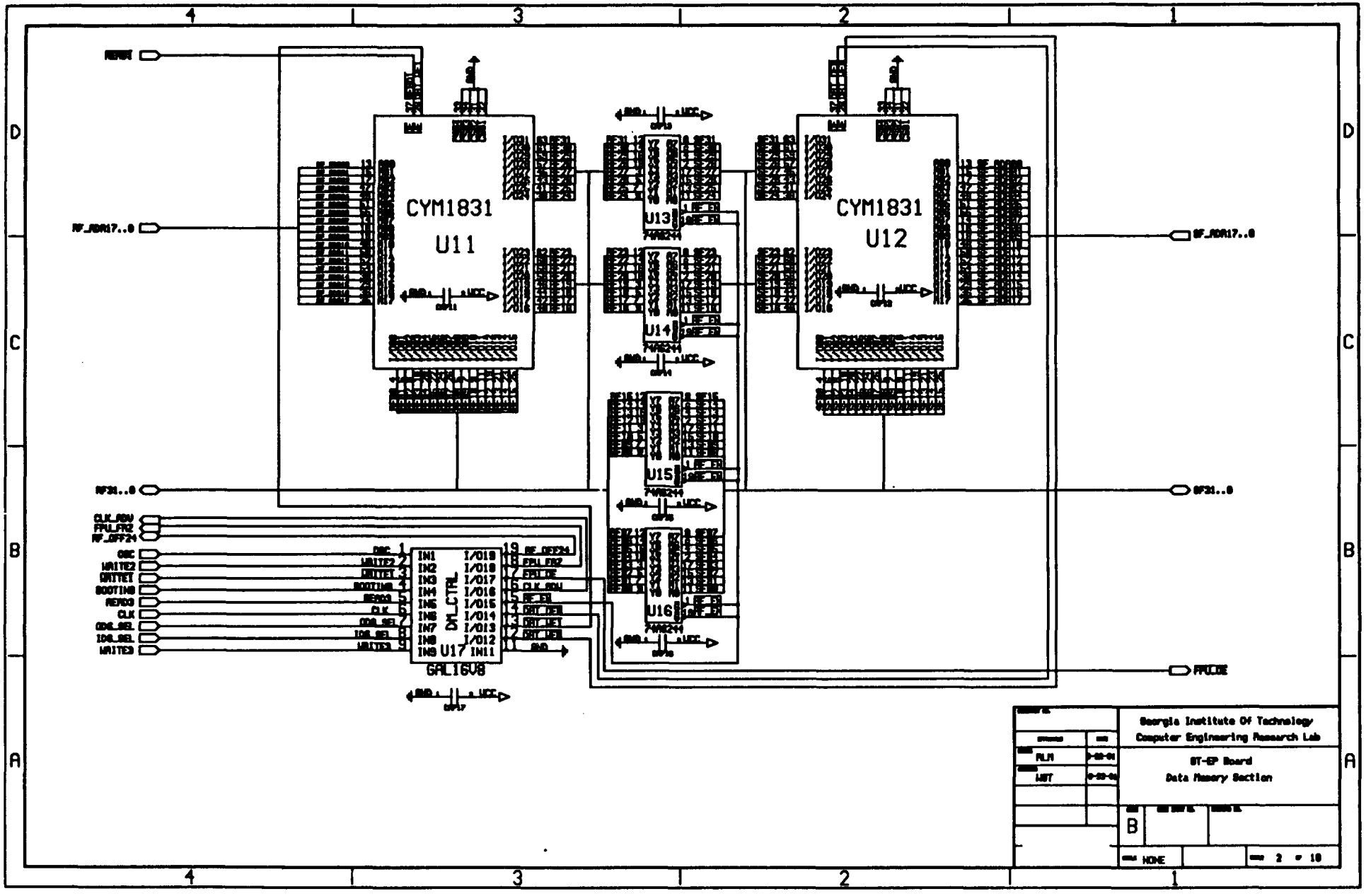
```
NAME      XBar;
Partno    000;
Date      9/11/90;
Revision  0.00;
Designer  Dr. Tan;
Company   Carl;
Assembly  Multibus interface;
Location  none;
Device    G16V8;

/* Allowable Target Device Types: GAL16V8 */

/* Inputs */

Pin 1 = ROT;
Pin 2 = WIN;
Pin 3 = !XB_FF;
Pin 4 = !XB_EF;
Pin 5 = DAV_IN;
Pin 6 = DAV_RST_IN;
Pin 7 = !reset;

/* Outputs */

Pin 12 = DAV;
Pin 13 = RFI;
Pin 14 = !XB_WR;
Pin 15 = !XB_RD;
Pin 16 = !XB_oe;
Pin 17 = XB_DIR;
Pin 18 = DAV_RST;

/* Logic Equations */

RFI     = !XB_FF;
XB_WR   = WIN;
XB_RD   = ROT & DAV_IN;
XB_DIR  = ROT;
XB_oe   = (WIN # ROT);
DAV     = ROT & !reset & !XB_EF # !DAV_RST_IN;
DAV_RST = (!ROT # reset) # !DAV_IN;
```

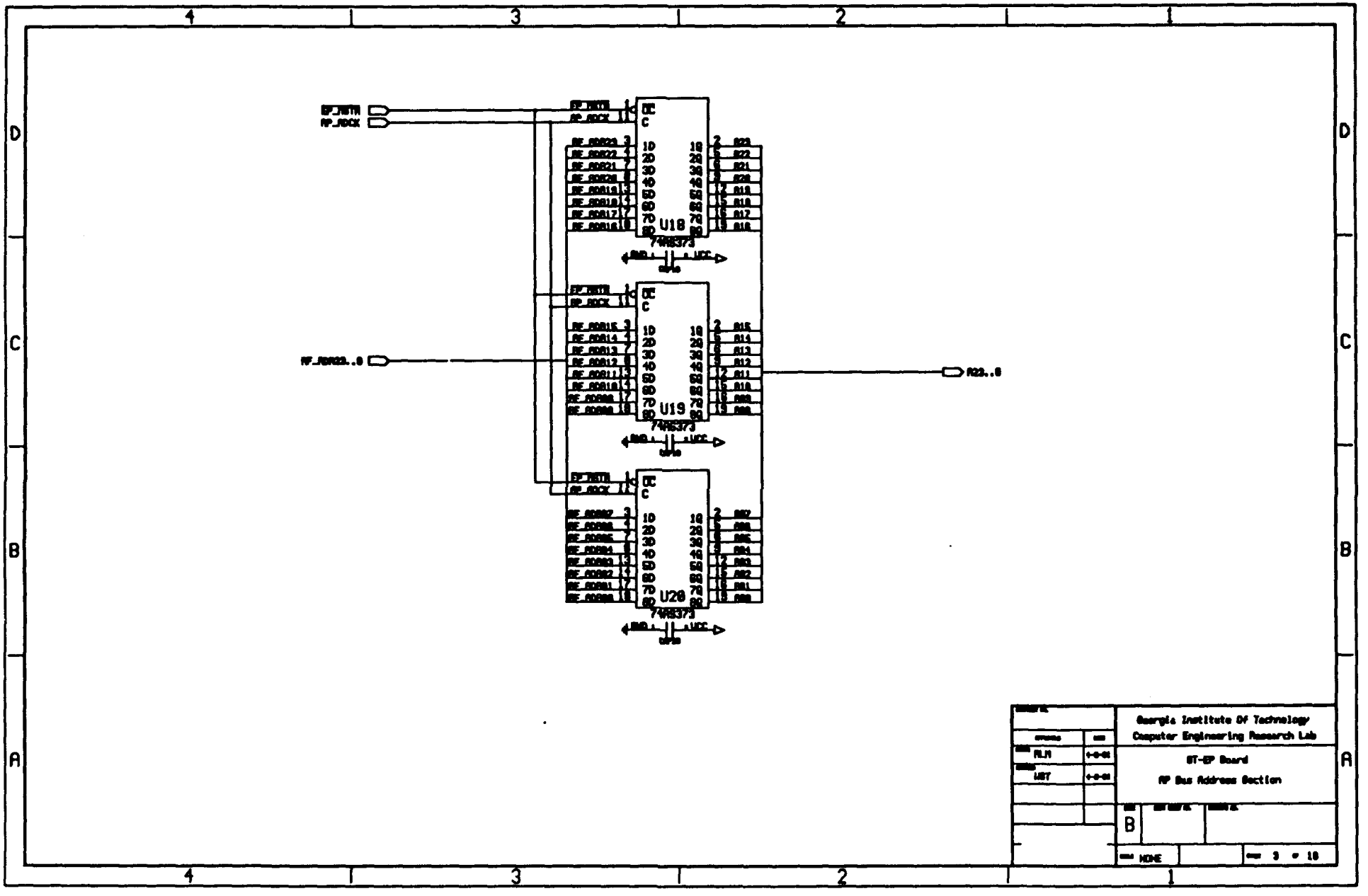
```
NAME      dec1;
Partno    000;
Date      5/7/91;
Revision  0.00;
Designer  Dr. Tan:
Company   Cerl;
Assembly  GT-DP/PFP;
Location  none;
Device    G16V8;

/* Allowable Target Device Types: GAL16V8 */

/* Inputs */

Pin 1 = !BS;
Pin 2 = HADR_4;
Pin 3 = HADR_5;
Pin 4 = HADR_6;
Pin 5 = ODSSEL;
Pin 6 = IDSSEL;
Pin 7 = !APXB_EF;
PIN 8 = !APXB_FF;
Pin 9 = !XACK;

/* Outputs */

Pin 12 = !CS_D;
Pin 13 = !CS_S;
Pin 14 = !CS_X;
Pin 15 = !APXB_RD;
PiN 16 = !APXB_WR;
Pin 17 = DATA_0;
Pin 18 = DATA_1;
Pin 19 = !DR;

/* Logic Equations */

CS_S = BS & !HADR_6 & !HADR_5 & !HADR_4;
CS_D = BS & !HADR_6 & !HADR_5 &  HADR_4;
CS_X = BS & !HADR_6 &  HADR_5 & !HADR_4;
APXB_RD = !ODSSEL &  IDSSEL & BS & !HADR_6 &  HADR_5 &  HADR_4;
APXB_WR =  ODSSEL & !IDSSEL & BS & !HADR_6 &  HADR_5 &  HADR_4;

DATA_1.OE = BS & !ODSSEL &  IDSSEL &  HADR_6 & !HADR_5 & !HADR_4;
DATA_1 = APXB_EF;
DATA_0.OE = BS & !ODSSEL &  IDSSEL &  HADR_6 & !HADR_5 & !HADR_4;
DATA_0 = APXB_FF;
DR.OE = BS;
DR = BS & (!HADR_6 & !HADR_5 # !HADR_6 & HADR_5 & !HADR_4) & XACK #
     BS & !HADR_6 &  HADR_5 &  HADR_4 #
     BS &  HADR_6 & !HADR_5 & !HADR_4;
```

