8 BIT REGISTER-BASED ALU ON FPGA

By

JALEN ONG SUET YENG

FINAL PROJECT REPORT

Submitted to the Electrical & Electronics Engineering Programme
in Partial Fulfillment of the Requirements
for the Degree
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Universiti Teknologi Petronas
Bandar Seri Iskandar
31750 Tronoh
Perak Darul Ridzuan

CERTIFICATION OF APPROVAL

8 BIT REGISTER-BASED ALU ON FPGA

by

Jalen Ong Suet Yeng

A project dissertation submitted to the
Electrical & Electronics Engineering Programme
Universiti Teknologi PETRONAS
in partial fulfilment of the requirement for the
Bachelor of Engineering (Hons)
(Electrical & Electronics Engineering)

Approved:

[illegible]

Dr. Yap Vooi Voon
Project Supervisor

UNIVERSITI TEKNOLOGI PETRONAS
TRONOH, PERAK

June 2007

CERTIFICATION OF ORIGINALITY

This is to certify that I am responsible for the work submitted in this project, that the original work is my own except as specified in the references and acknowledgements, and that the original work contained herein have not been undertaken or done by unspecified sources or persons.

[illegible]

Jalen Ong Suet Yeng

ABSTRACT

The objective of the project is to implement the ALU of an 8 bit register-based CPU on FPGA. The success of this project will be an asset to the education of computer architectures. Exposure to FPGA design will also become invaluable as the demand for embedded system increases. The scope of study involves gaining understanding of the architecture of the CPU and mastering HDL for FPGA design. The methodologies outlined include functional and timing analysis of the ALU, construction of test jigs for hardware interface with UP2 development board, hardware tests and troubleshooting, programming TTL components in Verilog, construction of interface with TTL CPU, interfacing with TTL CPU and implementing the control card on FPGA. A functional ALU was implemented on FPGA. Static tests have shown that the ALU unit is functioning.

ACKNOWLEDGMENTS

I would like to take this opportunity to express my deepest gratitude for my supervisor, Dr.Yap Vooi Voon for his wisdom and guidance throughout the project. He was not only an academic advisor but also someone who dares me to dream. He is is always there to encourage and put things in perspective in the face of adversity. Under his tutelage, I have learnt much, not only technically but also in shaping the direction of my future. I would also like to thank Mr. Patrick Sebastian, who is unofficially the next-door supervisor and who has been extremely helpful. His generosity ranges from unreserved technical advice to the infinite loaning of his personal stationery. Not forgetting, Mr. Lo, the third member of the trio who never hesitates to kick me back on track when I show any sign of straying. I want to thank Aidil Jazmi for patiently answering my streams of questions that haunted him even after he left UTP. I also want to thank Kak Hawa and all other technicians who put up with my incessant and annoying questions on everything under the sun and who kept loaning me everything from the lab even though they never seem to come back. Thanks also to my friends whom I rely upon for survival in UTP. I am greatly indebted.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ALO	**Arithmetic and Logic Operations**
ALU	**Arithmetic Logic Unit**
ASIC	**Application-Specific Integrated Circuit**
CPU	**Central Processing Unit**
FPGA	**Field Programmable Gate-Array**
TTL	**Transistor-Transistor Logic**

CHAPTER 1
INTRODUCTION

Computers have become a requisite. As a result of the boundless capabilities of modern computers, they are thought to be very convoluted and intellectual devices. The thought of building a computer is inconceivable. Essentially, the microprocessor or central processing unit (CPU) of a computer is made out of simple commands to move data around, perform simple math (add, subtract, multiply, and divide), bring data into the CPU from the outside world, and send data out of the CPU to the outside world. The power of the computer only surfaces with its ability to execute these simple instructions expeditiously. As its complexity increases many folds in a matter of months, it becomes a technology that is inexplicable. It is the aim of the project to go down to basics by starting from scratch and building a minicomputer and understanding the architecture of it. An 8 bit register-based CPU on TTL chips has already been implemented. The challenge is to implement the arithmetic logic unit (ALU) into a single chip in Field Programmable Gate Array (FPGA).

1.1 Background of Study

The goal is to implement the arithmetic logic unit (ALU) of an 8 bit register based CPU on FPGA. By 8 bits, it means the CPU can process information 8 bits at a time. For example it can subtract or add two 8 bit numbers at one instruction cycle. An FPGA is a semiconductor device containing programmable logic components and programmable interconnects. The programmable logic components can be programmed to duplicate the functionality of basic logic gates such as AND, OR, XOR, NOT or more complex combinational functions such as decoders.

A hierarchy of programmable interconnects allows the logic blocks of an FPGA to be interconnected as needed by the system designer, somewhat like a one-chip programmable breadboard. FPGAs are generally slower than their application-specific integrated circuit (ASIC) counterparts, can't handle as complex a design, and draw more power. However, there are several advantages such as a shorter time to market, ability to re-program in the field to fix bugs, and lower non-recurring engineering costs. This propels the aspiration to develop the ALU of the CPU on FPGA (a single chip) as the initial step.

1.2 Problem Statement

The main aim of the project is to implement the ALU in FPGA. It is envisaged that this would open up opportunities for further research in computer architecture. In addition, it also eases the teaching of computer architecture related courses as they present the most basic computer architecture. The first step that has been taken is to construct the whole CPU in TTL chips. That has already been achieved. It would then serve as a testbed for the chip that is to be implemented in FPGA. The next step now is to implement the ALU in FPGA.

1.3 Objective and Scope of Study

The objective of the project is to implement the ALU of an 8 bit register based CPU on FPGA. The CPU that we are targeting to implement our ALU on runs at 3MHz and is similar in capabilities and performance as the 8086. The ALU works on both 8 bits and 16 bits operations. To accomplish these objectives, one has to gain a thorough understanding on the architecture of the ALU to enable one to simulate and test run each part of the ALU separately. The conception enables one to know the expected results of a successful simulation. Next, one is required to master the hardware description language (HDL) to implement the ALU on FPGA.

CHAPTER 2
LITERATURE REVIEW

2.1 Microprocessor

A microprocessor or CPU executes a collection of machine instructions that tell the processor what to do. A microprocessor does a few basic things[1]:

1. Utilizing its ALU (Arithmetic/Logic Unit), a microprocessor can perform mathematical operations like addition, subtraction, multiplication and division.
2. A microprocessor can move data from one memory location to another and a microprocessor can make decisions and jump to a new set of instructions based on those decisions.

There may be very sophisticated things that a microprocessor does, but those are its three basic activities. A microprocessor comprises of registers as temporary storage area, buses to transfer data and select memory areas and control lines to control all the blocks inside the microprocessor so that the instruction are executed correctly.

2.2 Arithmetic Logic Unit

The ALU of the computer's CPU is part of the execution unit[1]. Generally it performs a wide variety of mathematical and logical operations in two's complement. It gets data from processor registers to be processed before storing them into ALU output registers. The control unit controls the ALU by instructing the ALU on which operations to perform. Most ALUs can perform the following operations[1]:

- Aritmetic operations (addition, subtraction, sometimes multiplication etc.)
- Bitwise logic operations (AND, NOT, OR, XOR)
- Bit-shifting operations

More complex arithmetic operations are usually performed in software like division and floating point operation[1]. The inputs to the ALU are the data to be operated on (called operands) and a code from the control unit indicating which operation to perform. Its output is the result of the computation. The ALU also takes or generates as inputs or outputs a set of condition codes from or to a status register. These codes are used to indicate cases such as carry-in or carry-out, overflow, divide-by-zero, etc.

CHAPTER 3
METHODOLOGY

The target minicomputer that the ALU is to be implemented upon runs on 3 MHz, supports user and supervisor modes, address translation via hardware page table, 6 external interrupts and up to 8MB of memory. The data bus is 8 bits wide and internal CPU data paths are 16 bits. The ALU can operate on both 8 and 16 bits operations. The block diagram of the CPU is shown in Figure 1. The portion enclosed in dotted lines is the ALU of the CPU.

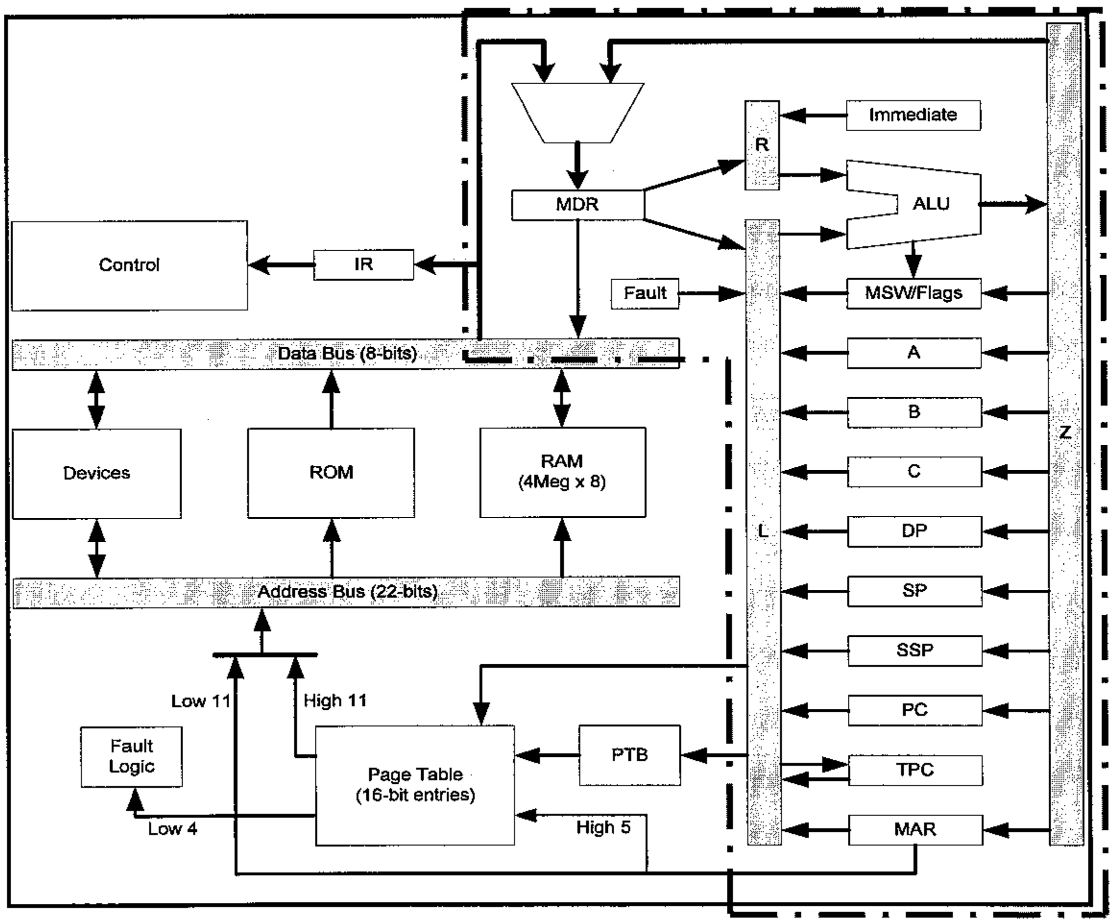
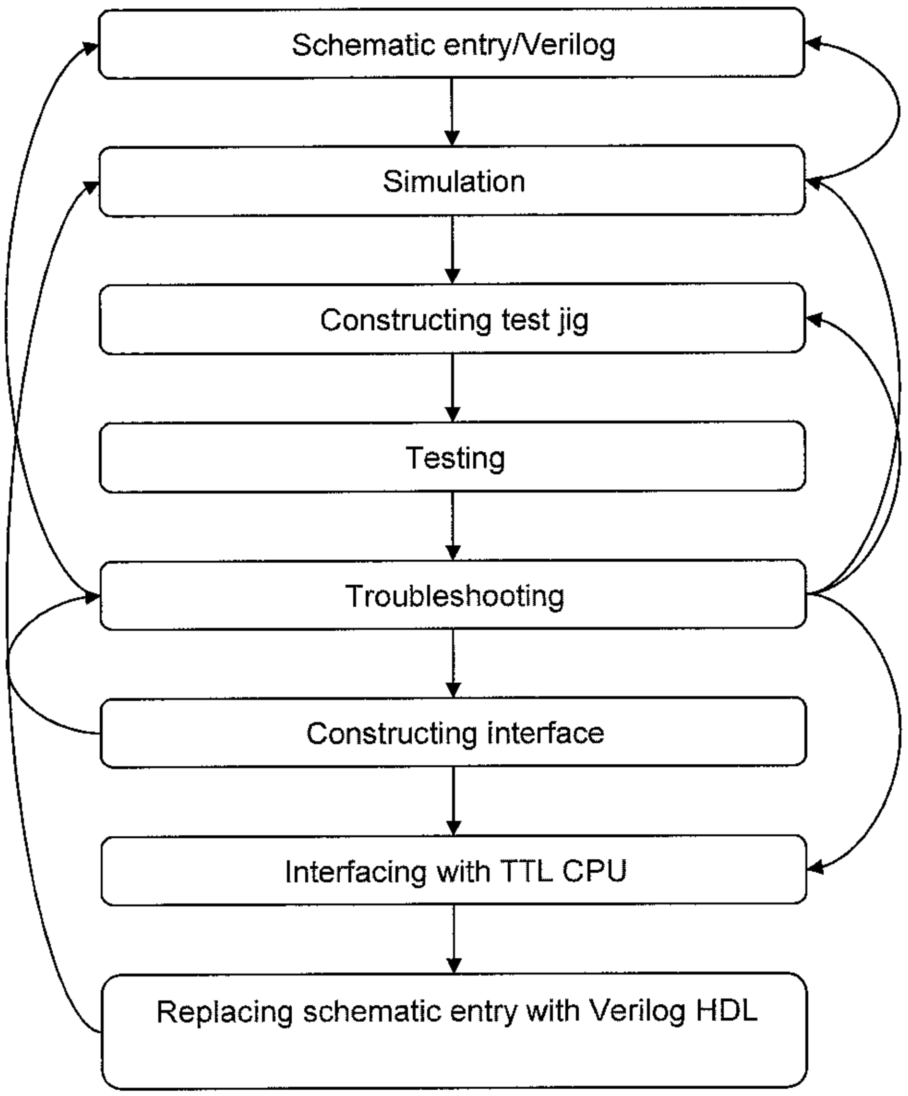


Figure 1: Block diagram of CPU



Firstly, the Quartus 2 software is installed and familiarization with the basic development environment begins. Schematic entry is being chosen over HDL programming as a start. Schematic entry was tried with simple circuits and gradually moving to the actual schematic that has to be built. The ALU subpart of the circuit was drawn first. Next, functional analysis is performed on it and errors are troubleshooted until the schematic is found to be working as expected.

The same is performed on the MSW, general registers, special registers, and MDR. Full functional analysis has also been performed and preliminary results show that it is functioning as expected. Preliminary timing analysis has also been carried out and no error was recorded. Next, the bidirectional buffer that cannot be drawn with schematic entry was programmed using verilog. It was then integrated into the schematics for hardware tests. Next, test rigs were built to interface with the UP2 board for hardware tests. Hardware tests have been carried out and have been verified to be working. Next, the interface with the TTL CPU is constructed. Interfacing is then performed. Then each TTL chip was replaced in verilog to gain familiarity with the language. The control card was also coded.

CHAPTER 4
RESULTS AND DISCUSSION

4.1 Arithmetic and Logic Operations (ALO)

The whole schematic that performs the arithmetic and logic operations was drawn and functional analysis was performed on it as a whole hopeful that the results could be satisfactory during the preliminary simulation. Preliminary results have shown that the waveform simulated for the Z bus is far from the theoretical value that was calculated for the test values that are fired at the inputs. As the source(s) of error is difficult to trace with so many connections, subsection of the ALO are being troubleshooted separately as standalones. The selector was taken out for troubleshooting first and it was found to work as expected. The selector is shown in Figure 2.