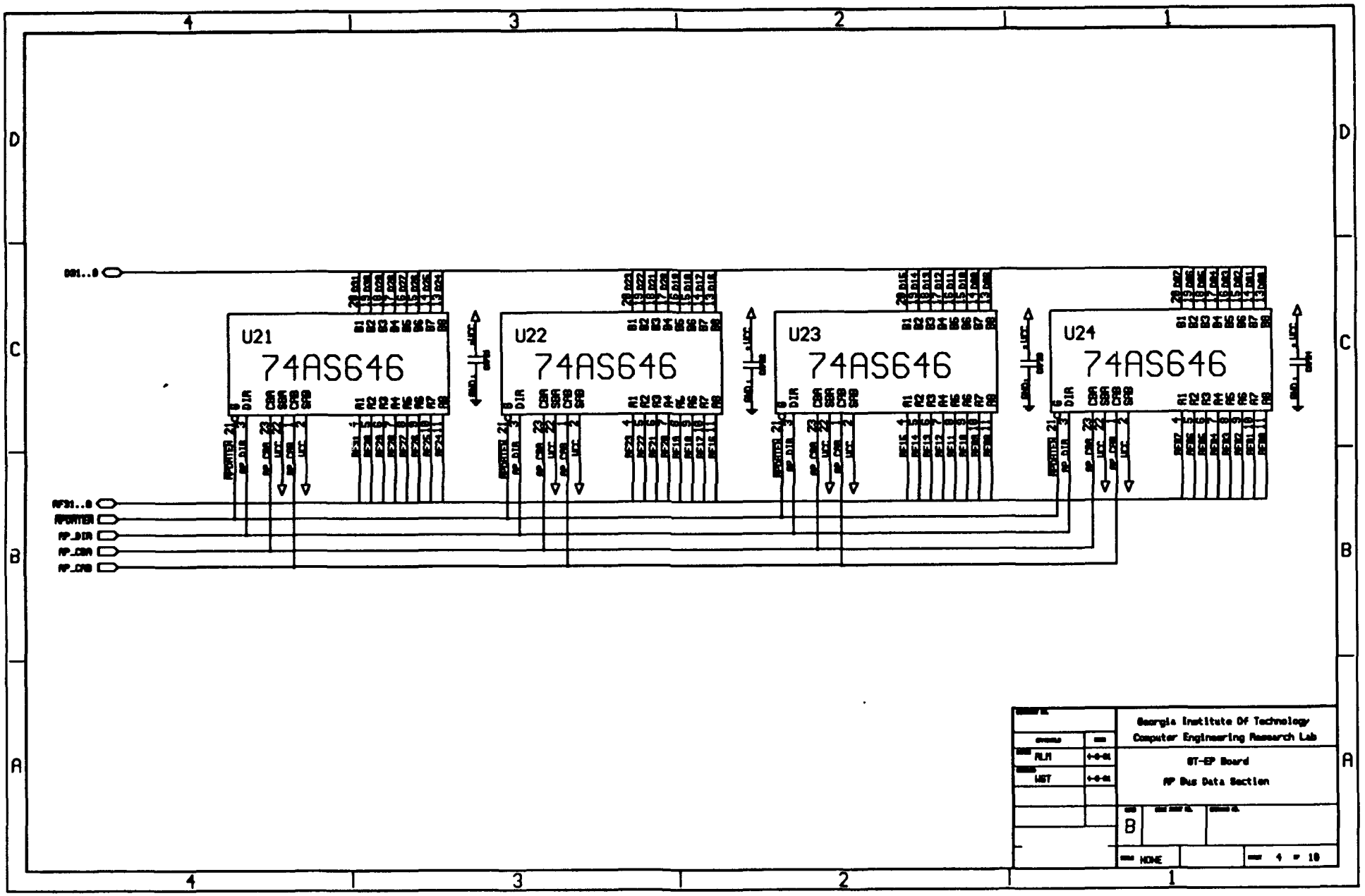
```
NAME     dec2;
Partno   000;
Date     5/7/91;
Revision 0.00;
Designer Dr. Tan;
Company  Cerl;
Assembly GT-DP/PFP;
Location none;
Device   G16V8;

/* Allowable Target Device Types: GAL16V8 */

/* Inputs */

Pin 1 = DS_0;
Pin 2 = DS_1;
Pin 3 = DS_2;
Pin 4 = DS_3;
Pin 5 = ODSSEL;
Pin 6 = IDSSEL;

/* Outputs */

Pin 12 = !MWTC;
Pin 13 = !MRDC;
Pin 14 = !BS;

/* Logic Equations */

BS   = (DS_3 & DS_2 & !DS_1 & !DS_0);
MWTC =  ODSSEL & !IDSSEL;
MRDC = !ODSSEL & IDSSEL;
```

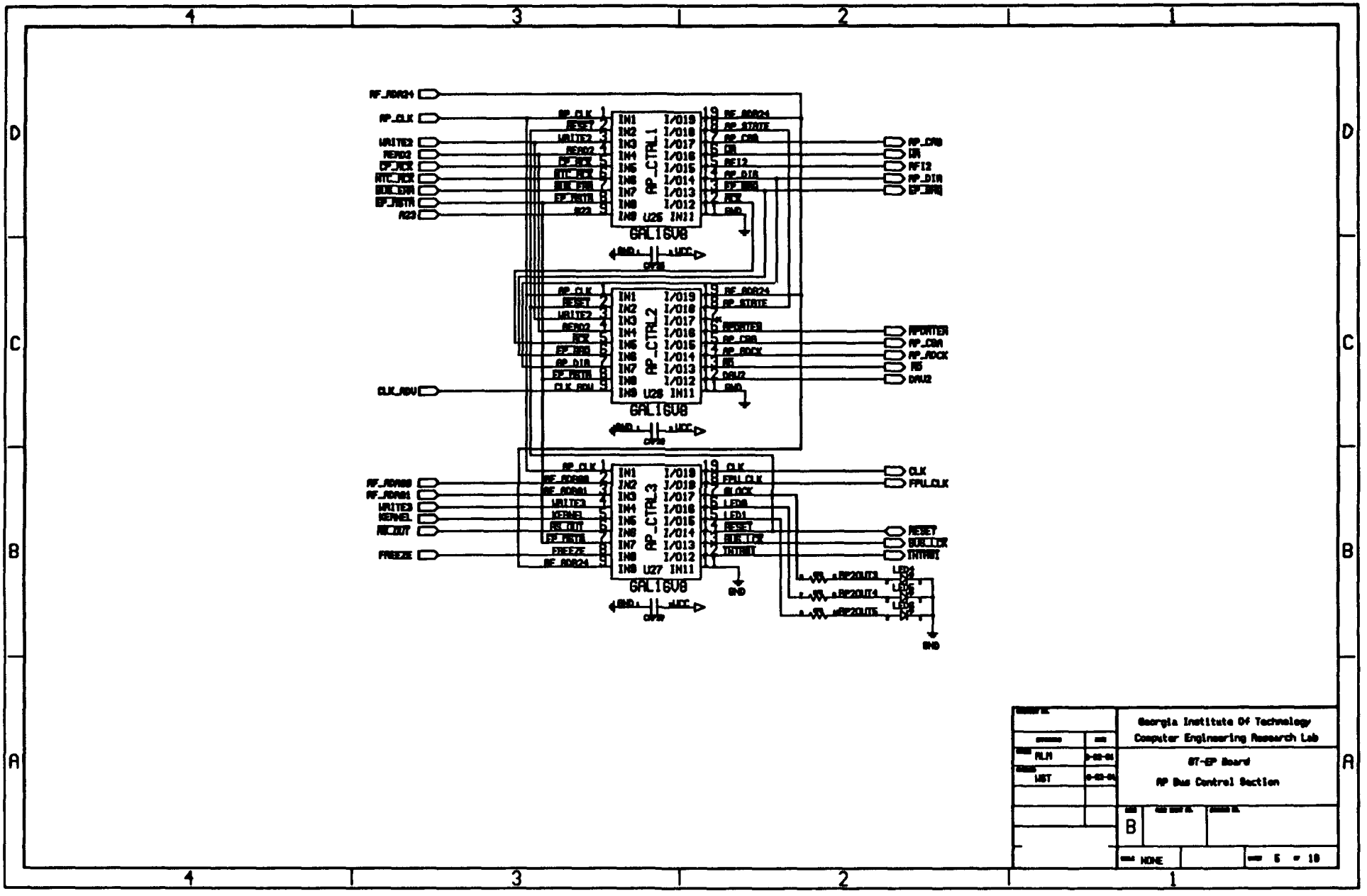
Appendix E : Board Schematics


CYM1831
U6
CYM1831
U8
CYM1831
U7
CYM1831
U9
U1
U2
U3
U4
U5
74HC244
Georgia Institute Of Technology
Computer Engineering Research Lab
GT-EP Board
Instruction Memory Section