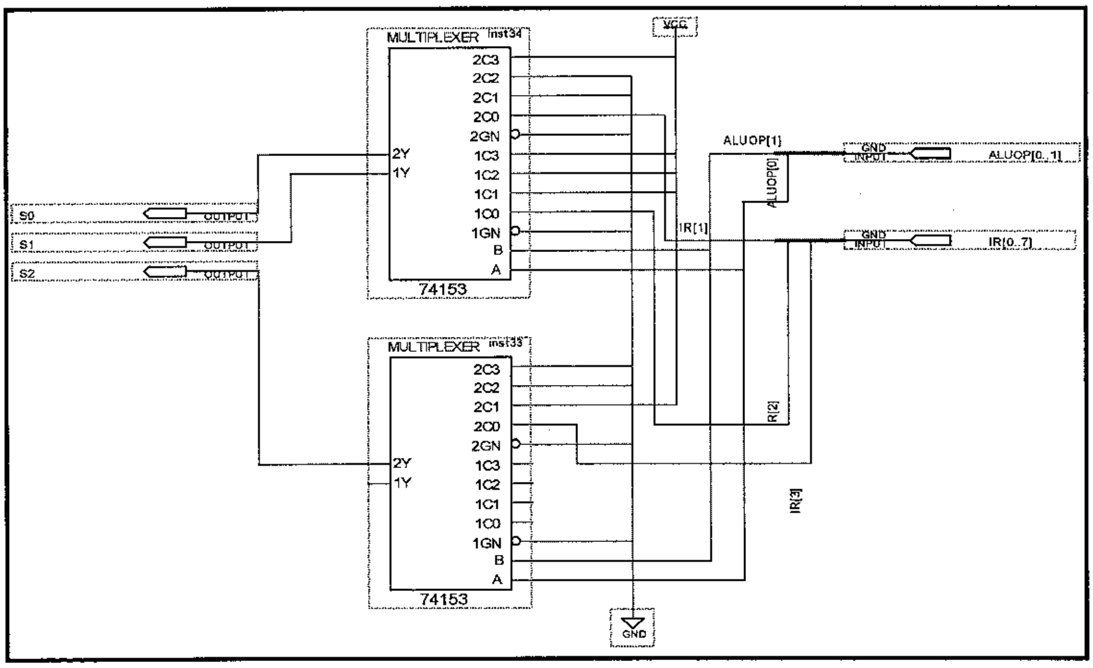


Figure 2: Schematic of selector

The expected results from the selector are shown in Table 1.

Table 1: Truth table of selector

ALUOP1	ALUOP0	S0	S1	S2	operation
A	B				
0	0		IR		IR
0	1	0	1	1	AB
1	0	0	1	0	A minus B
1	1	1	1	0	A plus B

Two waveform results of the selector are shown in Figure 3 and Figure 4. The lines that are bold indicate a HIGH logic level.

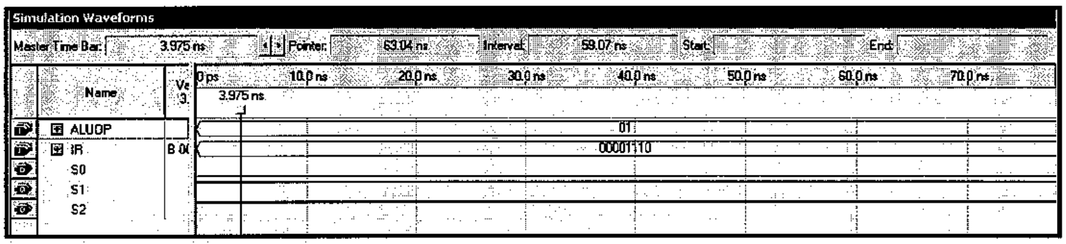


Figure 3: Output waveform of selector for ALUOP(01)

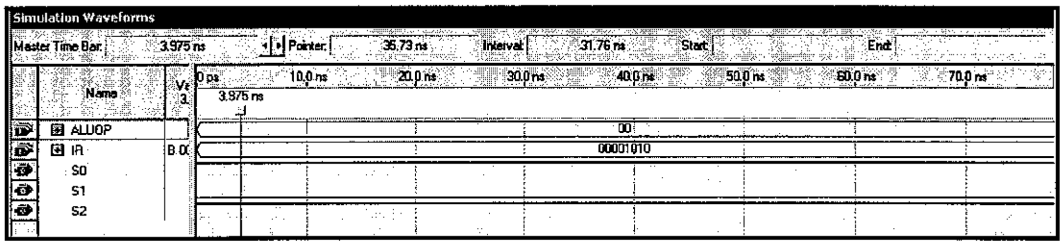


Figure 4: Output waveform of selector for ALUOP(00)

ALUOP (00) will give IR operation. This means that S0, S1, and S2 will take the values from IR1, IR2, and IR3. Values of IR0, and IR4 – IR7 will not play any role in this portion of the circuit. Thus referring to Figure 4, the results shown is as expected. All possible input combinations have been simulated to work as the truth table shown in Table 1.

Next, the 74382 and 74381 ALU parts are created as standalone to be performed functional analysis. The coverage of the standalone is shown in Figure 5.

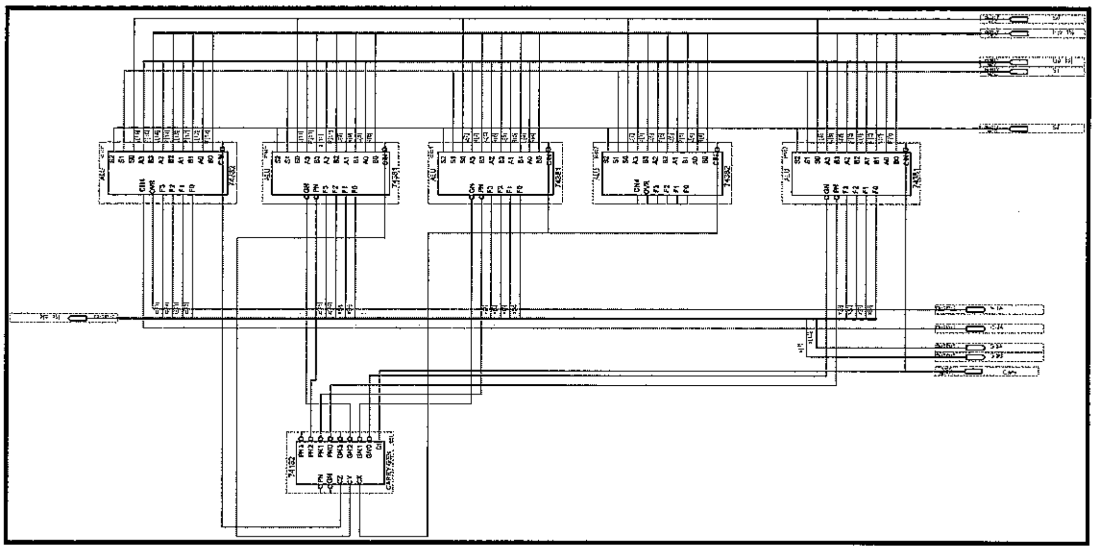


Figure 5: Schematic of ALU chips standalone

Several logic and arithmetic operations were simulated with the ALU chips standalone. Addition, AND and OR operation worked as expected. Test values for A and B and their expected theoretical outputs are shown in Table 2. The truth table for the resulting arithmetic and logic operation based on the inputs are shown in Table 3. The last bit of the minus operation was found to be faulty. After examining the datasheets again, it was discovered that the carry in input has to be force high for active high operation. The simulated waveform results for the mentioned logic and arithmetic operations are shown in Figure 6, 7, 8 and 9.

Table 2: Test values and theoretical results for ALU standalone

	1	1						1	1	1						operation
0	0	1	1	1	0	0	0	0	0	0	1	1	0	0	1	A
0	0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	B
0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	1	add
0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	and
0	0	1	1	1	0	0	0	0	1	1	1	1	0	0	1	or
0	0	1	0	0	1	1	1	1	0	1	0	1	0	0	1	minus

Table 3: Truth table of ALU standalone

S0	S1	S2	Operation
0	0	0	clear
1	0	0	B minus A
0	1	0	A minus B
1	1	0	A plus B
1	0	1	OR
0	1	1	AND

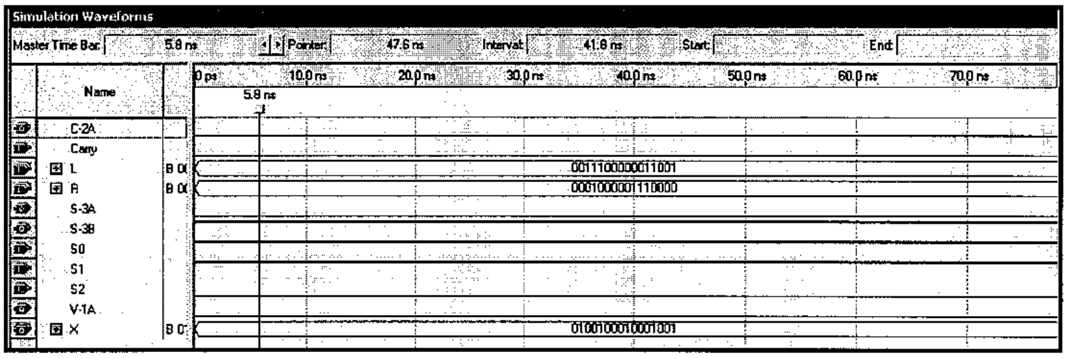


Figure 6: Simulation waveform for addition of test values

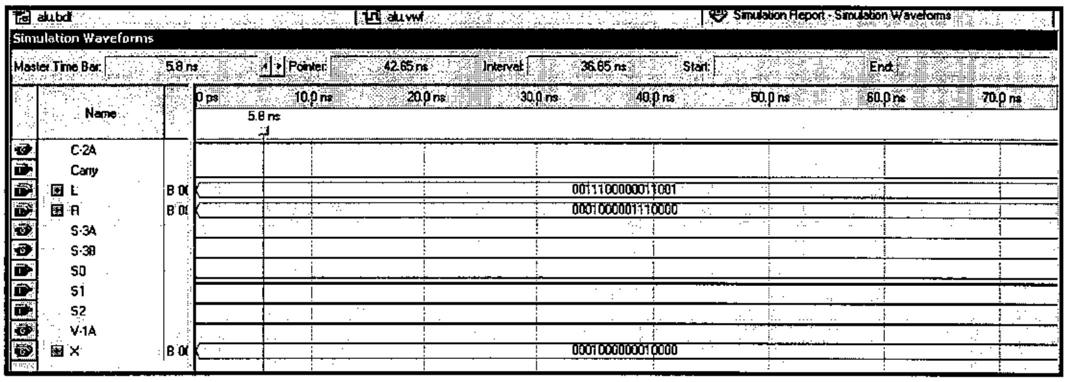


Figure 7: Simulation waveform for AND operation of test values

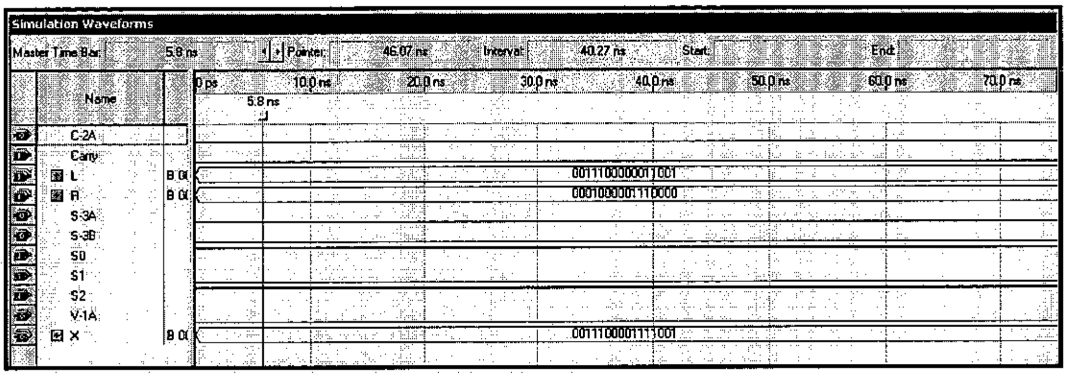


Figure 8: Simulation waveform for OR operation of test values

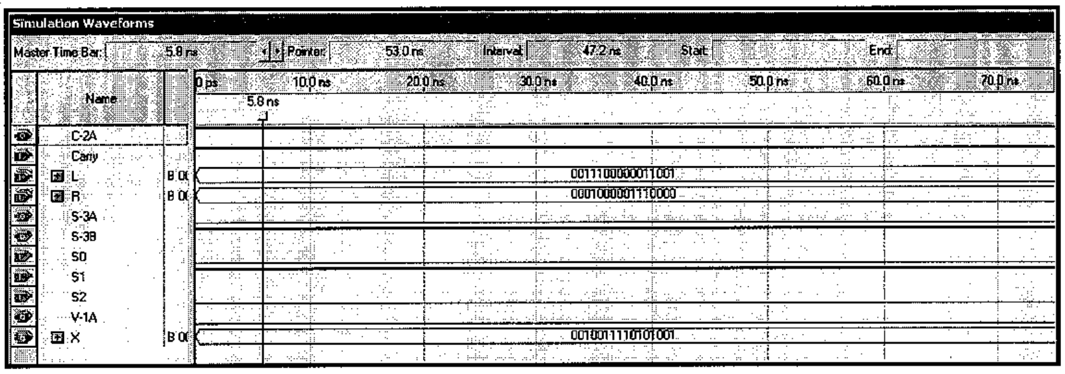


Figure 9: Simulation waveform for minus operation of test values

Since the ALU standalone chips are functioning as expected, the next step is to implement the buffers in the ALU. The section of the circuit is shown in Figure 10.

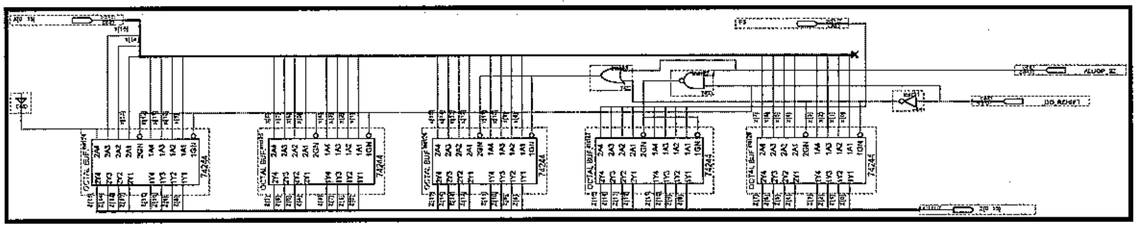
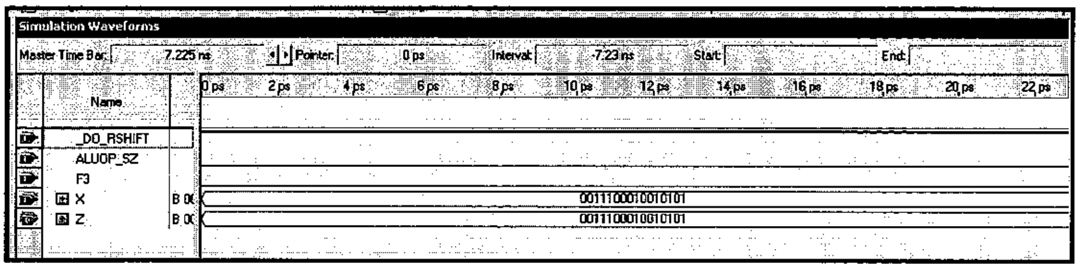


Figure 10: Buffers in the ALU

The initial simulation results indicate that the bits are shifted to the right. It was suspected then the cause of this could be due to the input from _DO_RSHIFT. Therefore, _DO_RSHIFT is forced low (initially it was force high) and the simulation results become accurate. This verifies that _DO_RSHIFT is an active low input. The simulation output waveform is shown in Figure 11.



```
_DO_RSHIFT
ALUOP_SZ
F3
X    B 0(   0011100010010101
Z    B 0(   0011100010010101
```

Figure 11: Simulation waveform of ALU buffers

After these few changes are made to the overall ALO circuit. Simulation was carried out again and it functioned as expected except the USE_CARRY input is an invert from the 'carry in' of the ALU chips standalone. This is because there were intermediate logic components being used in the overall circuit. That means that USE_CARRY is active low while 'carry in' is active high. The whole ALO schematic is shown in Figure 12. For the full waveform simulation, the truth table in Table 1 is still valid. The test values used for L and R bus are still the same as Table 2. The L bus will be fed to the A inputs and the R bus will be fed to the B inputs.

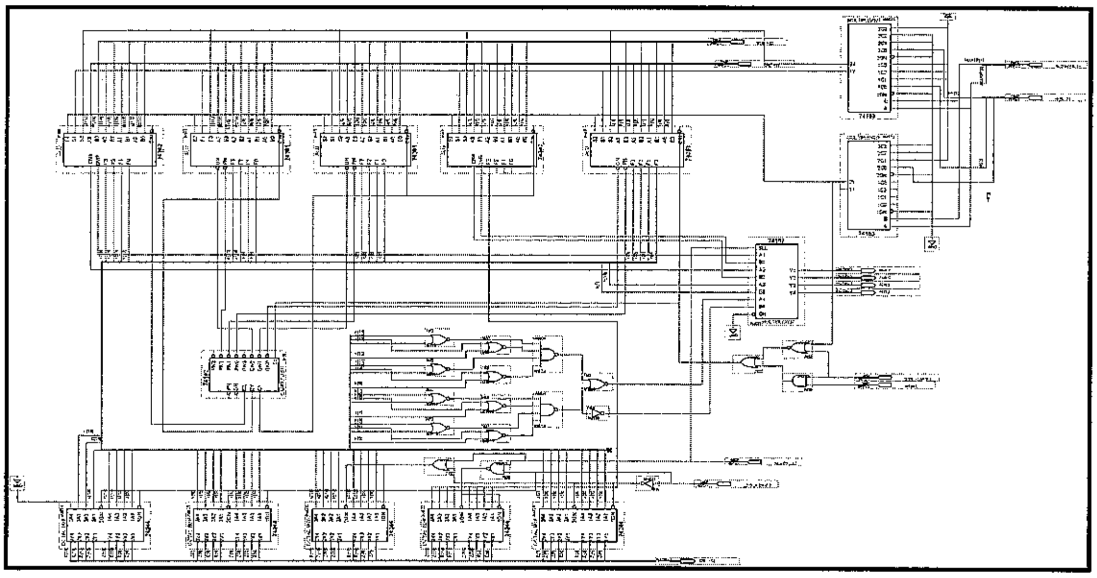


Figure 12: ALO schematic as a whole

The output waveforms for several logic and arithmetic operations are shown in Figure 13, 14, and 15. The shift right input will shift the output waveform one bit to the right. We used the result waveform of AND operation (Figure 13) and shift it to the right by one bit. The simulation waveform is shown in Figure 16.