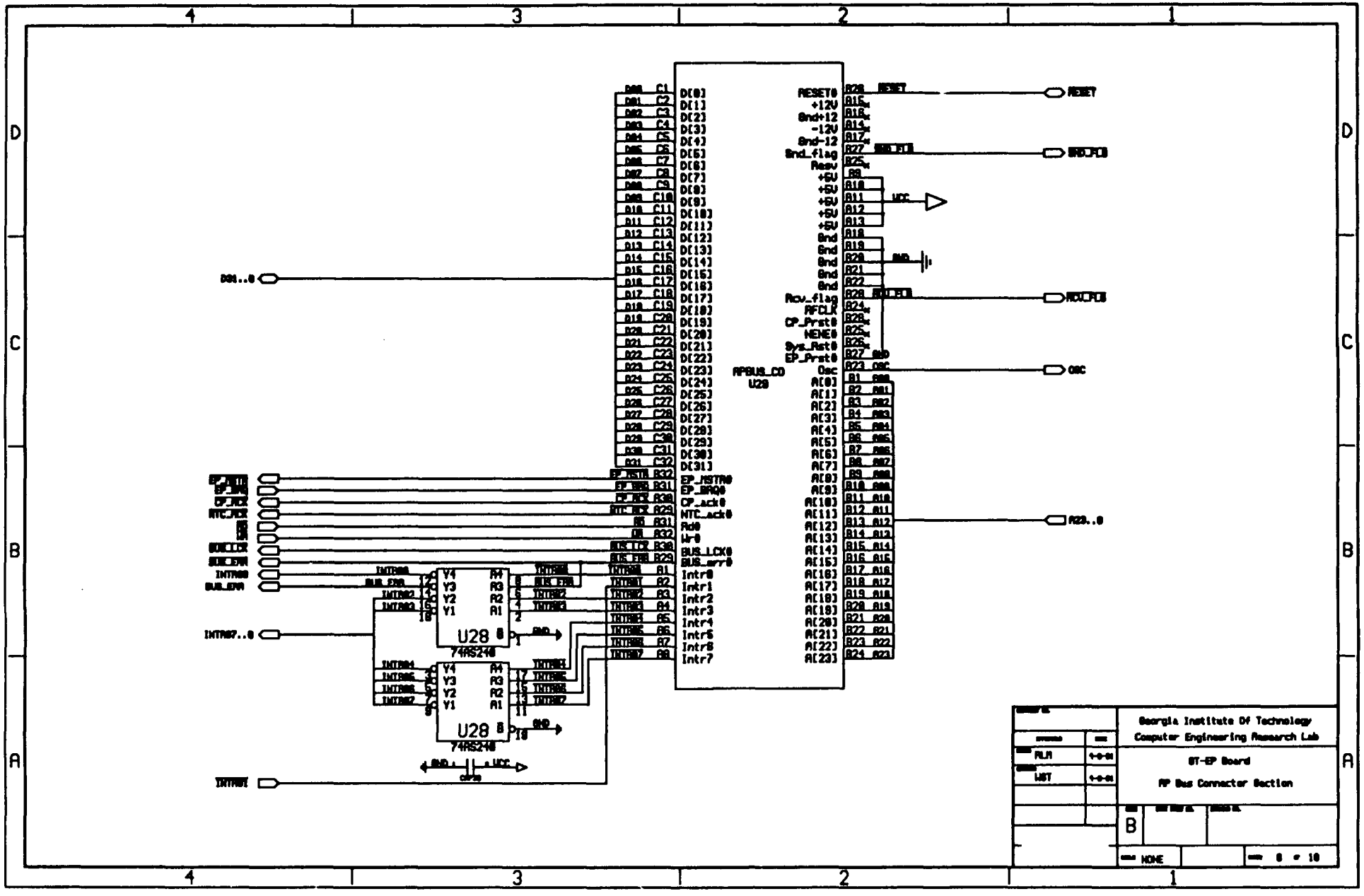

CYM1831
U11
CYM1831
U12
U13
U14
U15
U16
U17
GAL16V8
DYLCTRL
Georgia Institute Of Technology
Computer Engineering Research Lab
DT-EP Board
Data Memory Section

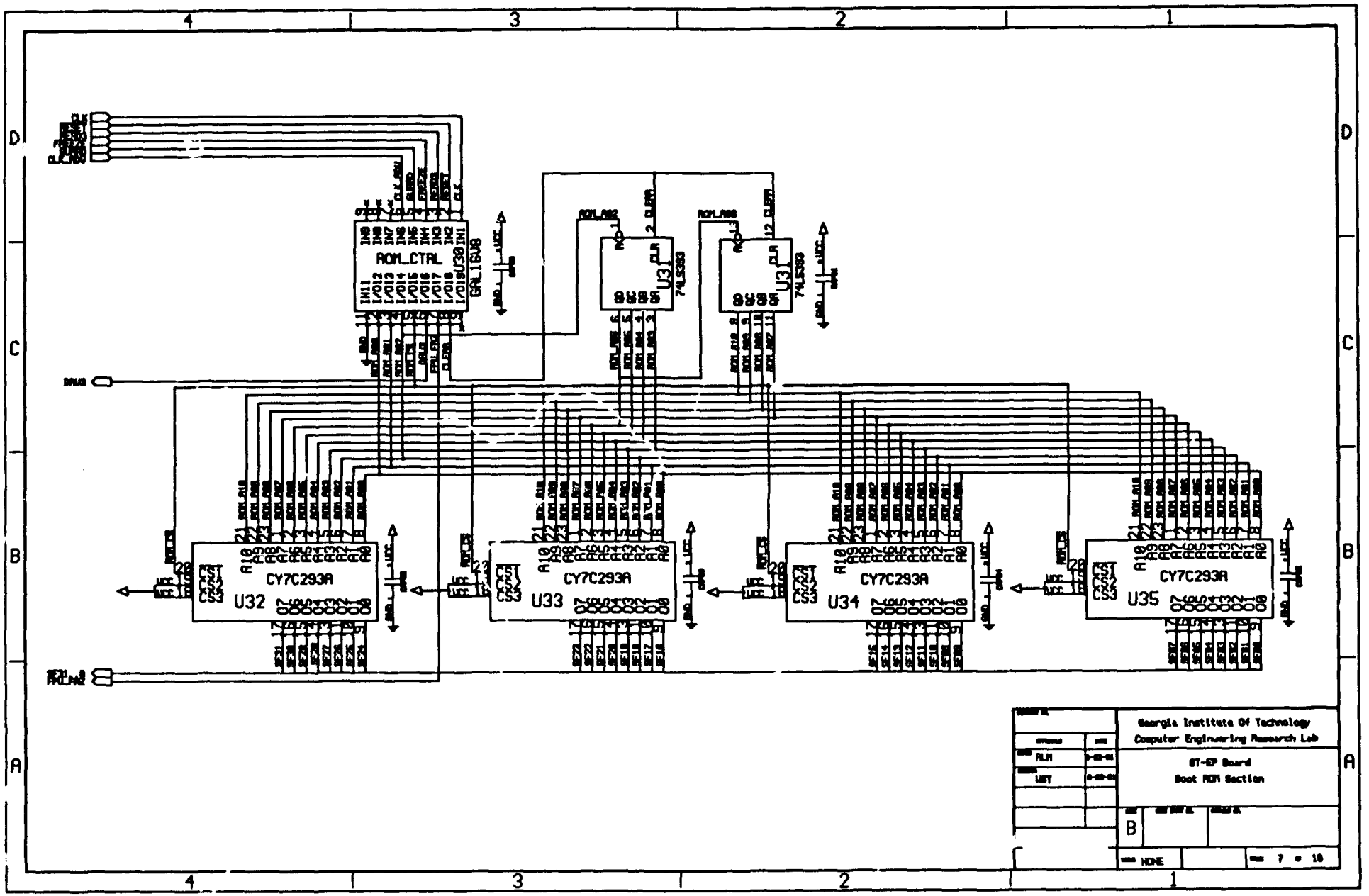

U18
74HS373
U19
74HS373
U20
74HS373
R23..0
Georgia Institute Of Technology
Computer Engineering Research Lab
RT-EP Board
RP Bus Address Section
B
HOME

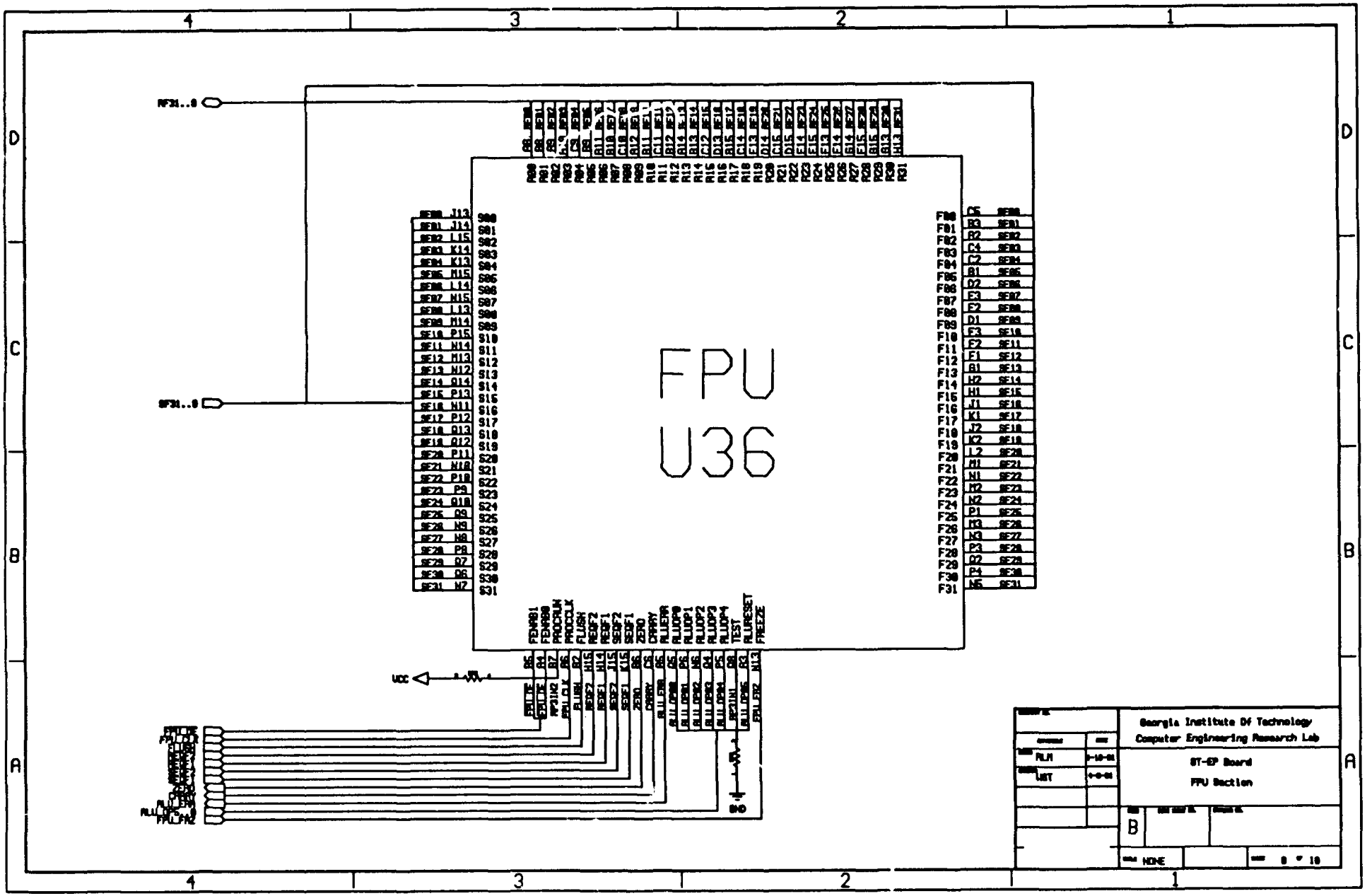

U21
74AS646
U22
74AS646
U23
74AS646
U24
74AS646
Georgia Institute Of Technology
Computer Engineering Research Lab
GT-EP Board
AP Bus Data Section