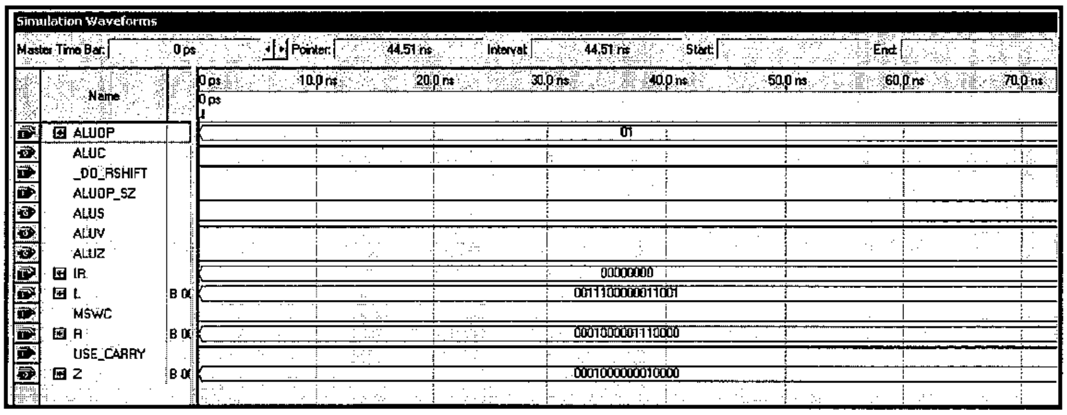


Figure 13: Simulation waveform for AND operation of test values of whole ALO

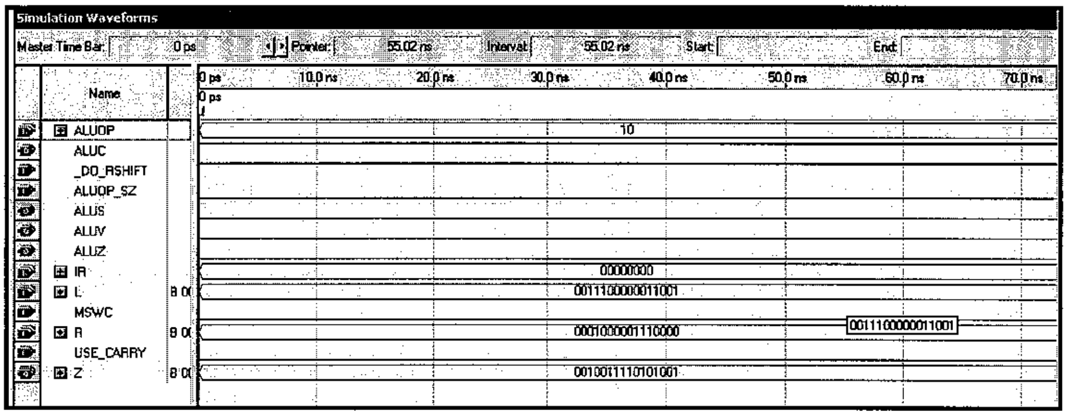


Figure 14: Simulation waveform for minus operation of test values of whole ALO

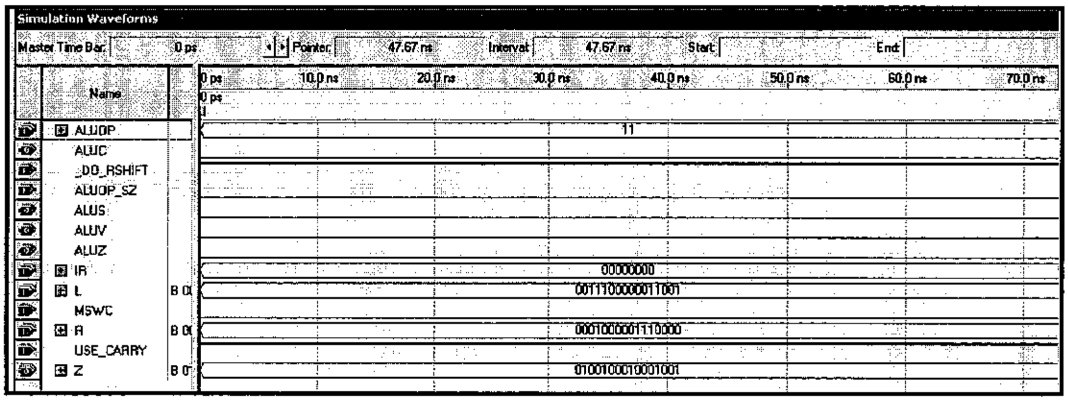


Figure 15: Simulation waveform for addition operation of test values of whole ALO

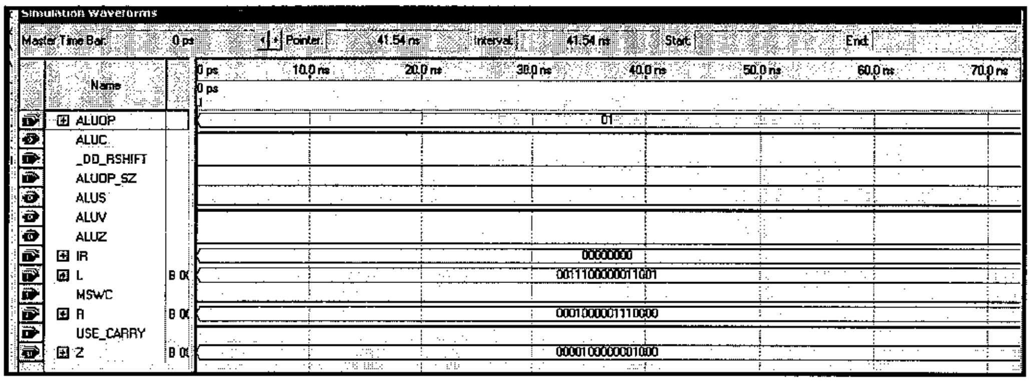


Figure 16: Simulation waveform for AND operation shifted one bit to the right of test values of whole ALO

4.2 MSW

Next, the schematic of MSW is being drawn in Quartus 2 (Figure 17). The MSW deals mainly with the ALU flags (carry, zero, sign and overflow), control flags (Mode for supervisor of user), paging and interrupts enable.

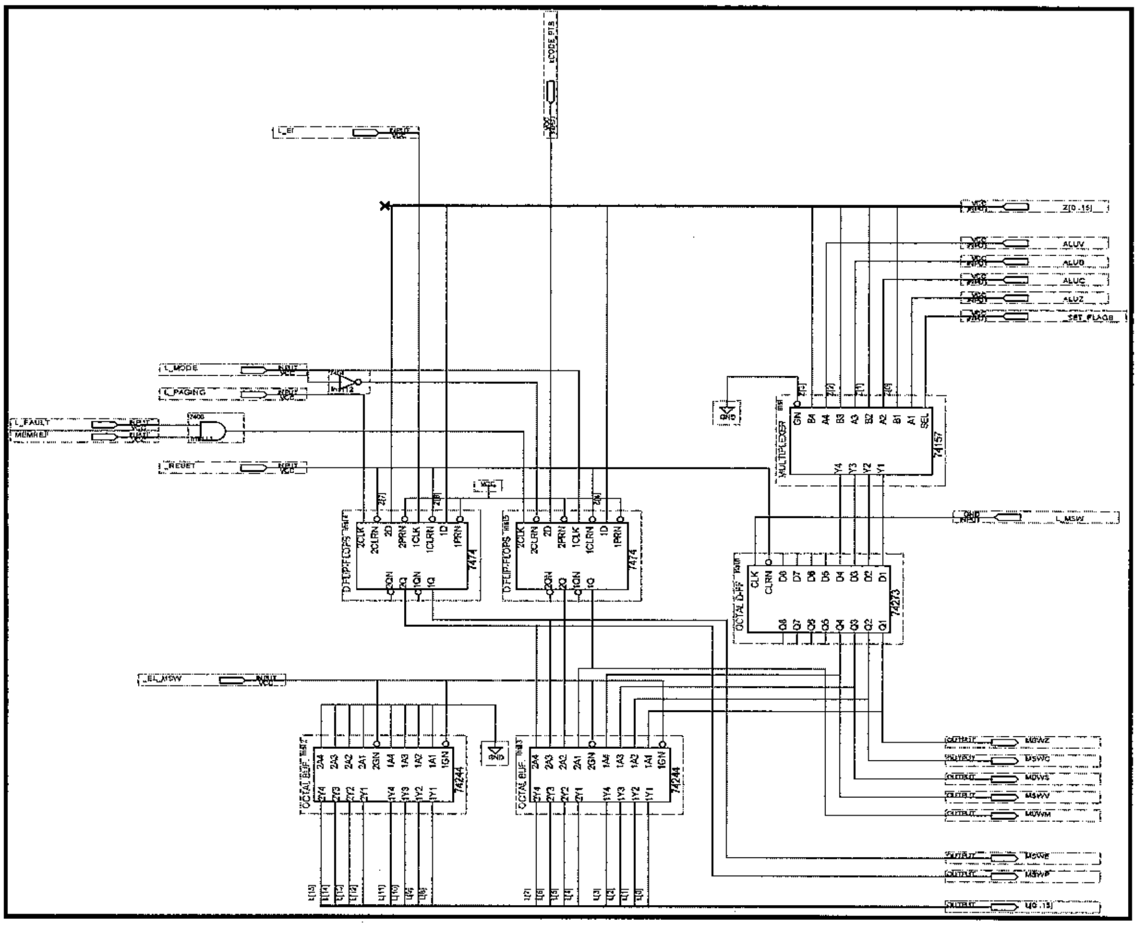


Figure 17: MSW schematic

All possible combinations of input have been tested to work accurately. One output waveform has been shown in Figure 18.

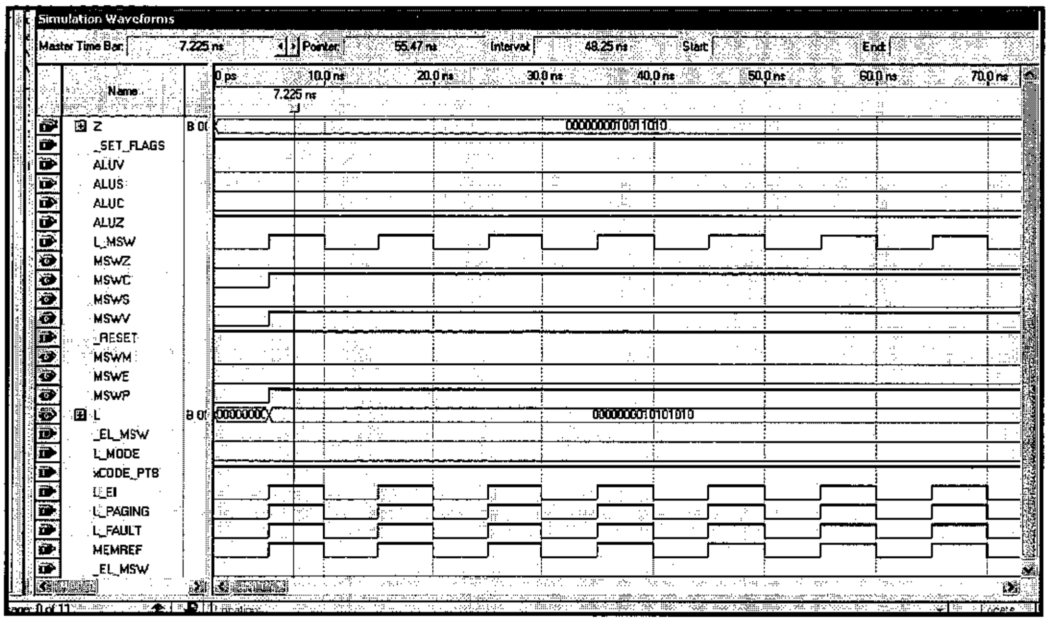


Figure 18: Simulation waveform of MSW

4.3 General Registers

Schematic of general registers have also been drawn. Refer to Appendix A.

The functional analysis of the general registers is much more straightforward than for ALU and MSW. It is generally just choosing which entity to drive the L bus at any particular time. One simulation output waveform is shown in Figure 19.

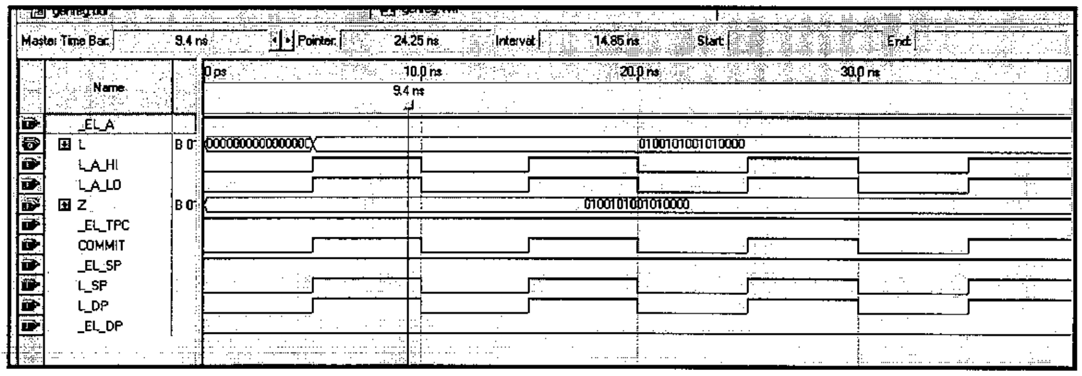


Figure 19: Simulation waveform of general registers

4.4 Special Registers

The special registers schematic is shown in Figure 20. It is also a rather simple schematic and the output waveform is shown in Figure 21.

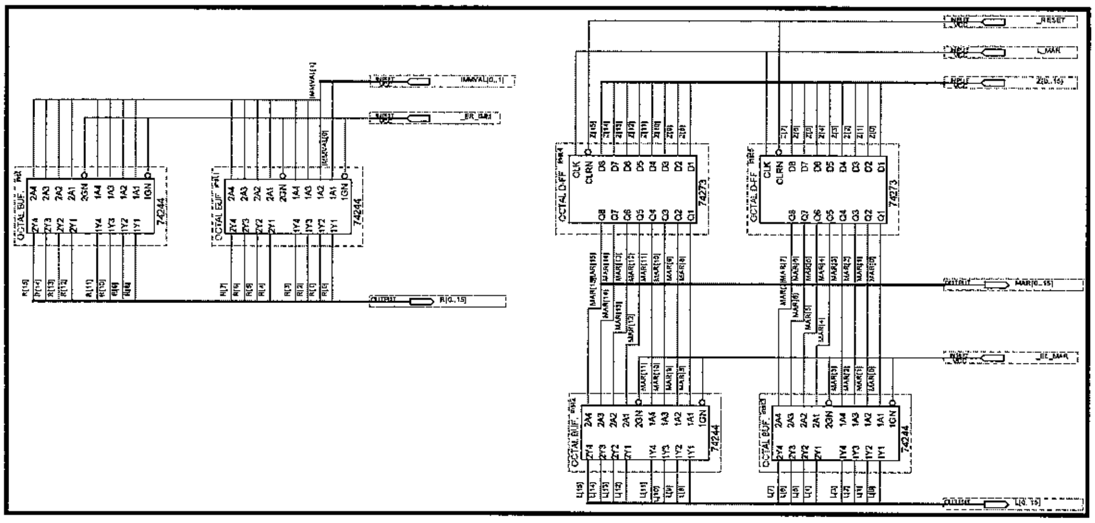


Figure 20: Schematic of special registers

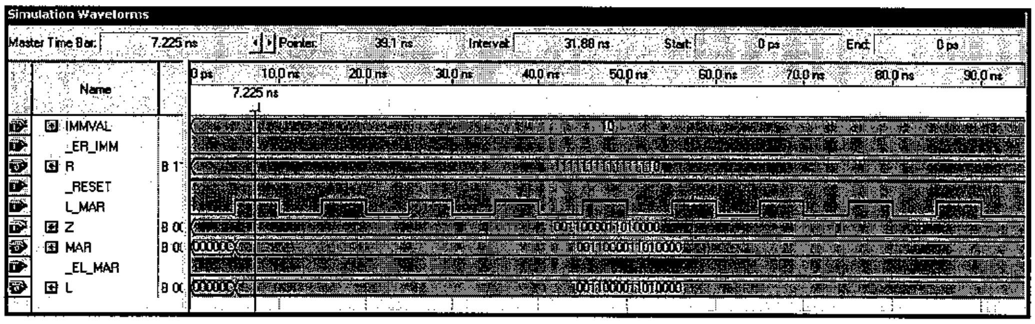


Figure 21: Simulation waveform of special registers

4.5 MDR

MDR The schematic for MDR has been shown in Figure 22. There is a missing component in the schematic which is the 74F245 which is an octal bidirectional transceiver with 3-state outputs. In the original design, this chip plays the role of a bidirectional buffer between the data bus and the MDR. The IC is not available in the schematic entry library. Therefore, the component will be included in the design in HDL.

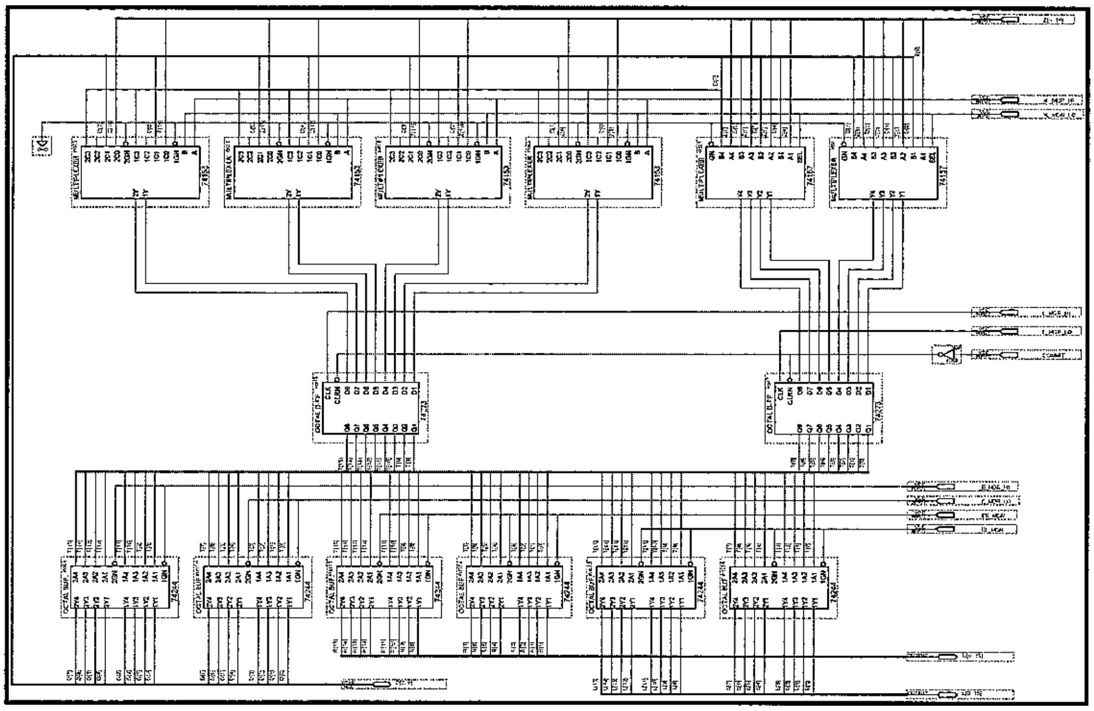


Figure 22: Schematic of MDR

The 16 bits that will be loaded into L, R and data busses will depend on the input of XL_MDR_HI and XL_MDR_LO whose combination will select input from either the 8 bits data bus (D bus) or 16 bits Z bus or the combination of both (see Table 4).

Table 4: Selection of input to MDR

XL_MDR_HI	XL_MDR_LO	Higher 8 bits	Lower 8 bits
0	0	Z bus	Z bus
0	1	-	D bus
1	0	D bus	Z bus
1	1	D7	D bus

The simulation output waveform is shown in Figure 23. When bidirectional busses are being used, input must not be driven strong high or low because if output to the busses contradict with the input values error will be produced. Therefore to avoid producing any errors, weak low or high is being used for input. Weak low is indicated by L and weak high, H whereas strong high is 1 and strong low 0 (see Figure 23)

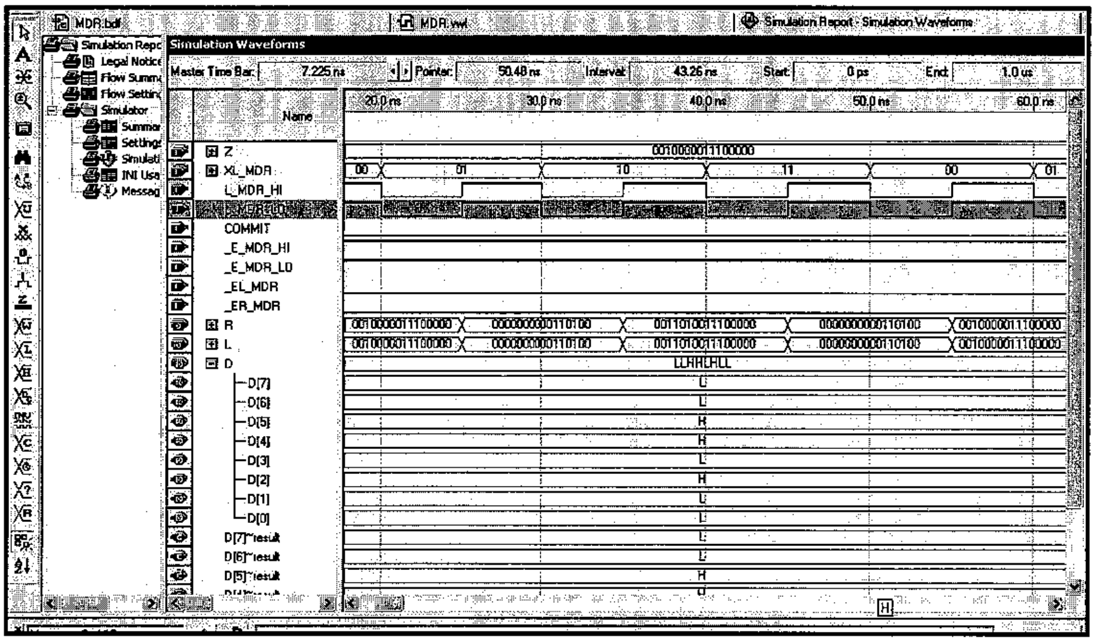


Figure 23: Simulation waveform of MDR

4.5.1 Bidirectional Buffer in Verilog

As the components library in Quartus 2 does not contain the 74245 bidirectional buffer. It has to be written in verilog. The code written is appended in Appendix B.

4.6 Integrated ALU

The separate ALU schematics are combined into an integrated schematic to carry out functional analysis. The integrated schematic is appended in Appendix A.

Preliminary results were unsuccessful and many errors were listed. These errors have been corrected and among them are:

1. The logic contention errors occurred because input and output buses are of a different logic value.
2. During integration some non-existent connections have been established and this has been corrected.
3. The registers to any busses can only be enabled one at a time.
4. L, R, Z and D busses have to be set as bidirectional.
5. Output cannot be obtained immediately. Each process has to be run separately. In the real operation of a microprocessor, a microcode will control

the operation of the ALU. Since the microcode is not available to simulate it, simulation of each processes are done manually. For example, data are loaded in the D bus first and buffered into L and R busses which will be fed to the Z bus for ALU operation. During the next process, the previous buffers have to be disabled and current related registers have to be enabled. After ALU operation, the output in Z bus have to be loaded in L bus which again required the previous bus to be disabled. As the operation is a feedback loop, the output waveform is very complicated.

6. Logic level must be specified for bidirectional node even if the tri-state that feeds the bidirectional node is always enabled. A constant logic level of high impedance (Z), weak low (L), or weak high (H) for the channel can be specified. If a logic level is not specified, the simulator will assume the logic level is forcing unknown (X), that typically produces unexpected results.

7. If a tri-state buffer that feeds a bidirectional node is enabled, the logic level of the bidirectional node must be high impedance (Z) or a weak signal—for example, weak low (L), weak high (H), or weak unknown (W). If the bidirectional node does not have the correct logic level, the Simulator could produce an error if there is logic contention. For example, if the logic level at the output of the tri-state buffer is different— strong high (1)—from the logic level that the bidirectional channel drives in—strong low (0), the Simulator produces an error.

8. If the tri-state buffer that feeds a bidirectional node is disabled, the logic level of the bidirectional node must not be high impedance (Z), because the Simulator propagates a forced unknown (X) logic level. If the logic level of the bidirectional node is any of the weak signals, for example, weak low (L), weak high (H), or weak unknown (W), the Simulator uses them as strong signals once the signals propagate through the device.

9. The input channel cannot be written to by the simulator. Thus, if an output node named "a" and an input channel named "a" exist in the waveform, simulation will fail.

The output waveform is shown in Figure 24.

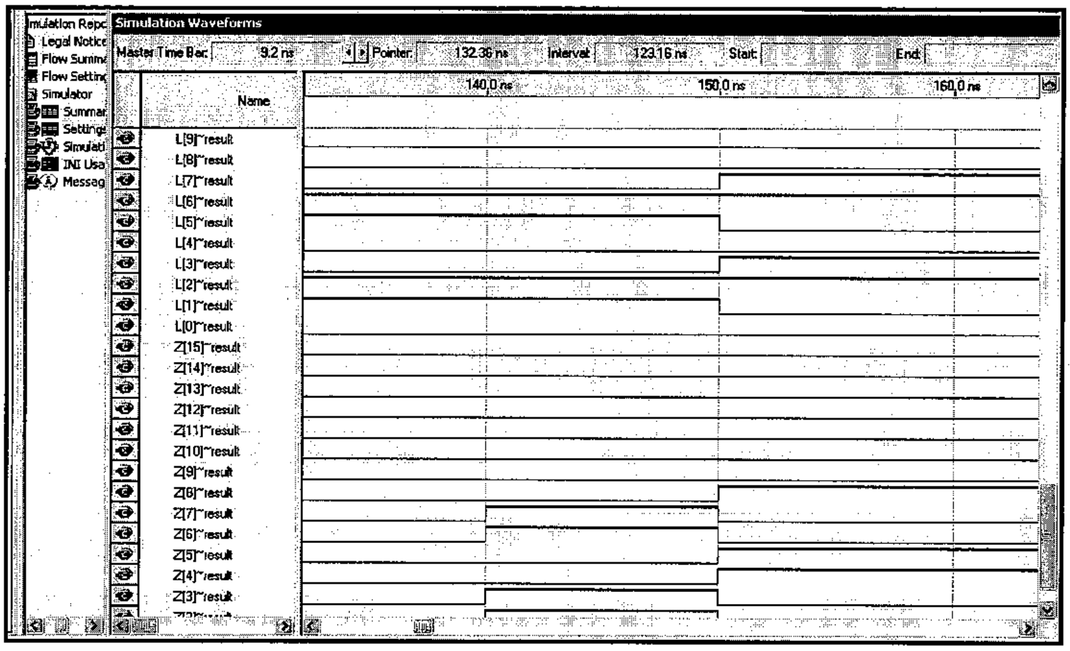
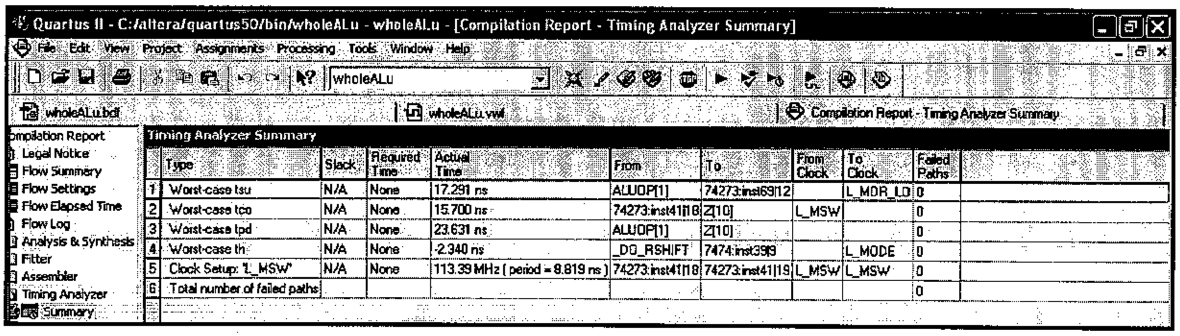


Figure 24: Output waveform of integrated ALU

4.7 Timing Analysis

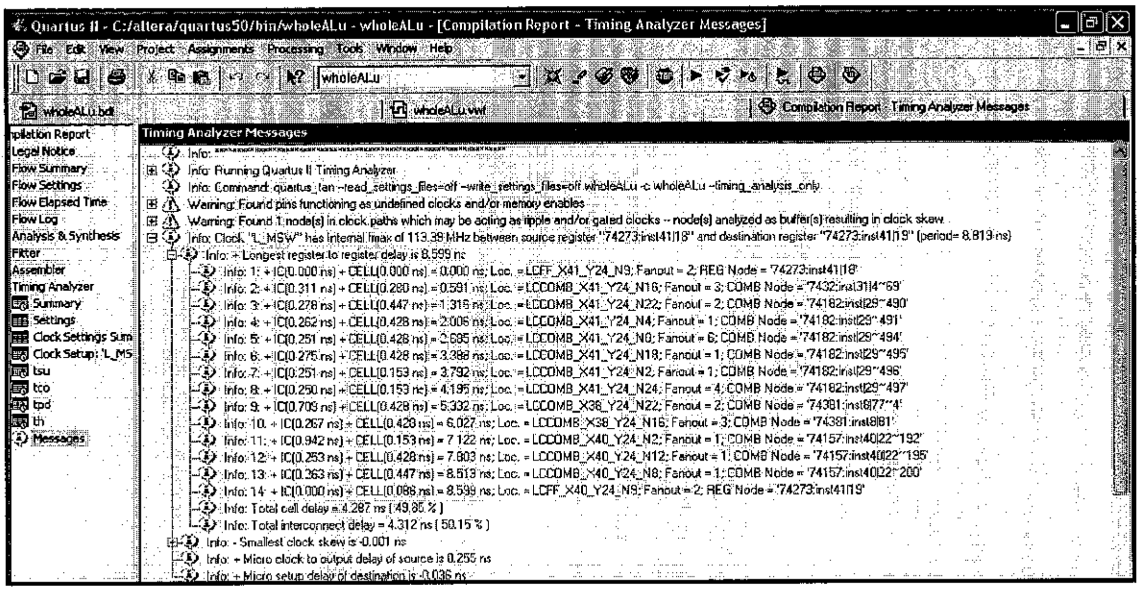
Timing analysis was performed on the integrated ALU schematic. The result of the analysis is shown in Figure 25 and Figure 26. No error was indicated during the timing analysis. The compilation was successful.