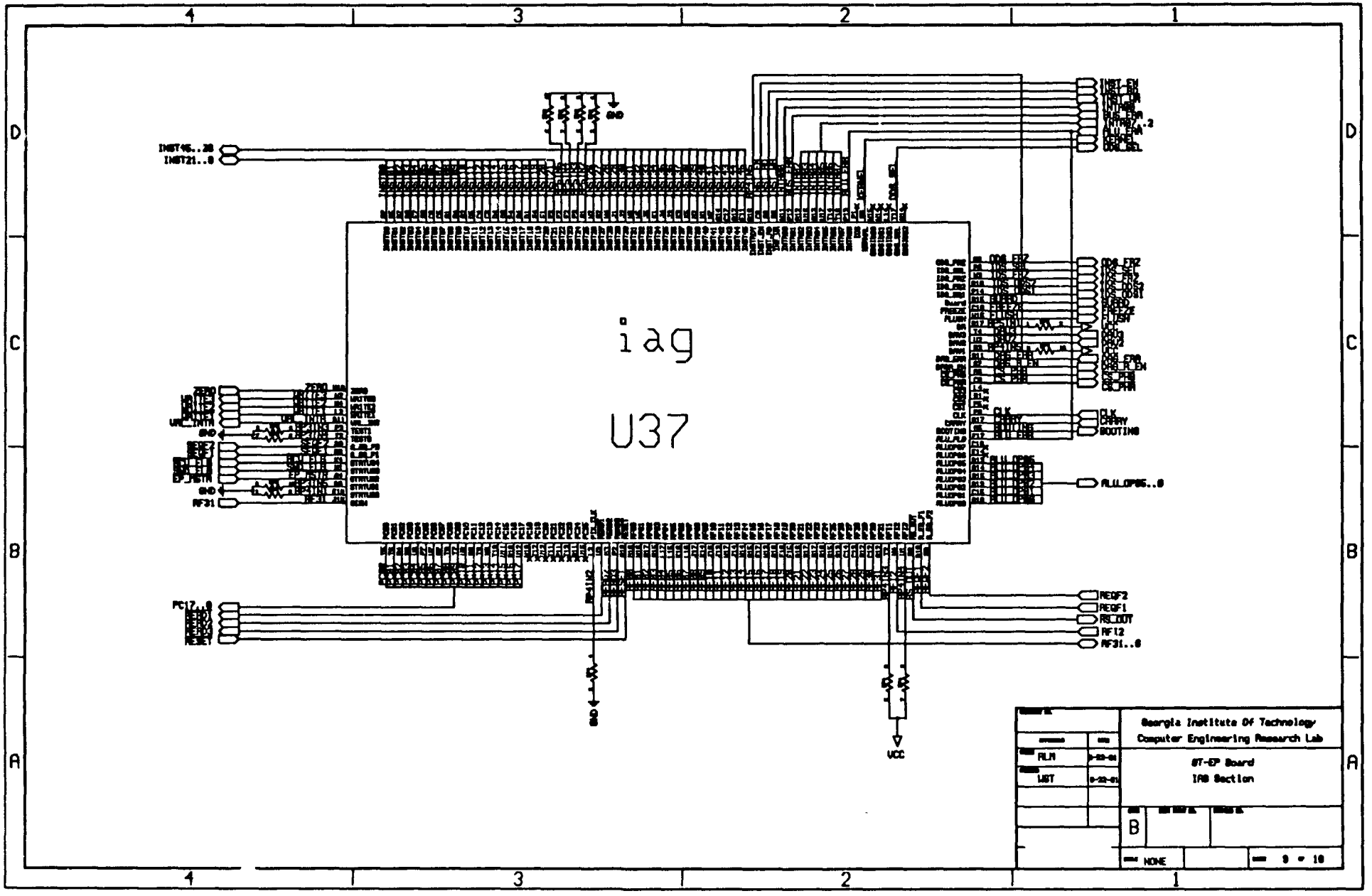


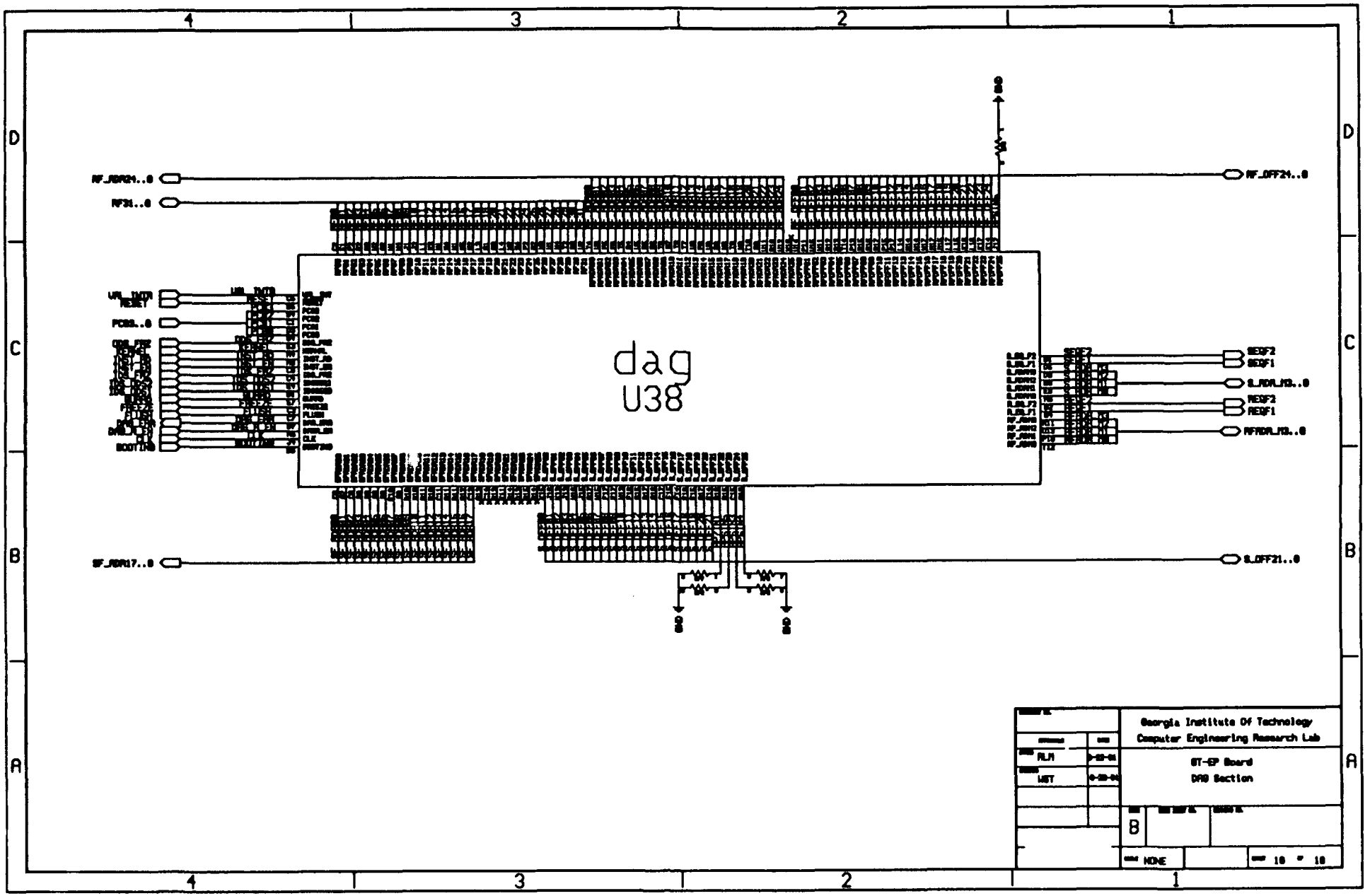
Georgia Institute Of Technology	
Computer Engineering Research Lab	
RLM	
MST	
87-EP Board	
AP Bus Control Section	
B	
NONE	5 of 10


Georgia Institute Of Technology
Computer Engineering Research Lab
GT-EP Board
AP Bus Connector Section
U28
74HS240
RESET
GND
VCC
OSC
RCV_FLG
BRD_FLG
INTR7..0
D31..0
A23..0