Quartus II - C:/altera/quartus50/bin/wholeALu - wholeALu - [Compilation Report - Timing Analyzer Summary]

Timing Analyzer Summary

	Type	Slack	Required Time	Actual Time	From	To	From Clock	To Clock	Failed Paths
1	Worst-case tsu	N/A	None	17.291 ns	ALUOP[1]	74273:inst69|12		L_MDR_LD	0
2	Worst-case tco	N/A	None	15.700 ns	74273:inst41|18	Z[10]	L_MSW		0
3	Worst-case tpd	N/A	None	23.631 ns	ALUOP[1]	Z[10]			0
4	Worst-case th	N/A	None	-2.340 ns	_DO_RSHIFT	7474:inst39|9		L_MODE	0
5	Clock Setup: 'L_MSW'	N/A	None	113.39 MHz (period = 8.819 ns)	74273:inst41|18	74273:inst41|19	L_MSW	L_MSW	0
6	Total number of failed paths								0

Figure 25: Timing analyzer summary



```
Info: ****************************************************
Info: Running Quartus II Timing Analyzer
Info: Command: quartus_tan --read_settings_files=off --write_settings_files=off wholeALu -c wholeALu --timing_analysis_only
Warning: Found pins functioning as undefined clocks and/or memory enables
Warning: Found 1 node(s) in clock paths which may be acting as ripple and/or gated clocks -- node(s) analyzed as buffer(s) resulting in clock skew
Info: Clock "L_MSW" has Internal fmax of 113.39 MHz between source register "74273:inst41|18" and destination register "74273:inst41|19" (period= 8.819 ns)
  Info: + Longest register to register delay is 8.599 ns
    Info: 1: + IC(0.000 ns) + CELL(0.000 ns) = 0.000 ns; Loc. = LCFF_X41_Y24_N9; Fanout = 2; REG Node = '74273:inst41|18'
    Info: 2: + IC(0.311 ns) + CELL(0.280 ns) = 0.591 ns; Loc. = LCCOMB_X41_Y24_N16; Fanout = 3; COMB Node = '7432:inst31|4~69'
    Info: 3: + IC(0.278 ns) + CELL(0.447 ns) = 1.316 ns; Loc. = LCCOMB_X41_Y24_N22; Fanout = 2; COMB Node = '74182:inst|29~490'
    Info: 4: + IC(0.262 ns) + CELL(0.428 ns) = 2.006 ns; Loc. = LCCOMB_X41_Y24_N4; Fanout = 1; COMB Node = '74182:inst|29~491'
    Info: 5: + IC(0.251 ns) + CELL(0.428 ns) = 2.685 ns; Loc. = LCCOMB_X41_Y24_N0; Fanout = 6; COMB Node = '74182:inst|29~494'
    Info: 6: + IC(0.275 ns) + CELL(0.428 ns) = 3.388 ns; Loc. = LCCOMB_X41_Y24_N18; Fanout = 1; COMB Node = '74182:inst|29~495'
    Info: 7: + IC(0.251 ns) + CELL(0.153 ns) = 3.792 ns; Loc. = LCCOMB_X41_Y24_N2; Fanout = 1; COMB Node = '74182:inst|29~496'
    Info: 8: + IC(0.250 ns) + CELL(0.153 ns) = 4.195 ns; Loc. = LCCOMB_X41_Y24_N24; Fanout = 4; COMB Node = '74182:inst|29~497'
    Info: 9: + IC(0.709 ns) + CELL(0.428 ns) = 5.332 ns; Loc. = LCCOMB_X38_Y24_N22; Fanout = 2; COMB Node = '74381:inst8|77~4'
    Info: 10: + IC(0.267 ns) + CELL(0.428 ns) = 6.027 ns; Loc. = LCCOMB_X38_Y24_N16; Fanout = 3; COMB Node = '74381:inst8|81'
    Info: 11: + IC(0.942 ns) + CELL(0.153 ns) = 7.122 ns; Loc. = LCCOMB_X40_Y24_N2; Fanout = 1; COMB Node = '74157:inst40|22~192'
    Info: 12: + IC(0.253 ns) + CELL(0.428 ns) = 7.803 ns; Loc. = LCCOMB_X40_Y24_N12; Fanout = 1; COMB Node = '74157:inst40|22~195'
    Info: 13: + IC(0.263 ns) + CELL(0.447 ns) = 8.513 ns; Loc. = LCCOMB_X40_Y24_N8; Fanout = 1; COMB Node = '74157:inst40|22~200'
    Info: 14: + IC(0.000 ns) + CELL(0.086 ns) = 8.599 ns; Loc. = LCFF_X40_Y24_N9; Fanout = 2; REG Node = '74273:inst41|19'
    Info: Total cell delay = 4.287 ns ( 49.85 % )
    Info: Total interconnect delay = 4.312 ns ( 50.15 % )
  Info: - Smallest clock skew is -0.001 ns
  Info: + Micro clock to output delay of source is 0.255 ns
  Info: + Micro setup delay of destination is -0.036 ns
```

Figure 26: Timing analyzer messages

4.8 Test rig

Test rigs are built to ensure that the hardware results are coherent and functionally working as the software simulation results. The conceptualized test rig is to interface the expansion slots of the UP2 board (Fig 28) with external circuits of purely LEDs and switches as inputs and outputs. The switches will be used to trigger input pins and LEDs to display output pins logic level from the UP2 board. The 3 three expansion slots can support 60 pins each. However, 60 pins interfacing components were not found and the more common 40 pins interface was constructed instead. Three test circuits were constructed (Figure27) for each of the three expansion slots.

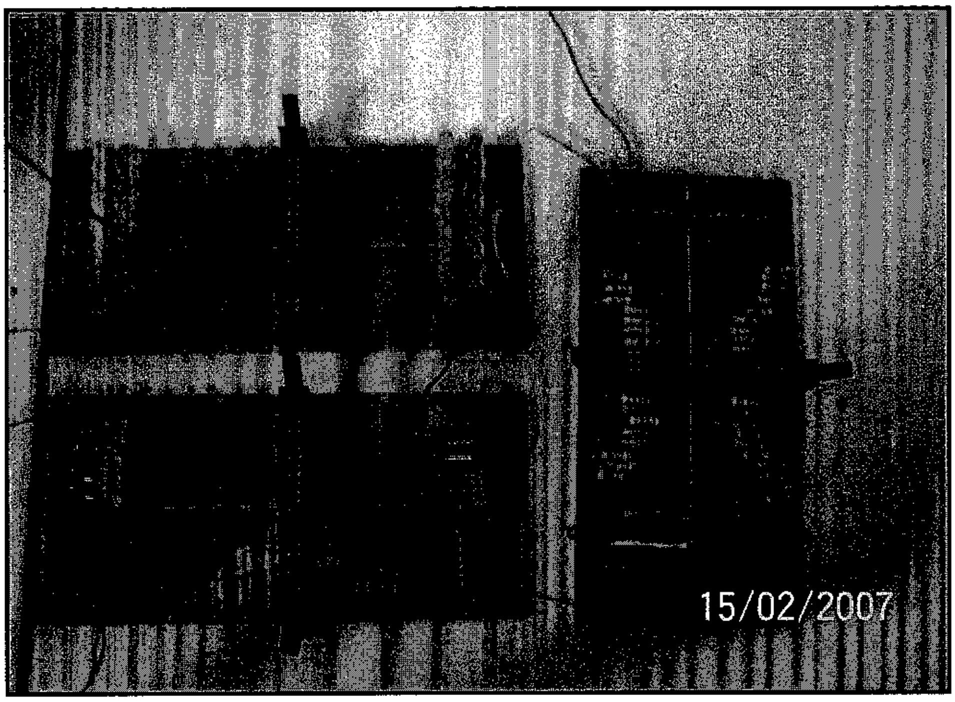


Figure 27: Test rigs

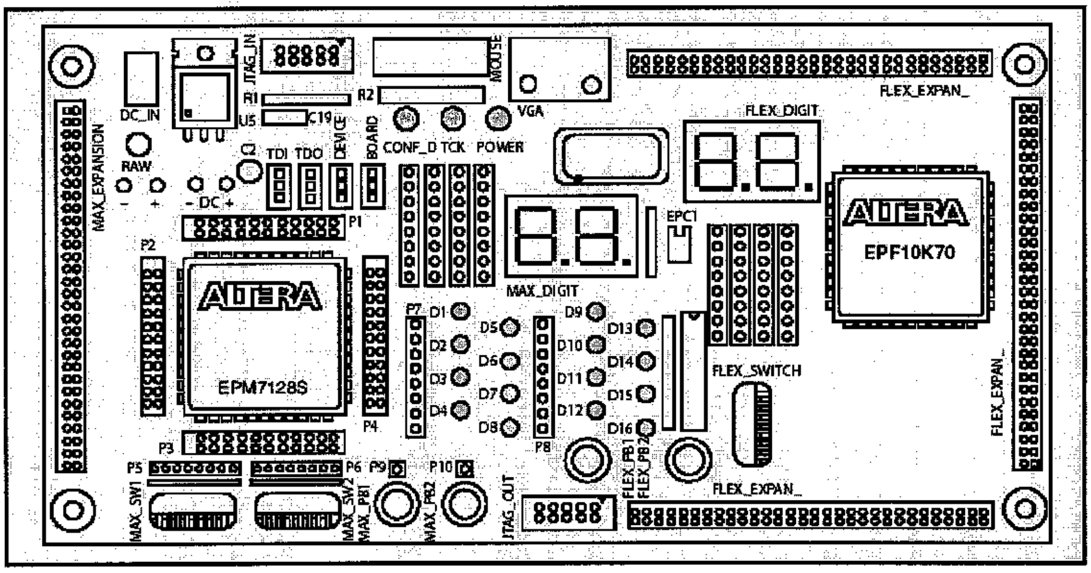


Figure 28: UP2 board

IDE cables were made to interface the flex expansion slots with the three test rigs.

4.9 MDR on test rig

The MDR module is programmed for hardware tests. Results were observed on L bus and seen through lighted LEDs (see Appendix C). The MDR module is fully functional on the test rig

4.10 ALU on test rig

The ALU module is also tested in the test rig and was verified to be functional. A picture is appended in Appendix C.

4.11 Integrated ALU on test rig

Pin assignments are made before final recompilation. The software file is programmed into the UP2 board to be interfaced with the test rig. Each switch is set to the default position of either logic 0 or 1. The test procedures that were used are outlined below:

1. Register T1 and T2 in MDR are cleared (COMMIT = positive pulse)
2. Immediate value is asserted at DBUS (01100110)
3. Two-way buffer direction is selected as B to A (_RW = low)
4. The buffer is then enabled, immediate data on D bus (_DMA_ACK = high)
5. MUX 2 is set to flow D into register T2 (XL_MDR_LO = high)
6. Load register T2 with immediate data (L_MDR_LO = positive pulse)
7. MUX 1 is set to flow D into register T1 (XL_MDR_LO = low, XL_MDR_HI = high)
8. Load register T1 with immediate data (L_MDR_HI = positive pulse)
9. Buffer 1 and buffer 2 are set to assert both bus R and L with the same content of bus T as right and left operand into the ALU – 0110011001100110 (_ER_MDR = low, _EL_MDR = low), content of bus L which is already connected to LEDs can be viewed
10. ALU operation is set to ADD (ALUOP0 = high, ALUOP1 = high)
11. Use of carry is prohibited (USE_CARRY = low)
12. ALU operation size selected as 16 bits (ALUOP_SZ = low)

13. Result is not shifted right by one bit (_DO_RSHIFT = high), result of the bitwise addition of the same operands with a carry in is now on the Z bus)
14. Result is then stored into one of the registers, selectively register C (L_C = positive pulse, clock in)
15. To read the content of register C, first disable the buffering of operand into bus L by buffer2 (_EL_MDR = high)
16. Read the content of register C through bus L (_EL_C = low), result of the addition can now be viewed through the LEDs
17. Reading the flags (_SET_FLAGS = low, L_MSW = positive pulse)
18. ALU operation now replaced with AND and minus in step 10 and repeat steps 11-17
19. Results are right shifted in step 13 and steps 14 – 17 are repeated
20. 8 bit operations is then chosen (ALUOP_SZ = high) in step 12 and steps 13 – 17 are repeated
21. Carry-in in step 11 is set to high (USE_CARRY = high) and steps 12 – 17 are repeated
22. General register used in step 14 is tested one at a time with 6 other general registers (A, B, DP, SP, SSP, PC). Step 15 – 17 is repeated with step 16 replaced by the enable of the respective register
23. _Set_flags set high in step 17 and flags are read
24. The paging enable, interrupts enable, mode (supervisor/user), and data checked are tested by varying the inputs of L_MODE, L_PAGING, L_FAULT, MEMREF, xCODE_PTB, and L_EI.
25. Special registers are also tested by varying IMMVAL and observe output in R bus
26. Memory address register are also checked at the MAR bus.

The test procedures covers a comprehensive test on all input and output pins.

4.11.1 Troubleshooting

Initially no output was observed in procedure 9. Not all pins are assigned on hardware as the interface cannot accommodate all pins at once. The initial assumption is that the choice of assigned pins could affect the test rig results. Software simulation that was performed successfully encompasses all the pins and they were set to the correct logic level. But the test rig has pin limitation constraints. Therefore, to ensure that the choice of pin will not affect its functionality, software simulation is performed with only the assigned pins. The choice of assigned pins is changed until the functional simulation is coherent. Once functional simulation is successful, timing simulation is performed but was unsuccessful. The values on the busses were of unknown logic level (Fig.29).

Length of applied enable signal was extended to longer than the worse setup time of the timing report. 100ns pulse length was used. R bus could now register logic levels but not L bus. Some signals are in 'X' state and Z bus is in 'U' state (see Figure 29).

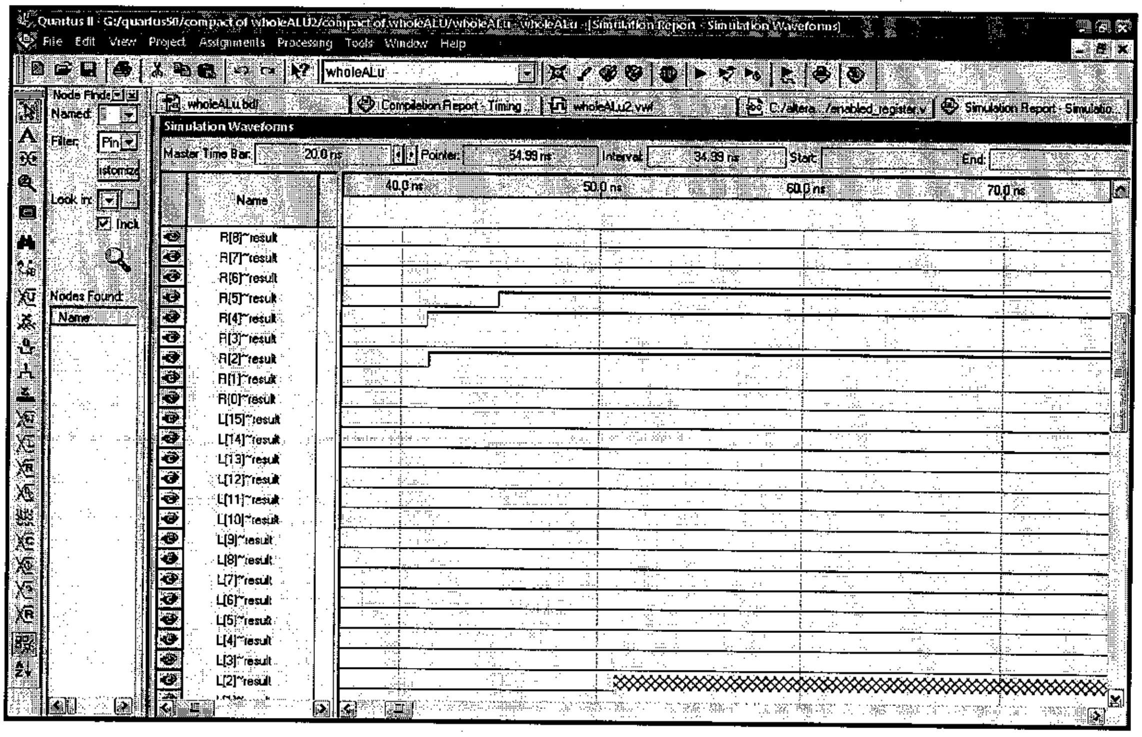
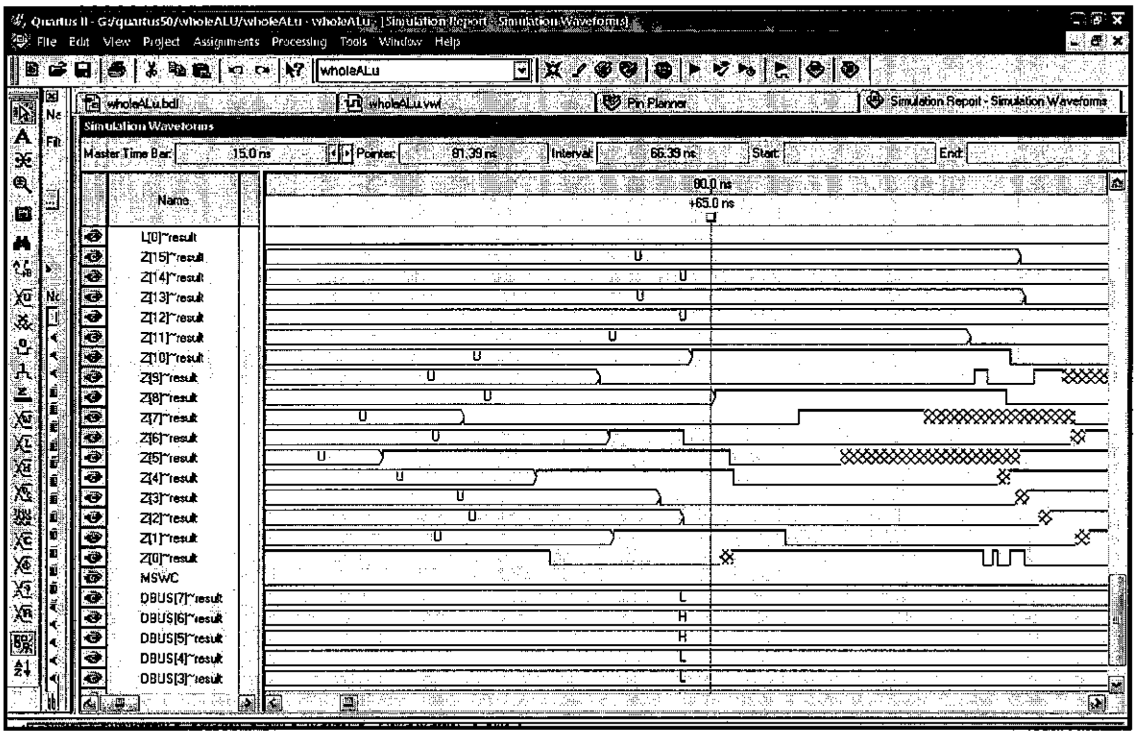


Figure 29: Waveform of unknown logic levels

Since R bus was successfully registered in the timing simulation, pins were assigned to hardware to test if R bus could be displayed with the LEDs in the test rig. However no results were obtained. It was later ascertained that the IR pins that were omitted in pin assignment resulted in Z bus contention with software simulation. This means that IR pins cannot be omitted from hardware pin assignment (see Figure 30).



Master Time Bar: 15.0 ns Pointer: 81.39 ns Interval: 66.39 ns Start: End:

Figure 30: Waveform of logic contention

Pins were rearranged pins in a more organized approach in the pin planner and tests were reconducted while ensuring all logic levels of the default condition in the test rig switches are accurate. No results were observed yet. Unassigned pins (pins not connected to the test rig) in pin planner were set as reserved pins. Compilation fails. This indicates that Quartus 2 automatically assign unassigned inputs/outputs to other pins not user-assigned. Therefore pins that are not interfaced but were automatically assigned as input/output pins by the program are left in a floating condition (or driven to the wrong logic state by switches thought to be unused) and could affect the test results and cause logic contention error.

The source of error could also be the test rig itself. There are possibilities that some switches have trouble registering the logic levels to the UP2 board even though it is at the correct logic level. Tests were conducted on each switch by constructing a register in schematics that output the logic level of each switch to a particular LED in the test rig when enabled. Some inputs/outputs did not display expected results. The cause of this could be the pins in the UP2 board are faulty, or the LEDs/switches in the test rig are faulty. Pin assignments were avoided at these designated pins. Finally, results were observed in the LEDs that represent the L bus (Fig 31) (test procedure 9). However when ALU operations are performed, the results observed in general registers are inaccurate (test procedure 16).

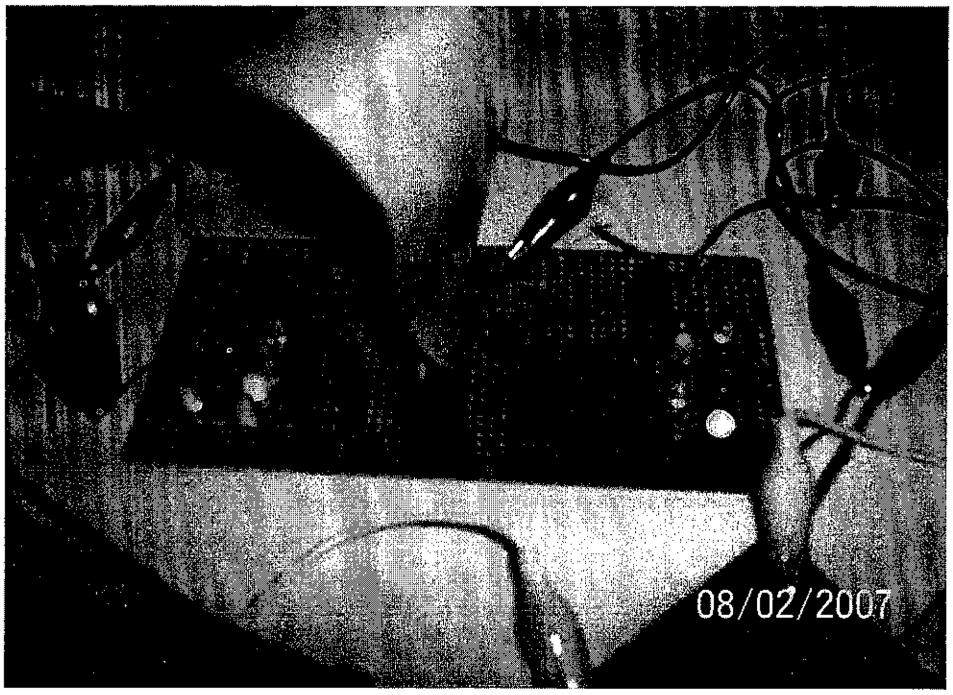


Figure 31: L bus values of MDR in integrated ALU

After numerous tests, it is observed that occasionally some input switches do not function as expected. This means, the right logic level could not be input to the UP2 input pins. However when it does, the test rig verified the functionality of the integrated ALU. Pictures are appended in Appendix C (Fig 42,45). The test results obtained are accurate and shown in Table 5. Changing ALU size to high (8 bits) turns the mode of operation to 8 bit and flags are read based on the 8 bits results. Setting Carry_in to high will result in an output higher by 1 bit for arithmetic operations. Setting _SET_FLAGS to high makes the flags output the upper four bits of Z bus. Other than results shown in the table, the registers in MSW that output the paging enable, interrupt enable and mode (user/supervisor) were also tested for functionality. The memory data registers was also tested to output the loaded values. The other general registers (A,B,PC,SSP,DP,PC,SP) were also used to replaced C register to ensure that they work as expected.

Table 5: Test results

Operation	ALUOP		Alu size	Carry in	Set flags	Output/Input																Flags			
																						Z	C	S	V
A						0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0				
B						0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0				
AND	0	1	0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	-	-	-
minus	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0
add	1	1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	1	1
shift			0	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0
AND	0	1	1	0	0	-	-	-	-	-	-	-	-	0	1	1	0	0	1	1	0	0	-	-	-
minus	1	0	1	0	0	-	-	-	-	-	-	-	-	0	0	0	0	0	0	0	0	1	1	0	0
add	1	1	1	0	0	-	-	-	-	-	-	-	-	1	1	0	0	1	1	0	0	0	0	1	1
shift			1	0	0	-	-	-	-	-	-	-	-	0	1	1	0	0	1	1	0	0	0	0	0
AND	0	1	0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	-	-	-
minus	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
add	1	1	0	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	1	0	0	1	0
shift			0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	0	0	0
AND	0	1	0	0	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
minus	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
add	1	1	0	0	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
shift			0	0	1	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0

IMMVAL		Output(R bus)															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Operation	IR			ALUOP		Output/Input																Flags			
																						Z	C	S	V
A						1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	0				
B						1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1				
AND	0	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	-	-	-
minus	0	0	0	1	0	0	0	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	1	0	0
add	0	0	0	1	1	0	0	1	0	1	0	1	1	0	1	1	1	1	1	1	1	0	0	1	0
OR	1	0	1	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	1	0	-	-	-
XOR	0	0	1	0	0	0	0	1	0	1	0	1	0	0	1	1	1	1	1	1	1	0	-	-	-

The inconsistencies of test results were further investigated. Sometimes, data values cannot effectively load into the respective busses. One possible deduction was the bounce-back effect of the mechanical switch used that especially when used as clock signal. This will cause a few clock signal to pass and might result in the wrong set of data latched. Therefore a debouncer module is implemented in the design. The codes are attached in Appendix B.

Unused I/O pins should remain unconnected as tying it to Vcc or ground could create contention that can damage the output driver of the device. Unused dedicated inputs should also be tied to the ground plane. Otherwise, pins may "float" in an indeterminate state, possibly increasing the DC current in the device and introducing noise. Other UP2 boards have been used. Interface is seldom successful with new boards because their flex expansion holes were larger and thus the connections with the test rigs become unreliable. This prompted the effort to solder the expansion slots to the interface to ensure reliability of connection.

4.12 Interface with TTL CPU

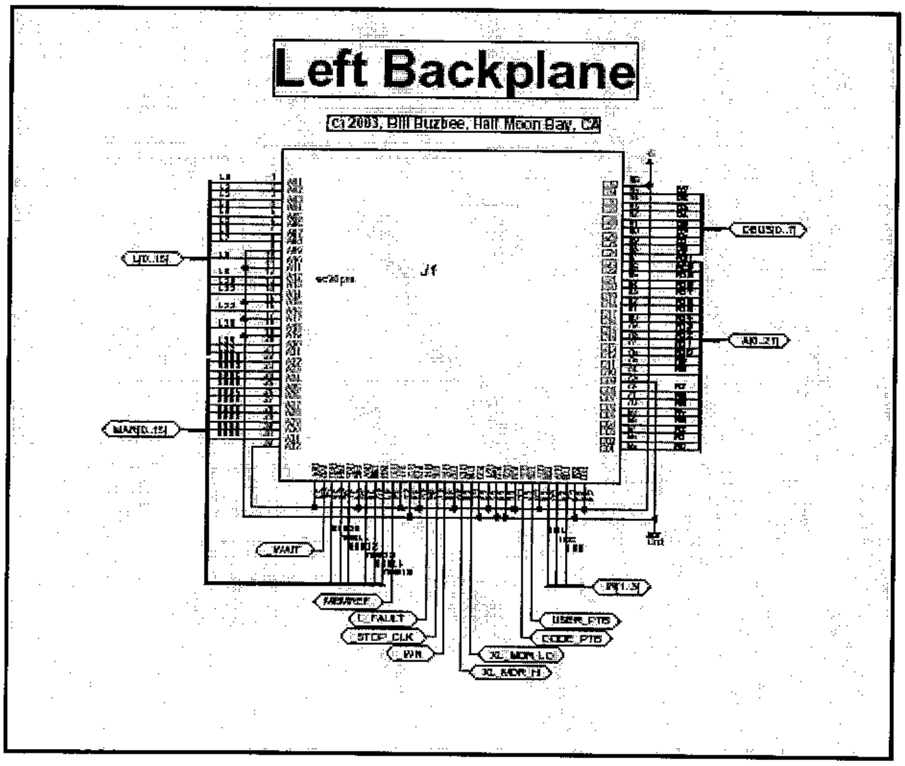


Figure 32: Left backplane

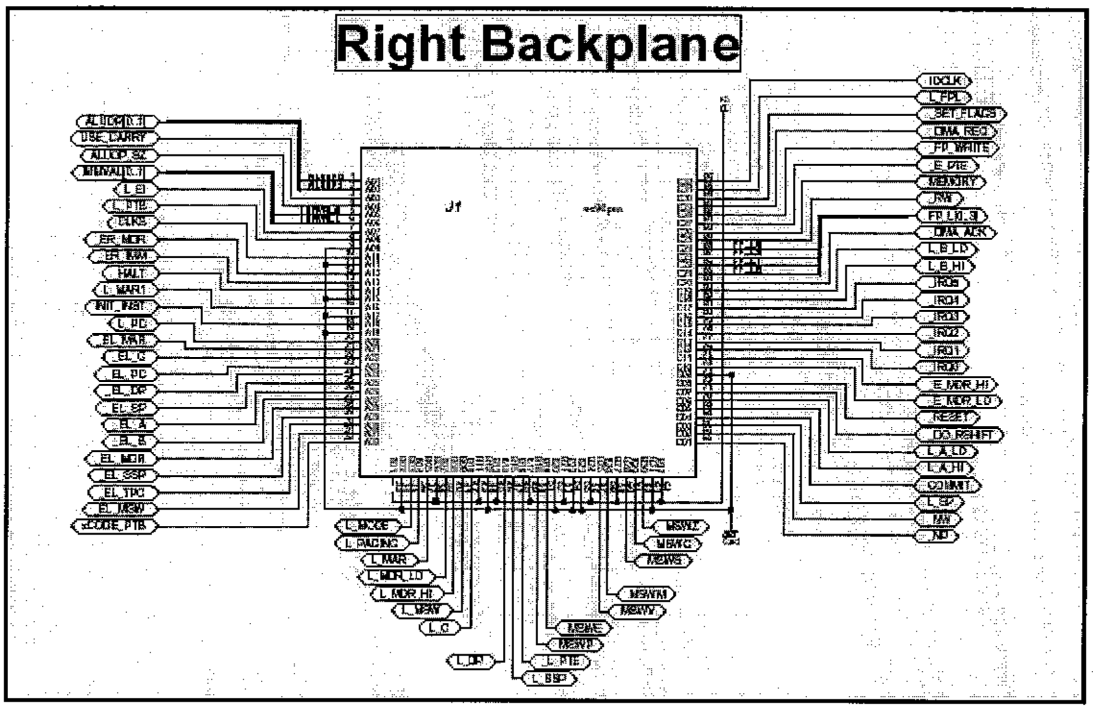


Figure 33: Right backplane

The interface with the TTL CPU is made to a connectionless veraboard. The interface is made to connect to the backplanes of the TTL CPU (Figure 32,33). 60 pins interface components which were initially unavailable was successfully sourced. IDE cables will connect the UP2 board to the veraboard and interfaced with the backplanes of the TTL CPU with 3 rows edge connectors. Connections are made by wire wrapping. Precautions not taken in the test rig were taken into account in the construction of the interface. The interface is shown in Appendix C (Figure 46,47). The size of the veraboard was cut to match the size of the TTL ALU card. However, the TTL CPU casing was designed in such a way that the thickness of the slot can only fit a particular PCB board. Therefore, an additional interface was constructed to be connected (see Appendix C).

During interfacing, the boot loader fails to upload the program to the PC. Deductions of possible causes include the previously mentioned data integrity during transmission, error in wire wrapping interface, and the differing operating requirements of the TTL CPU and the FPGA which is CMOS based. In addition, the discrepancies of speed of these two technologies could also be the cause. Significant propagation delay is sometime observed during tests conducted in test jig which highly likely is due to the long interface connections. Since the CPU is operating at 1Mhz, the built interface of the ALU card on FPGA will not be able to support it. Therefore, the next approach adopted was to load the Fibonacci program into the device card instead of the boot loader. Successful loading of the program should display the Fibonacci series. First, the original ALU card is used. However the program fails to output the Fibonacci series in the L bus. If the program did not work as expected on the original TTL ALU card, it is highly unlikely to function with the FPGA version. However, the L bus values observed as the manual clock is being clocked is similar for both the TTL ALU and the ALU card on FPGA.

4.13 TTL components in verilog

Each TTL chip that was previously entered as schematics have been programmed in Verilog with its functionality verified through software simulation. All the chips were coded except for the ALU chip and the carry-lookahead-generator which was combined and coded in a top down approach. The codes were appended in Appendix B.

4.14 Control card in verilog

The control card is also programmed in Verilog as separate modules into the microcode section, field decode section, field decode 2 section, and faults and interrupts section. The schematics are attached in Appendix A and the codes are appended in Appendix B.

CHAPTER 5
CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

The objective of the project is to implement the ALU of an 8 bit register-based CPU on FPGA. This would facilitate further research in computer architecture. The methodologies employed begin involves schematic entry into Quartus 2, simulations, synthesis of design, constructing the test jig, performing tests on the test jig, troubleshooting, constructing the interface with TTL, interfacing with TTL CPU, replacing schematic entry with Verilog HDL and also the implementation of the control card in FPGA. The issues that caused simulation to fail have been solved. The solutions include ensuring registers to a bus is only enabled one at a time and weak signals have to be specified for bidirectional nodes, among others. The test jig constructed is a set of generic switches and LEDs. Errors that have been resolved during testing include assigning pins that has to be driven a logic level to the test jig, disconnecting unused I/O pins, tying unused dedicated input pins to ground, soldering the interface and implementing a debouncer module in the design to address bounce-effect of mechanical switch used for clock signals. A detailed test was carried out to test the functionality of the ALU. The details are described in Chapter 4.8 until Chapter 4.11. The test results verified that the ALU is successfully implemented on FPGA.

Interface with TTL CPU is successfully constructed. Interface has not been successful yet and initial deduction for the cause include the discrepancies in speed of the FPGA and the TTL CPU. Schematic entry was replaced by verilog HDL codes. The control card has also been coded and simulated in individual modules. In conclusion, the ALU of the 8-bit register based CPU was successfully implemented on FPGA.

5.2 Recommendations

- Perform troubleshooting using oscilloscope to verify deductions made on interface
- Implement memory and device card on FPGA
- Design own computer architecture and implement using top down approach
- Measure performance difference by implementing different computer architecture

REFERENCES

[1] Unknown author, “Wikipedia, the free encyclopedia”, http://en.wikipedia.org.

[2] Bill Buzzbee, “Magic-1, Homebrew CPU”, 2003, http://www.homebrewcpu.com.

[3] Muhamad Aidil b Jazmi, “Development of an 8-bit CPU using TTL logic”, B.ENG Electrical & Electronic Final Year Project, Universiti Teknologi PETRONAS, Malaysia, 2006.

[4] M.G. Arnold, *Verilog Digital Computer Design*. Upper Saddle River: Prentice Hall, 1999.

[5] T.R. Padmanabhan, B.Bala, *Design Through Verilog HDL*. Hoboken: IEEE Press, 2004.

[6] Ken Koffman, *Real World FPGA Design with Verilog*. Upper Saddle River: Prentice Hall, 2000.

[7] Michael D.Ciletti, *Modelling, Synthesis, and Rapid Prototyping with Verilog HDL*. Upper Saddle River: Prentice Hall, 1999.

[8] James M. Lee, *Verilog Quickstart*. Kluwer Academic Publisher, 1999.