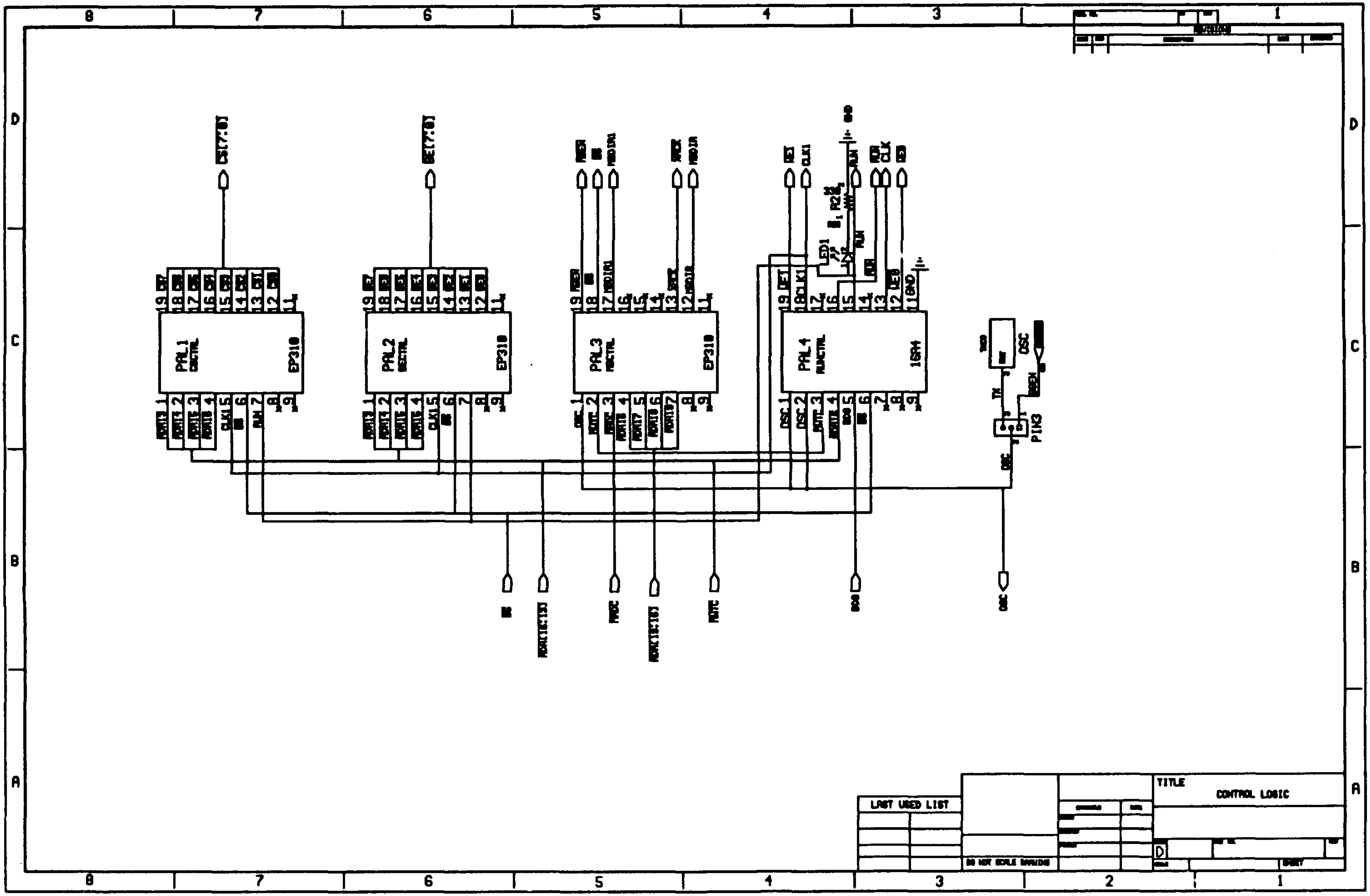

ROM_CTRL
GAL16V8
U30
U3
74LS393
CY7C293A
U32
U33
U34
U35
Georgia Institute Of Technology
Computer Engineering Research Lab
RLH
MST
GT-EP Board
Boot ROM Section
B
NONE

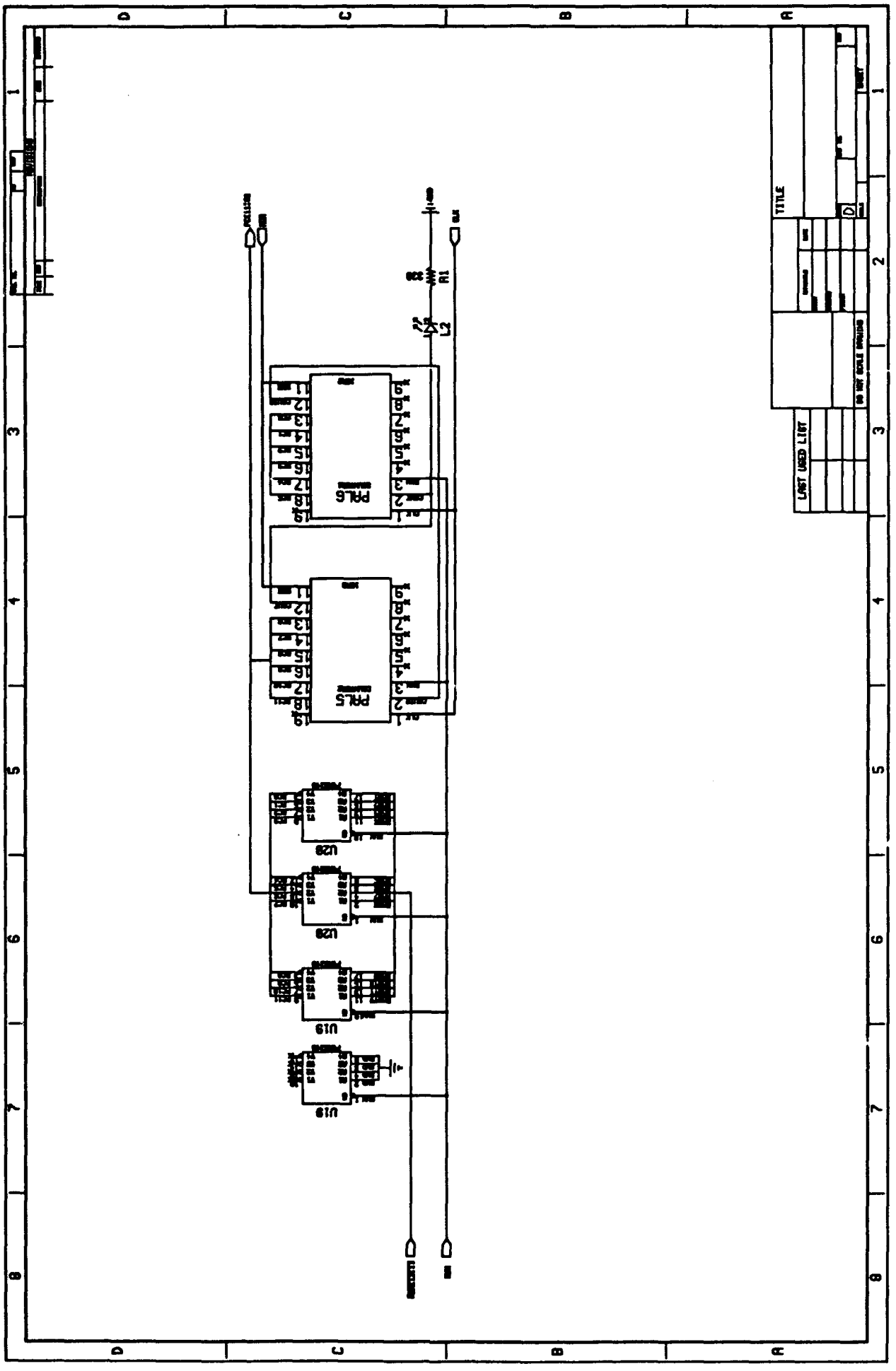

FPU
U36
Georgia Institute Of Technology
Computer Engineering Research Lab
ST-EP Board
FPU Section

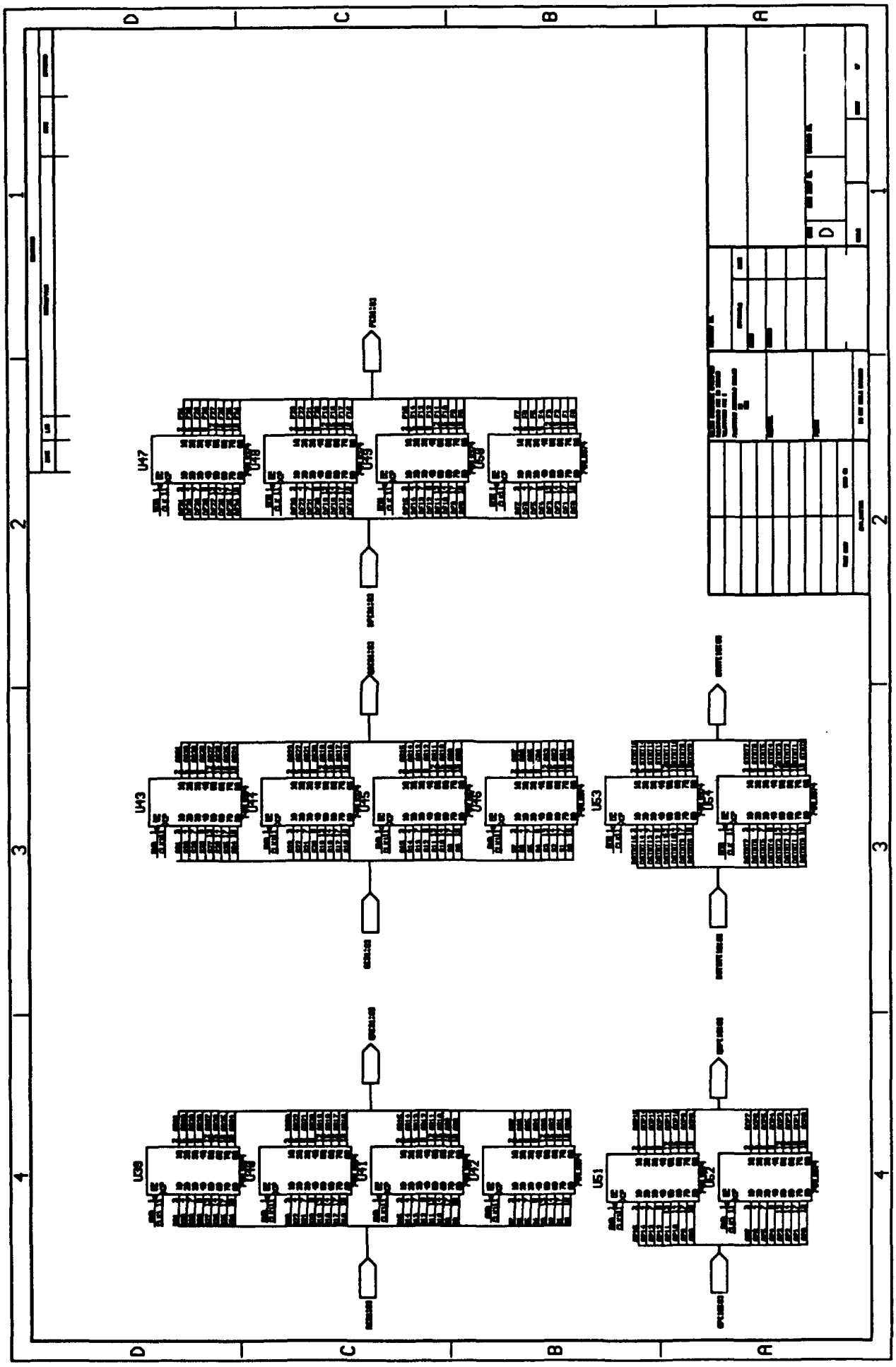

iag
U37
Georgia Institute Of Technology
Computer Engineering Research Lab
87-EP Board
IAG Section
B
NONE

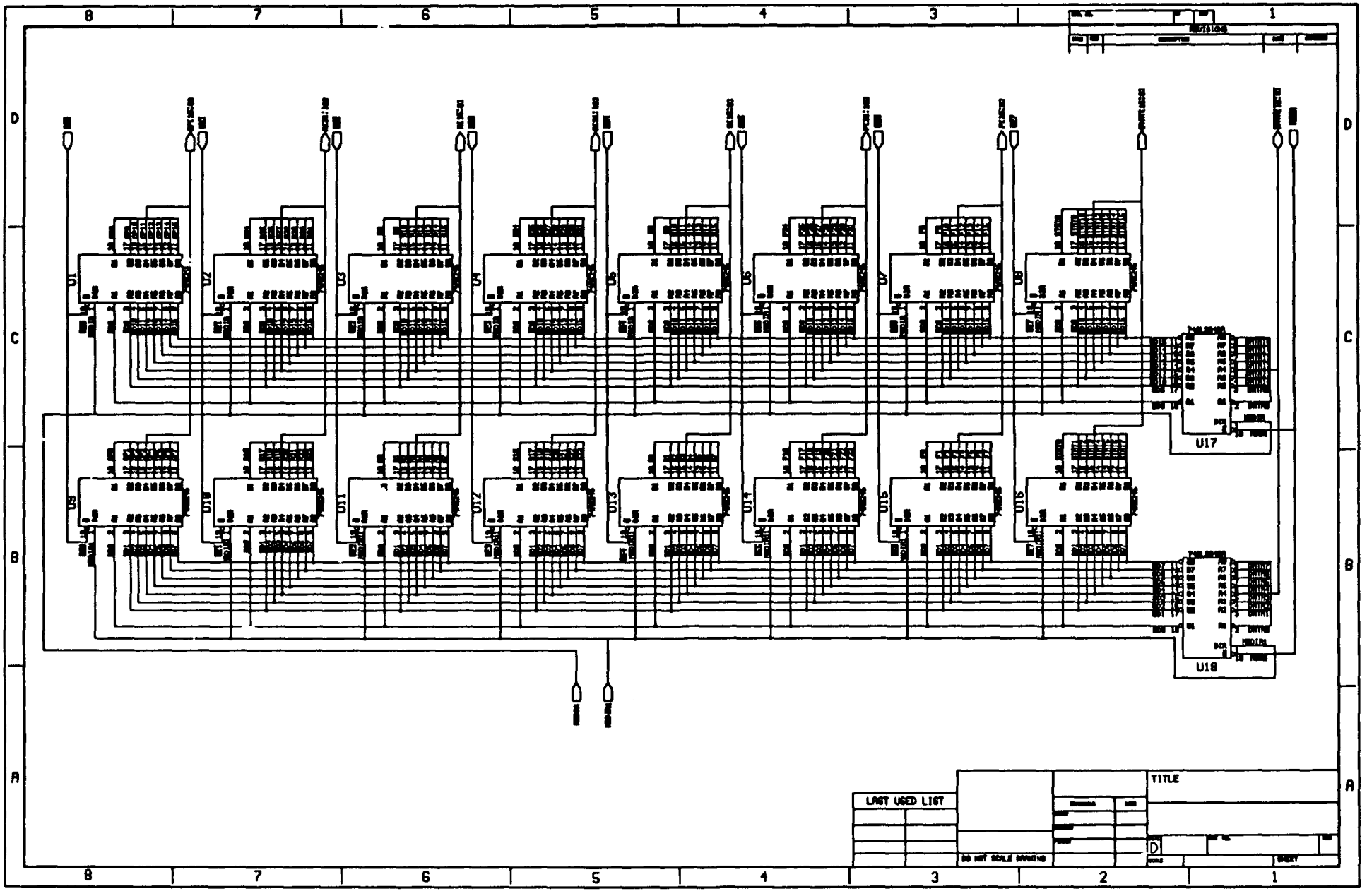

dag
U38
Georgia Institute Of Technology
Computer Engineering Research Lab
GT-EP Board
DAG Section