APPENDIX A – SCHEMATICS

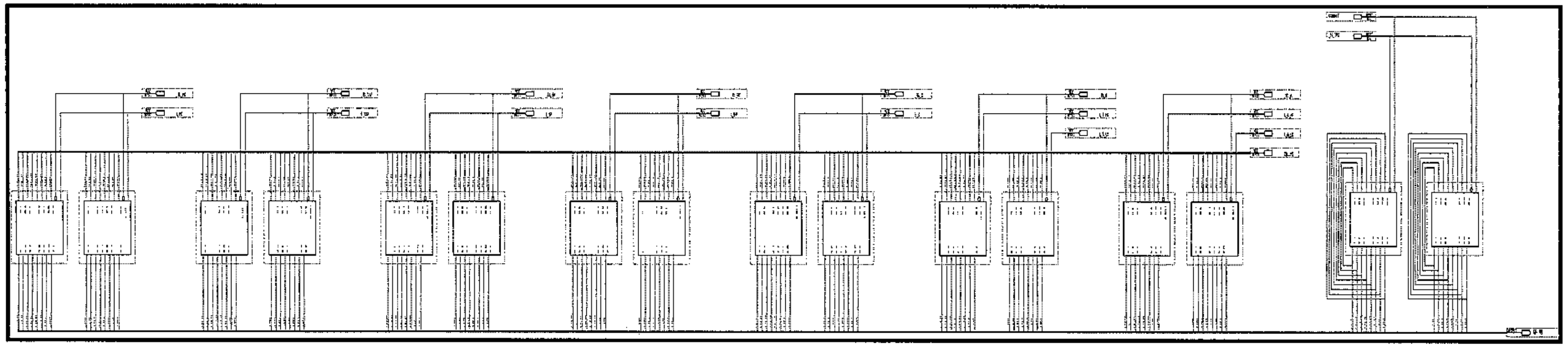


Figure 34: Schematic of general register

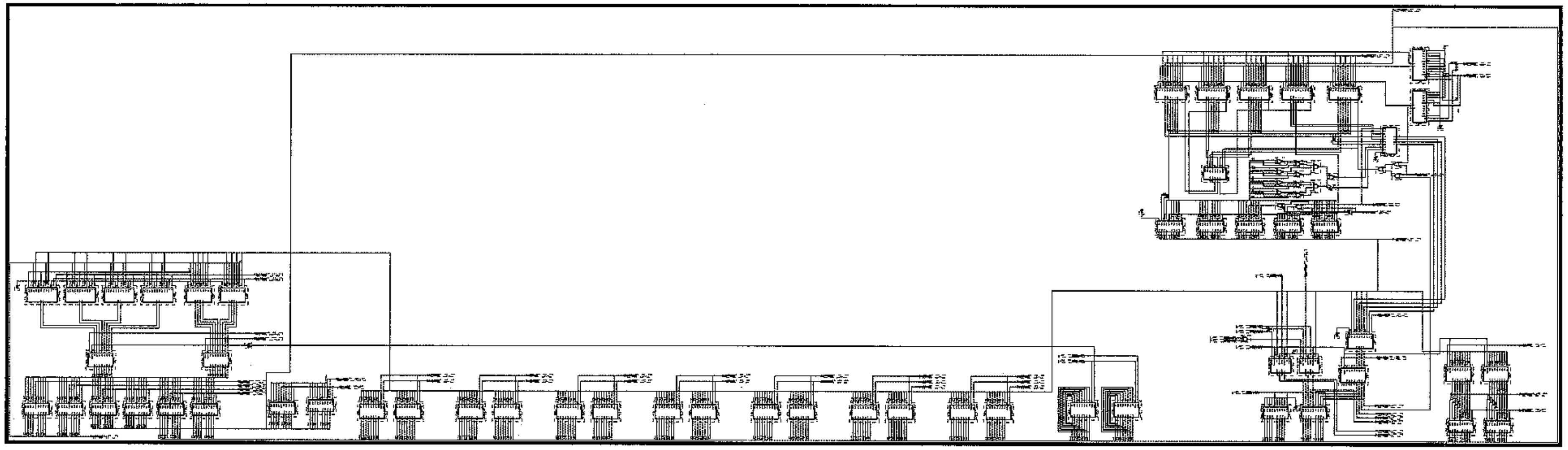


Figure 35: Schematic of integrated ALU

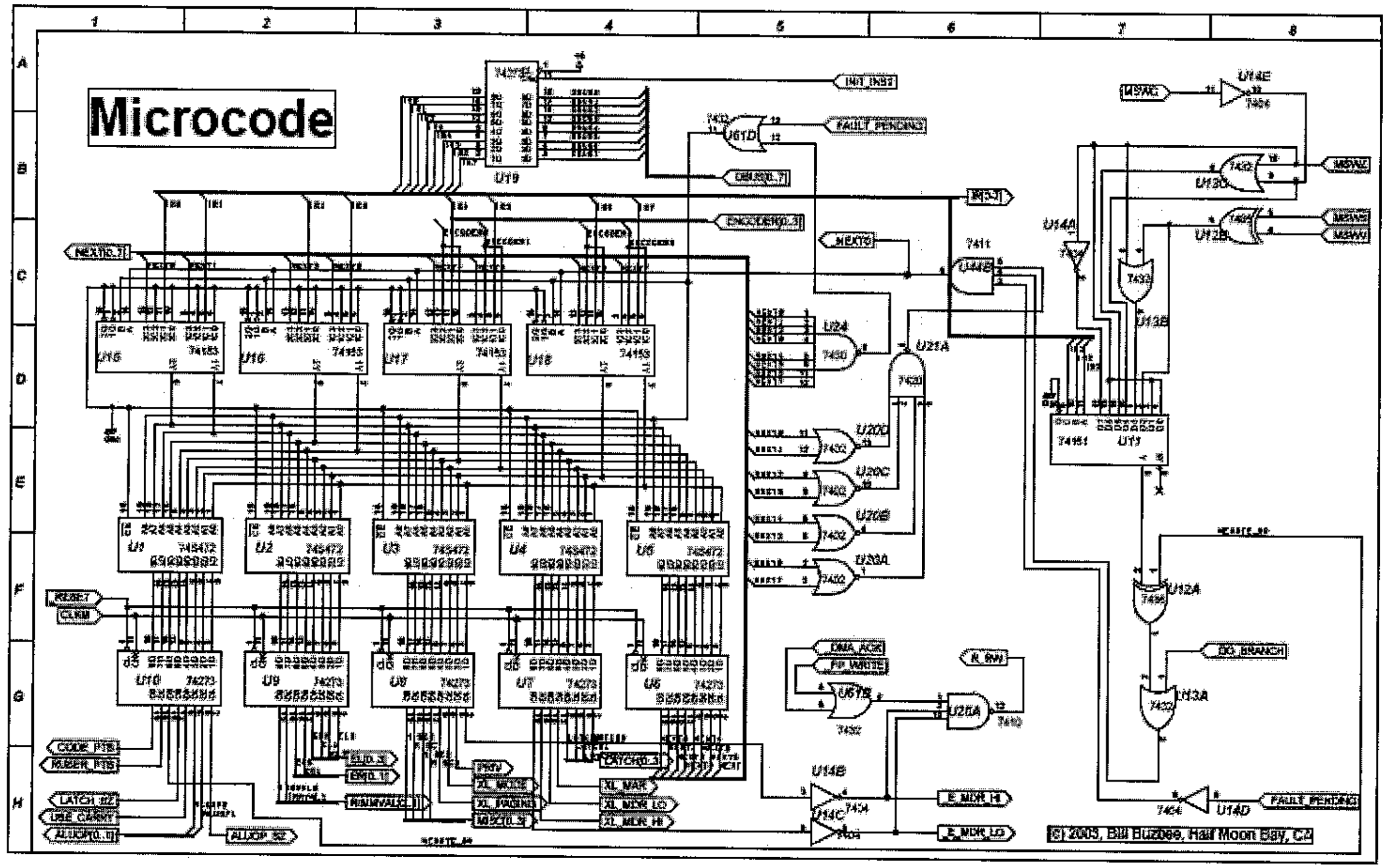


Figure 36: Microcode

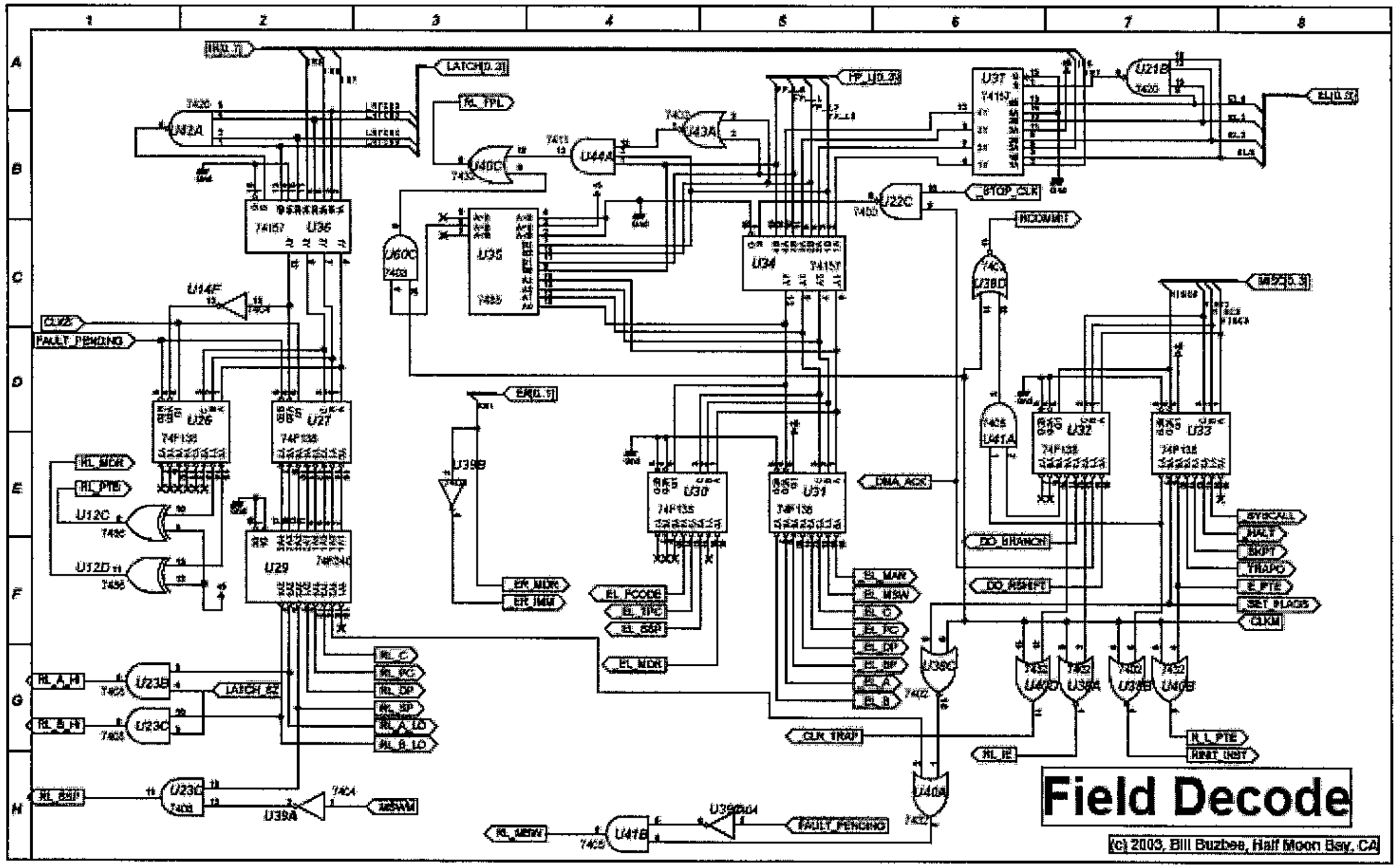


Figure 37: Field decode

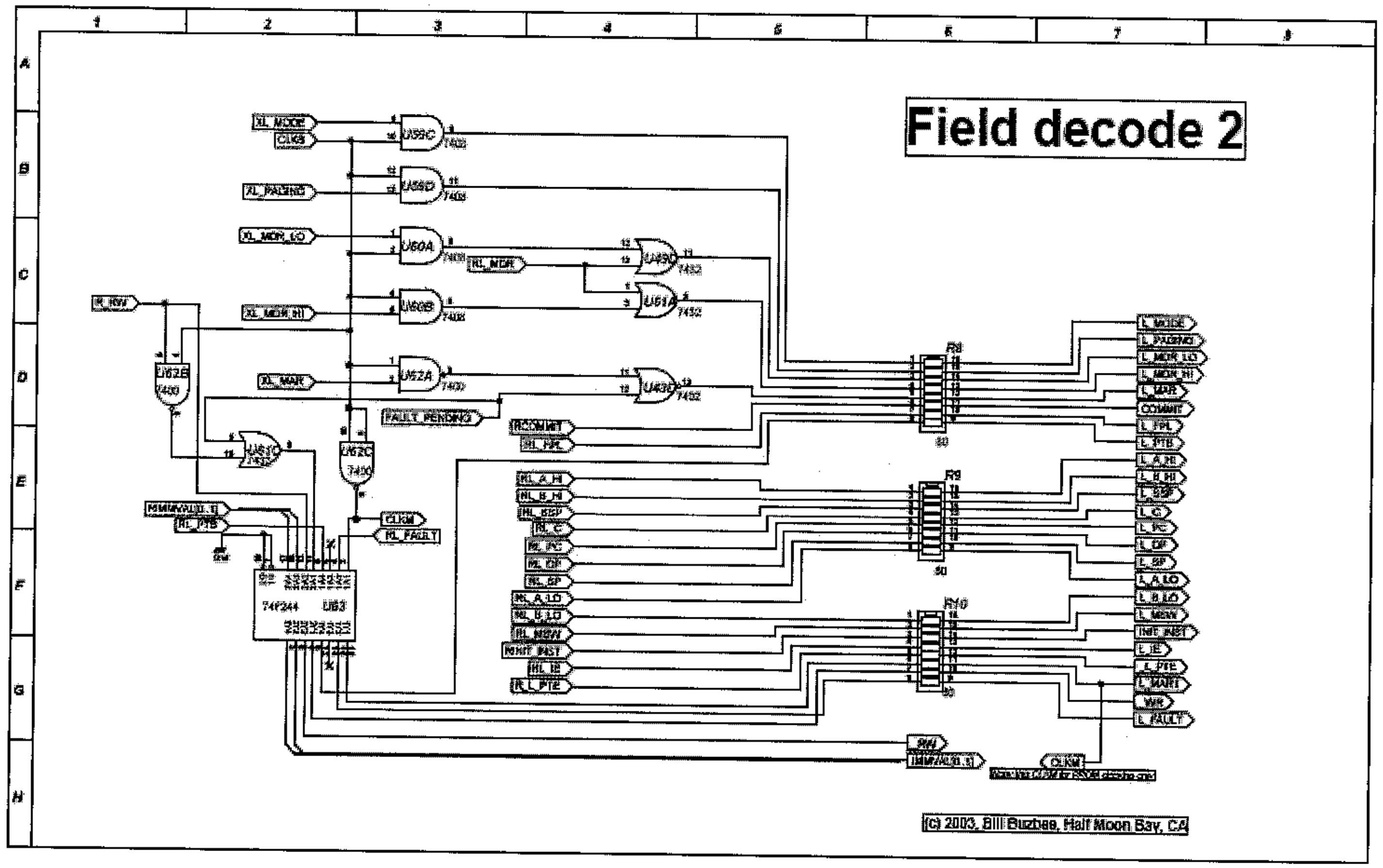


Figure 38: Field decode 2

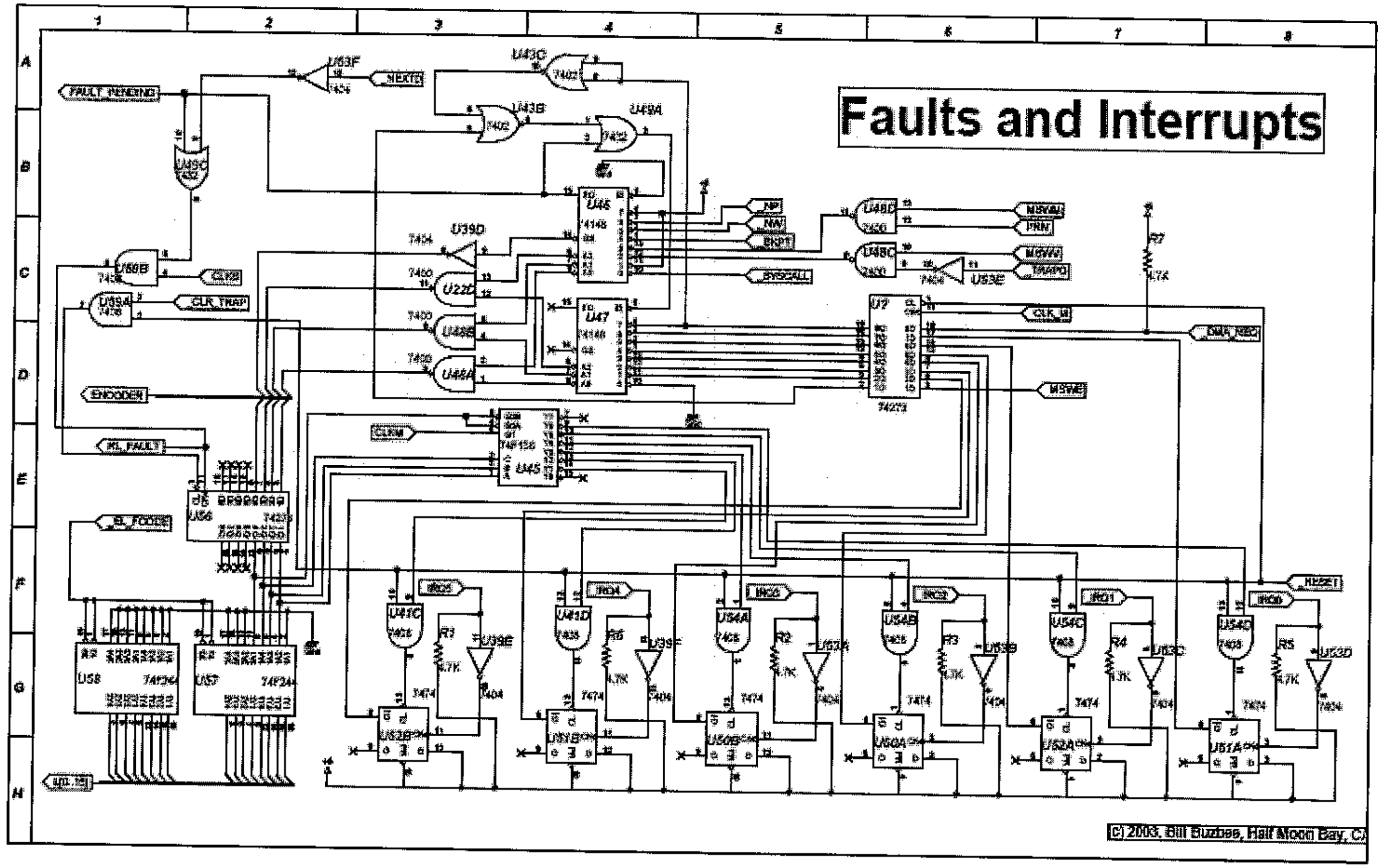


Figure 39: Faults and interrupts

APPENDIX B – VERILOG CODES

Bidirectional Buffer in Verilog

```
module tristate_buffer(out,in,en)
      parameter SIZE = 8;
      output out;
      input in, en;
      reg [SIZE-1:0] out;
      wire [SIZE-1:0] in;
      wire en;
      always @ (in or en)
            begin
                  if (en === 1)
                        out = in;
                  else if (en === 0)
                        out = 'bz;
                  else
                        out = 'bx;
            end
endmodule

module enabled_register (di, do, enable, clk)
      parameter SIZE = 8;
      input di, enable, clk;
      output do;
      reg [SIZE-1:0] do;
      wire [SIZE-1:0] di;
      wire enable;
      wire clk;
      always @ (posedge clk)
            if (enable)
                  do = di;

endmodule

module rw_register(bus, rd, wr, clk)
      parameter SIZE = 8;
      inout bus;
      input rd, wr, clk;
      wire [SIZE-1:0] bus;
      wire [SIZE-1:0] do;
      wire rd, wr;

      enabled_register #SIZE r1(bus,do,wr,clk);
      tristate_buffer #SIZE b1(bus,do,rd);

endmodule
```

74F273 in verilog

```
module reg273
(
            D, clk, clear, Q
);
// Port Declaration

            input [7:0] D;
      input clk;
      input clear;
      output [7:0] Q;


      reg Q;


      always @(posedge clk or negedge clear)
      begin
            if (clear == 0)
                  Q <= 8'b0;
            else
                  Q <= D;
      end

endmodule
```

7474 in Verilog

```
module flipflop7474
(

      d, clk, s, r, q, q_b

);
// Port Declaration

      input d;
      input clk;
      input s;
      input r;
      output q;
      output q_b;

      reg q, q_b;

      always @ (posedge clk or negedge s or negedge r)
      begin
            if (s == 0)begin
                  #4 q <= 1'b1;
                  #3 q_b <= 1'b0;
            end else if ( r == 0 ) begin
                  #4 q <= 1'b0;
                  #3 q_b <= 1'b1;
            end else begin
                  #4 q <= d;
                  #3 q_b <= ~d;
            end
      end


endmodule
```

74F374 on Verilog

```
module reg374
(
      D, CK, OC, Q
);
// Port Declaration


      input [7:0] D;
      input CK;
      input OC;
      output [7:0] Q;


      reg Q;

      always @(posedge CK or posedge OC)
      begin
            if (OC == 1)
                  Q <= 8'bz;
            else if (OC == 0)
                  #5 Q <= D;
            else
                  Q <= 8'bx;
      end

endmodule
```

74F244 on Verilog

```
module buf244
(

      A, en, Y

);
// Port Declaration


      input [7:0] A;
      input en;
      output [7:0] Y;


      reg Y;
      wire A, en;

      always @(A or en)
      begin
            if (en === 0)
                  Y <= A;
            else if (en === 1)
                  Y <= 8'bz;
            else
                  Y <= 8'bx;
      end


endmodule
```

74F153 on Verilog

```
module mux153
(
      sel, i0, i1, i2, i3, Y

);
// Port Declaration


      input [1:0] sel;
      input i0;
      input i1;
      input i2;
      input i3;
      output Y;

      wire i0, i1, i2, i3, sel;
      reg Y;

      always @ (i0 or i1 or i2 or i3 or sel)
      begin
      case (sel)
            2'b00 :
                  Y <= i0;
            2'b01 :
                  Y <= i1;
            2'b10 :
                  Y <= i2;
            2'b11 :
                  Y <= i3;
            default:
                  Y <= 1'b0;
      endcase
      end
endmodule
module mux157
(
      A, B, S, Y
);
// Port Declaration

      input [3:0] A;
      input [3:0] B;
      input S;
      output [3:0] Y;
      wire A, B, S;
      reg [3:0]Y;

      always @(A or B or S)
      begin
            if (S == 1'b0)
                  Y <= A;
            else
                  Y <= B;
      end

endmodule
```

ALU module

```
module      aluchip (Sel, A, B, Cin, _DO_RSHIFT, F, v, c, s, z);

input       [2:0] Sel;
input       [15:0] A, B;
input       Cin, _DO_RSHIFT;
output      [15:0] F;
output      v,c,s,z;
reg         F,v,c,s,z,c_out;
reg         [15:0] out;

always@(Sel,A,B,Cin,_DO_RSHIFT)
begin

      case(Sel)
            3'b000: out = 16'h0000;
            3'b001:
            begin
            out = A^B;
            z = out==0;
            end
            3'b010:
            begin
            {c_out,out} = A+(~B)+1'b1;
            z = out==0;
            s = out[15];
            c = ~c_out;
            v = c_out^out[15]^A[15]^B[15];
            end
            3'b011:
            begin
            out = A&B; z = out==0;
            end
            3'b100:
            begin
            {c_out,out} = B+(~A)+1'b1; z = out==0; s = out[15]; c =
~c_out; v = c_out^out[15]^A[15]^B[15];
            end
            3'b101:
            begin
            out = A|B; z = out==0;
            end
            3'b110:
            begin
            {c,out} = A+B+Cin;
            z = out==0;
            s = out[15];
            v = c^out[15]^A[15]^B[15];
            end
            3'b111: out = 16'hFFFF;

      endcase

      if(_DO_RSHIFT)
            F = (out >> 1);

      else
            F = out;

end

endmodule
```

Debouncer

```
module debouncer
(
      set, clear, Q, Q_b

);


      input set;
      input clear;
      output Q;
      output Q_b;
      reg Q, Q_b;

      nand(Q,set,Q_b);
      nand(Q_b,clear,Q);


endmodule

74151 (Multiplexor)

module mux151
(

      D, sel, W, Y);

      input [7:0] D;
      input [2:0] sel;
      output W;
      output Y;


      reg Y, W;

      always @ (D or sel)
      begin
            case (sel)
                  3'b000      : Y <= D[0];
                  3'b001      : Y <= D[1];
                  3'b010      : Y <= D[2];
                  3'b011      : Y <= D[3];
                  3'b100      : Y <= D[4];
                  3'b101      : Y <= D[5];
                  3'b110      : Y <= D[6];