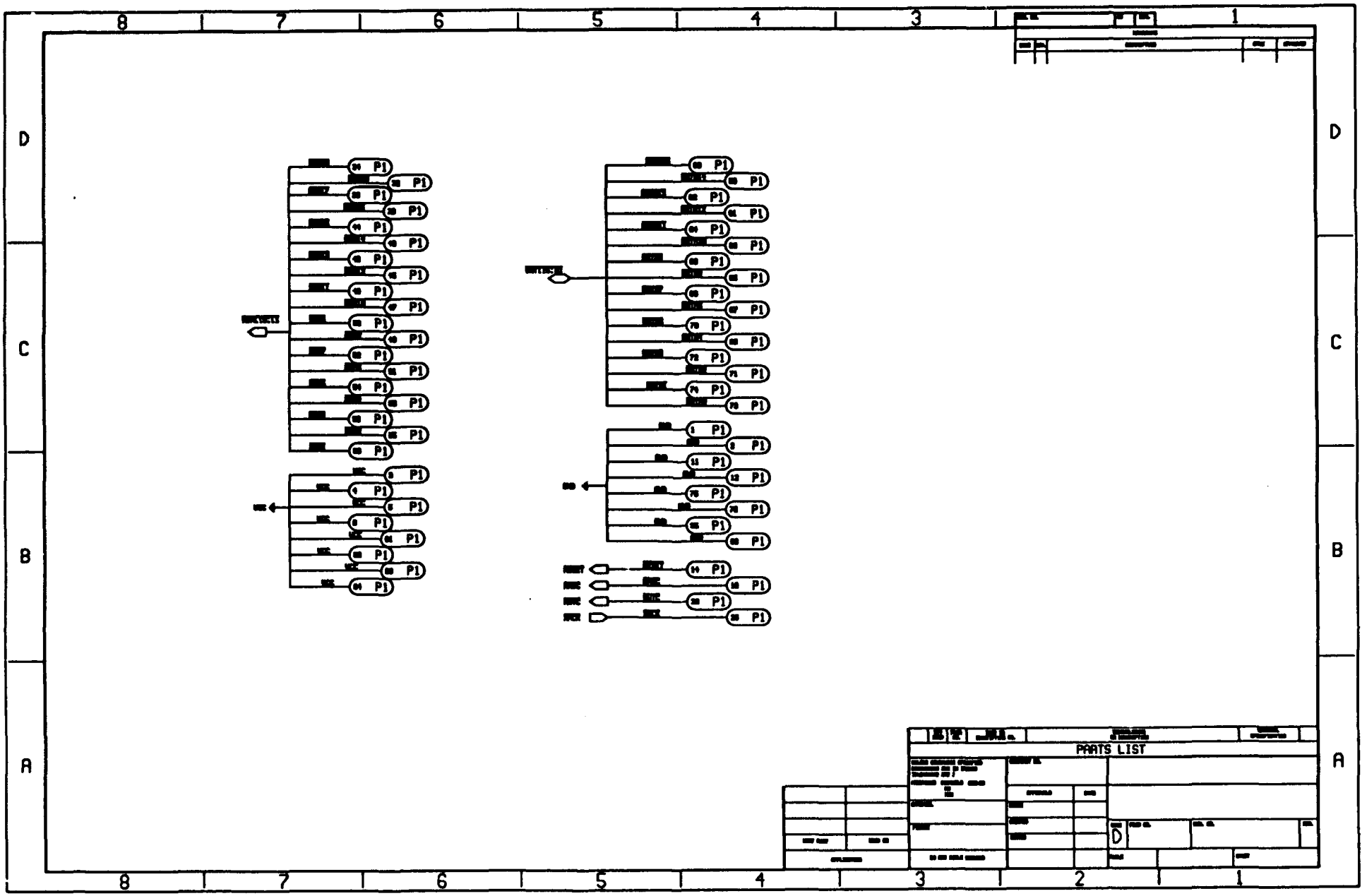
Appendix F : Board Schematics

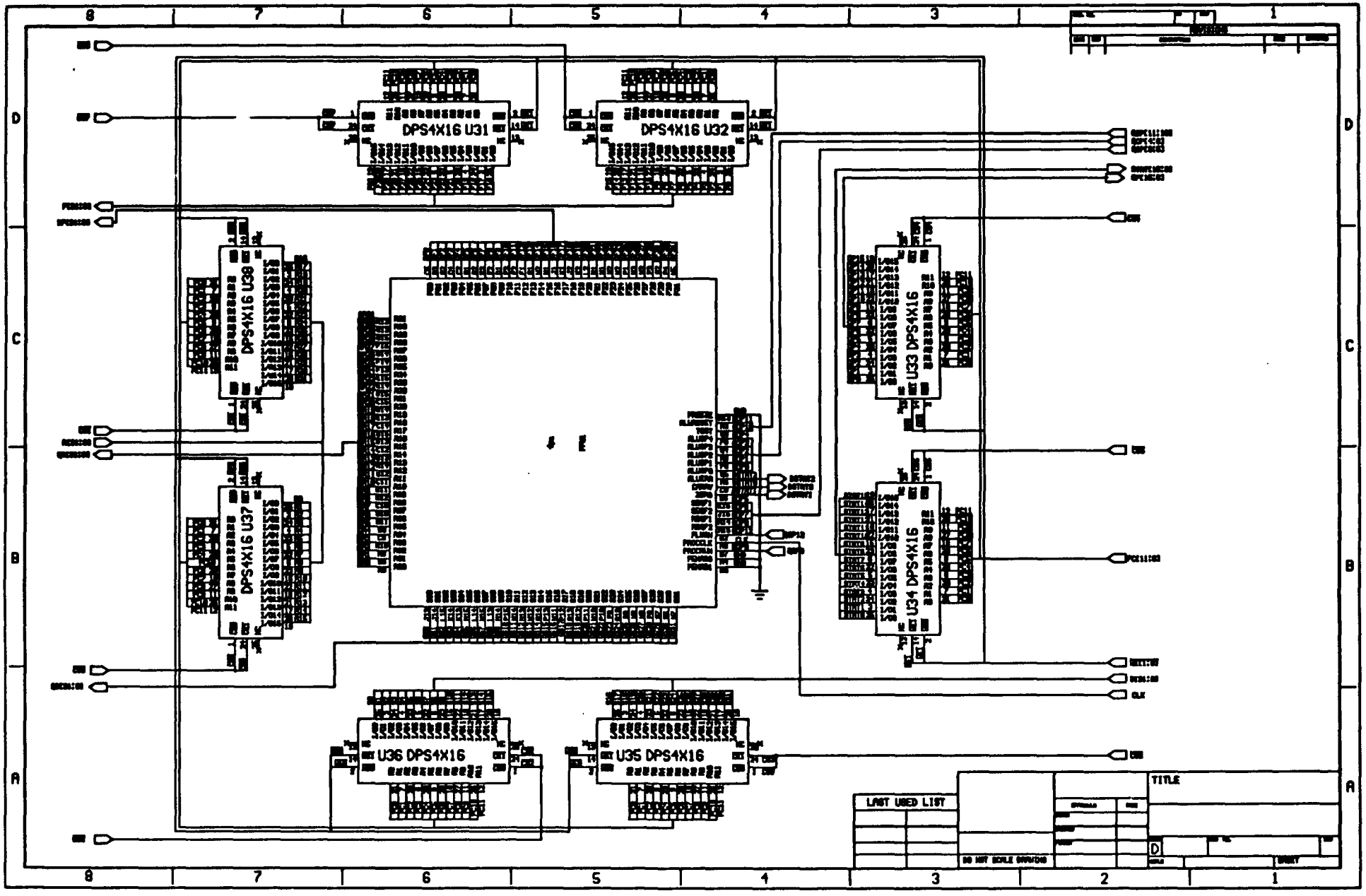

TITLE
CONTROL LOGIC
LAST USED LIST
PAL1
EP310
PAL2
EP310
PAL3
EP310
PAL4
16R4
PIN3
OSC
TX
LED1
R28
GND
CLK
CLK1
CS[7:0]
BE[7:0]


TITLE
LAST USED LIST
PRL6
PRL5
U28
U28
U19
U19

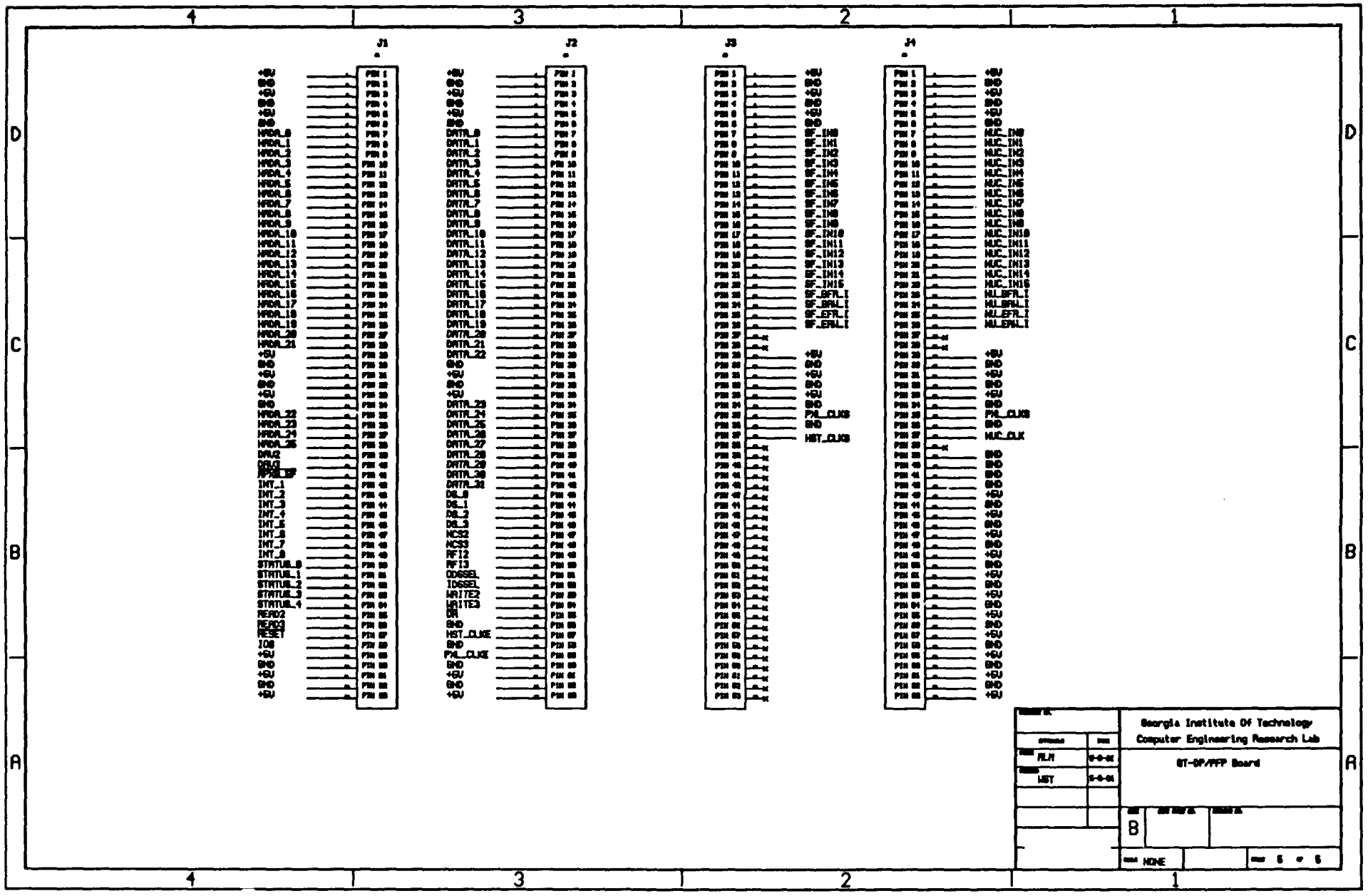

D
C
B
A
1
2
3
4
U47
U48
U49
U50
U43
U44
U45
U46
U53
U54
U38
U40
U41
U42
U51
U52

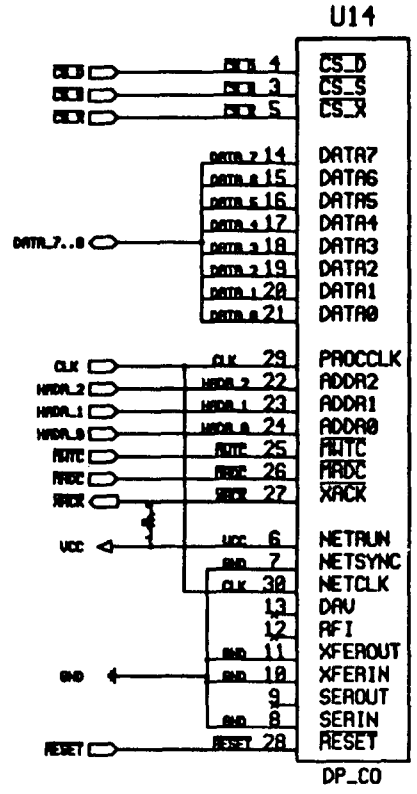

U17
U18
LAST USED LIST
TITLE
DO NOT SCALE DRAWING


PARTS LIST


DPS4X16 U31
DPS4X16 U32
U33 DPS4X16
U34 DPS4X16
U36 DPS4X16
U35 DPS4X16
LAST USED LIST
TITLE

Appendix G : GT–DP/PFP Board Schematics


J1
J2
J3
J4
Georgia Institute Of Technology
Computer Engineering Research Lab
GT-DP/FFP Board
RLM
MST
B
NONE



Georgia Institute Of Technology
Computer Engineering Research Lab

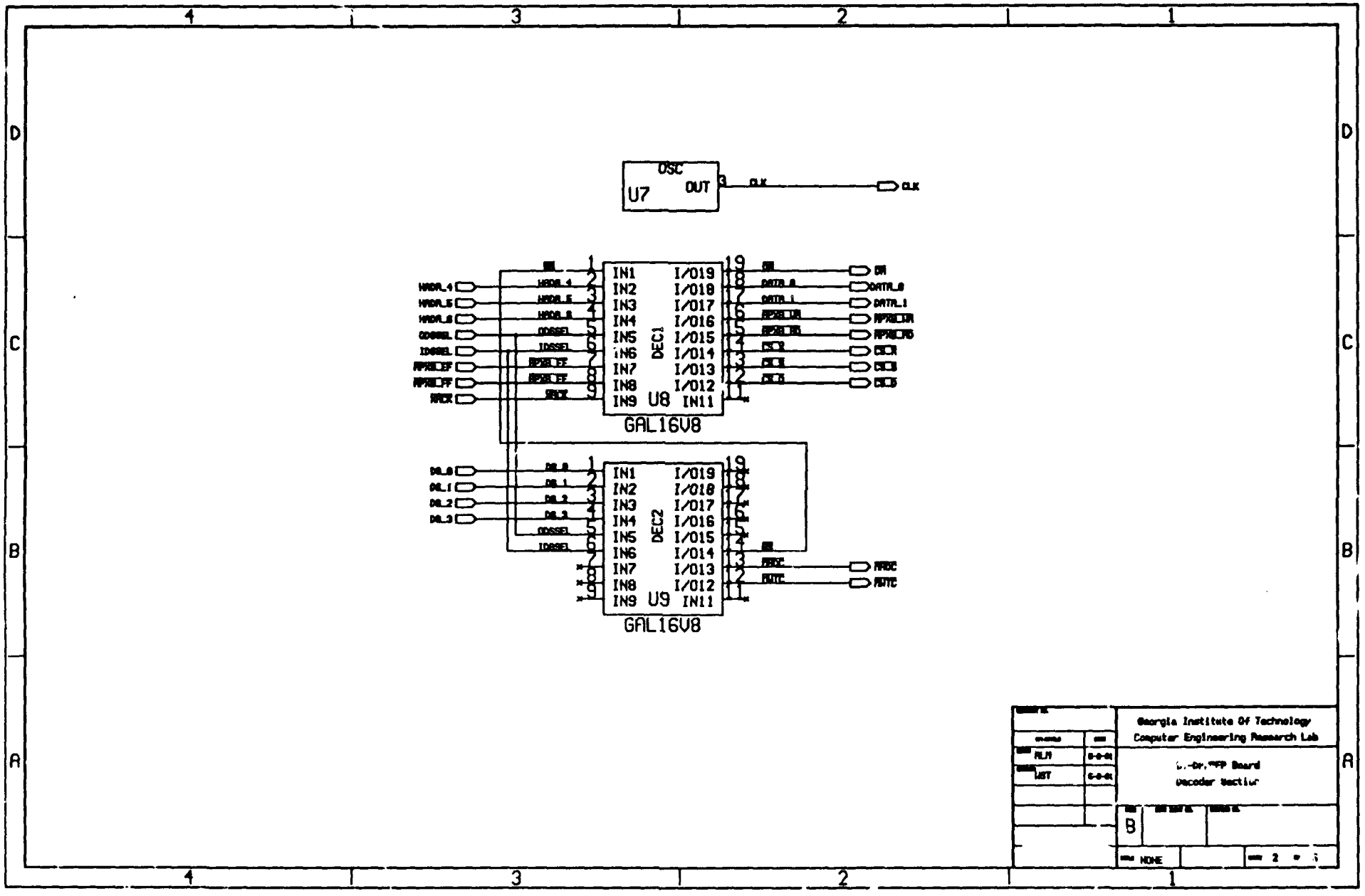
RLM 8-8-91
WST 8-8-91

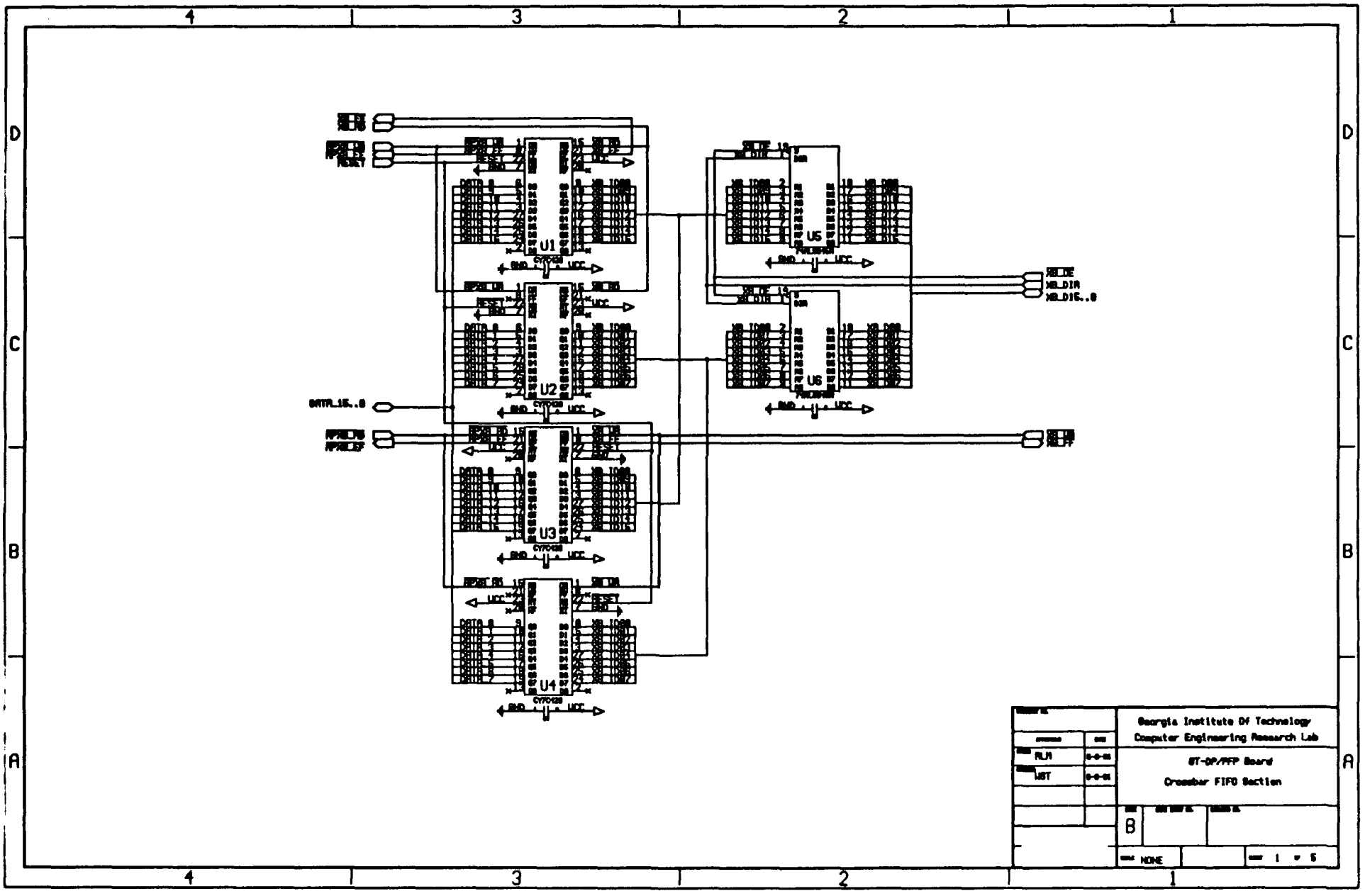
BT-DP/FFP Board
Data Processor Connector

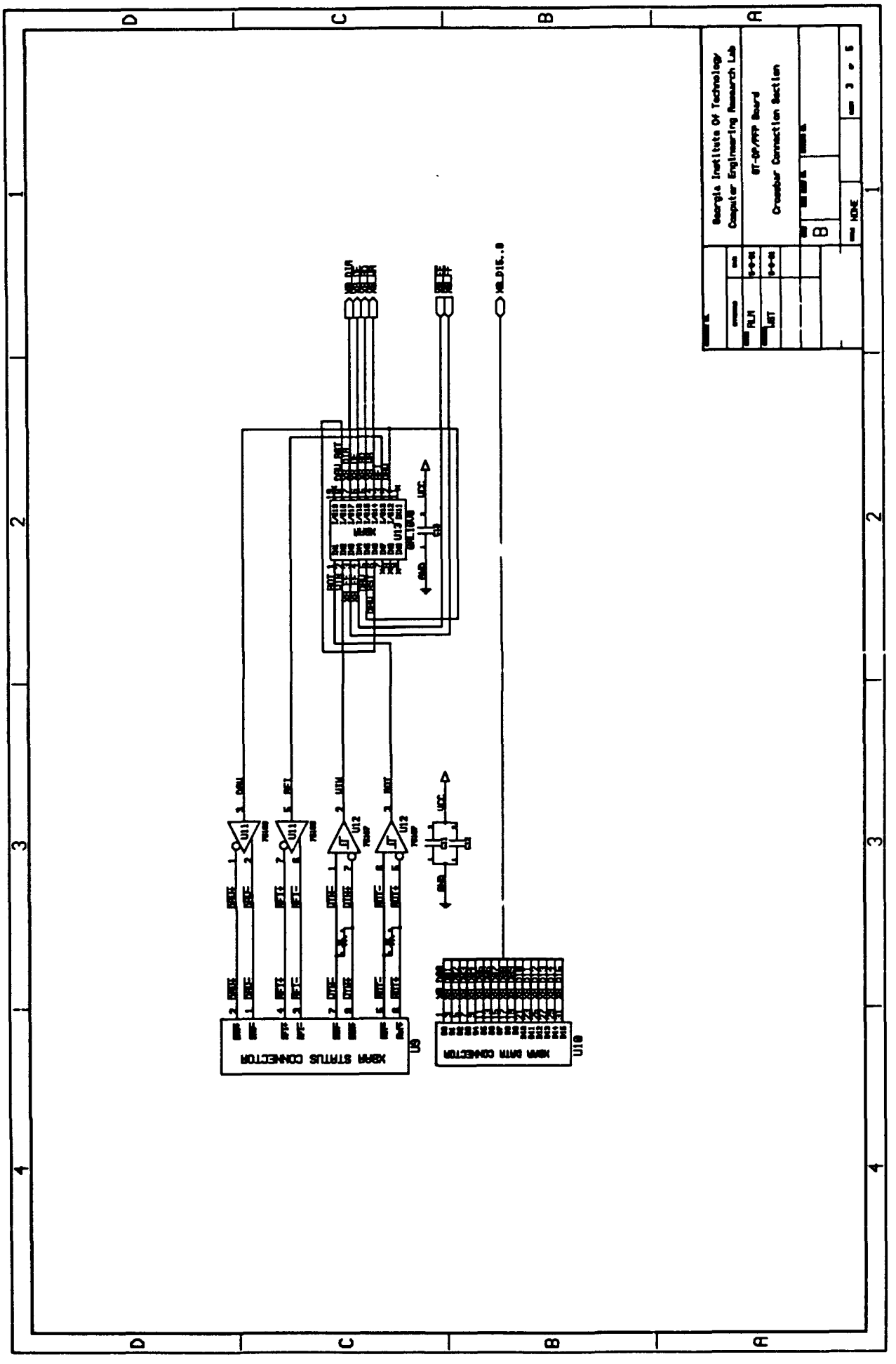
B

NONE

4 of 5


OSC
U7
OUT
CLK
CLK
IN1
IN2
IN3
IN4
IN5
IN6
IN7
IN8
IN9
I/O19
I/O18
I/O17
I/O16
I/O15
I/O14
I/O13
I/O12
IN11
DEC1
U8
GAL16V8
DEC2
U9
GAL16V8
Georgia Institute Of Technology
Computer Engineering Research Lab
B
NONE


4
3
2
1
D
C
B
A
U1
U2
U3
U4
U5
U6
XMB_DIR
XMB_D16..0
DATA_15..0
Georgia Institute Of Technology
Computer Engineering Research Lab
RLH
MST
ST-DP/FFP Board
Crossbar FIFO Section
B
NONE
1 of 5


Georgia Institute Of Technology
Computer Engineering Research Lab
RT-DP/FPP Board
Crossbar Connection Section
XBAR STATUS CONNECTOR
XBAR DATA CONNECTOR