                  3'b111      : Y <= D[7];
                  default : Y <= D[0];
            endcase

            W <= ~Y;
      end


endmodule
```

74138 (Decoder)

```
module dec138
(

      A, G1, G2A, G2B, Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7

);
```

```
        input [2:0] A;
        input G1;
        input G2A;
        input G2B;
        output Y0;
        output Y1;
        output Y2;
        output Y3;
        output Y4;
        output Y5;
        output Y6;
        output Y7;


        reg Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7;

        always @ (G1 or G2A or G2B or A)
        begin
        if (G1 && !G2A && !G2B) begin
               Y0 <= (A == 3'b000) ? 1'b0:1'b1;
               Y1 <= (A == 3'b001) ? 1'b0:1'b1;
               Y2 <= (A == 3'b010) ? 1'b0:1'b1;
               Y3 <= (A == 3'b011) ? 1'b0:1'b1;
               Y4 <= (A == 3'b100) ? 1'b0:1'b1;
               Y5 <= (A == 3'b101) ? 1'b0:1'b1;
               Y6 <= (A == 3'b110) ? 1'b0:1'b1;
               Y7 <= (A == 3'b111) ? 1'b0:1'b1;
         end

        else
        begin

               {Y0,Y1,Y2,Y3,Y4,Y5,Y6,Y7} <= 8'b11111111;
        end
        end
endmodule
```

74240 (Tri-state buffer)

```
module buf240
(

        A, en, Y

);


        input [7:0] A;
        input en;
        output [7:0] Y;


        reg Y;
        wire A, en;

        always @(A or en)
        begin
              if (en === 0)
                    Y <= ~A;
              else if (en === 1)
                    Y <= 8'bz;
```

```
            else
                    Y <= 8'bx;
        end


endmodule
```
7485 (Comparator)

```
module comp7485
(

        A, B, eq, A_gt_B, A_lt_B

);


        input [3:0] A;
        input [3:0] B;
        output eq;
        output A_gt_B;
        output A_lt_B;

        reg eq, A_gt_B, A_lt_B;

        always@(A or B)
        begin



        if (A > B)
        begin
        eq <= 0;
        A_gt_B <= 1;
        A_lt_B <= 0;
        end

        else if (A < B)
        begin
        eq <= 0;
        A_gt_B <= 0;
        A_lt_B <= 1;
        end

        else
        begin
        eq <= 1;
        A_gt_B <= 0;
        A_lt_B <= 0;
        end

        end


endmodule
```

74472 (512x8 (4096 bits) PROM)

```
module rom472
(

        A, CE, O
);


      input [8:0] A;
      input CE;
      output [7:0] O;
      reg [7:0] O;
      always @ (A,CE)
      begin
      if (CE == 1)
      O <= 8'bz;

      else
      begin
      case (A)

      /*Insert prom bits here. Below are test bits*/

      9'b000000000: O <= 8'b00000000;
      9'b000000001: O <= 8'b00000001;
      9'b000000010: O <= 8'b00000010;
      9'b000000011: O <= 8'b00000011;
      9'b111111100: O <= 8'b11111100;
      9'b111111101: O <= 8'b11111101;
      9'b111111110: O <= 8'b11111110;
      9'b111111111: O <= 8'b11111111;
      default: O <= 8'b11111111;
      endcase
      end
      end

endmodule
```

74148 (8-line to 3-line priority encoder)

```
module enc148
(
            EI, Data, EO, GS, A

);


      input EI;
      input [7:0] Data;
      output EO;
      output GS;
      output [2:0] A;


      reg [2:0] A;
      assign GS = &Data | EI;
      assign EO = ~&Data | EI;

      always@(Data,EI)

      if(EI) A=7;
      else
      begin
            casex(Data)
                  8'b0xxxxxxx :A=0;
                  8'b10xxxxxx :A=1;
                  8'b110xxxxx :A=2;
                  8'b1110xxxx :A=3;
                  8'b11110xxx :A=4;
                  8'b111110xx :A=5;
                  8'b1111110x :A=6;
                  8'b11111110 :A=7;
                  default          :A=7;
            endcase
      end


endmodule
```

Microcode section of control card

```
module microcode (INIT_INST, DBUS, IR, NEXT, _NEXT0, ENCODER,
_RESET, CLKM,
CODE_PTB,RUSER_PTB,LATCH_SZ,USE_CARRY,ALUOP,ALUOP_SZ,RIMMVAL,ER,
EL, MISC, XL_PAGING, XL_MODE, PRIV, XL_MDR_HI, XL_MDR_LO, XL_MAR,
LATCH,_E_MDR_LO, _E_MDR_HI,_DMA_ACK, _FP_WRITE, R_RW, MSWC, MSWZ,
MSWS, MSWV, _DO_BRANCH, FAULT_PENDING
);

output       [7:0] IR, NEXT;
output

      _NEXT0,CODE_PTB,RUSER_PTB,LATCH_SZ,USE_CARRY,ALUOP_SZ,XL_PAGIN
G, XL_MODE, PRIV;
output       XL_MDR_HI, XL_MDR_LO, XL_MAR, _E_MDR_LO, _E_MDR_HI,
R_RW;
output       [1:0] ALUOP, RIMMVAL,ER;
output       [3:0] EL, MISC, LATCH;
input        INIT_INST, _RESET, CLKM, _DMA_ACK, _FP_WRITE, MSWC,
             MSWZ, MSWS, MSWV, _DO_BRANCH, FAULT_PENDING;
input        [7:0] DBUS;
input        [3:0] ENCODER;
wire         [7:0]A, ROM1, ROM2, ROM3, ROM4, ROM5;
wire
IR,B,_NEXT0,NEGATE_BR,EMDRHI,EMDRLO,w1,w2,w3,w4,w5,w6,w7,w8,w9,w10,w
11,w12,NEXT,W,Y;

reg          w12,w3;

reg273 U19(DBUS, INIT_INST, 1, IR);
mux153 U15b({B,_NEXT0}, 0, IR[7], 0, NEXT[7], A[7]);
mux153 U15a({B,_NEXT0}, 0, IR[6], 0, NEXT[6], A[6]);
mux153 U16b({B,_NEXT0}, 0, IR[5], 0, NEXT[5], A[5]);
mux153 U16a({B,_NEXT0}, 0, IR[4], 0, NEXT[4], A[4]);
mux153 U17b({B,_NEXT0}, ENCODER[3], IR[3], ENCODER[3], NEXT[3],
A[3]);
mux153 U17a({B,_NEXT0}, ENCODER[2], IR[2], ENCODER[2], NEXT[2],
A[2]);
mux153 U18B({B,_NEXT0}, ENCODER[1], IR[1], ENCODER[1], NEXT[1],
A[1]);
mux153 U18a({B,_NEXT0}, ENCODER[0], IR[0], ENCODER[0], NEXT[0],
A[0]);

rom472 U1({B,A[7:0]}, 0, ROM1);
rom472 U2({B,A[7:0]}, 0, ROM2);
rom472 U3({B,A[7:0]}, 0, ROM3);
rom472 U4({B,A[7:0]}, 0, ROM4);
rom472 U5({B,A[7:0]}, 0, ROM5);
```

```
reg273 U10(ROM1, CLKM, _RESET,
{CODE_PTB,RUSER_PTB,NEGATE_BR,LATCH_SZ,USE_CARRY,ALUOP,ALUOP_SZ});
reg273 U9(ROM2, CLKM, _RESET, {RIMMVAL,ER, EL});
reg273 U8(ROM3, CLKM, _RESET, {MISC,XL_PAGING, XL_MODE, PRIV,
EMDRHI});
reg273 U7(ROM4, CLKM, _RESET, {EMDRLO,XL_MDR_HI, XL_MDR_LO, XL_MAR,
LATCH});
reg273 U6(ROM5, CLKM, _RESET, NEXT);

not U14C(_E_MDR_LO,EMDRLO);
not U14B(_E_MDR_HI,EMDRHI);

or U61B(w1,_DMA_ACK, _FP_WRITE);
nand U25A (R_RW, _E_MDR_LO, _E_MDR_HI, w1);

nor U20 (w2, NEXT[7], NEXT[6], NEXT[5], NEXT[4], NEXT[3], NEXT[2],
NEXT[1], NEXT[0]);
not U21 (w3, w2);

not U14E (w4, MSWC);
or U13C (w5, w4, MSWZ);
xor U12B (w6, MSWS, MSWV);
not U14A (w7, MSWZ);
or U13B (w8, MSWZ, w6);

mux151 U11 ({w7,MSWZ,w5,w4,w8,MSWZ,MSWZ}, IR[6:4], W, Y);

xor U12A (w9, Y, NEGATE_BR);
or U13A (w10, _DO_BRANCH, w9);

not U14D (w11, FAULT_PENDING);
and U44B (_NEXT0, w3, w10, w11);

nand U24 (w12, NEXT[7], NEXT[6], NEXT[5], NEXT[4], NEXT[3], NEXT[2],
NEXT[1], NEXT[0]);
or U61D (B, w12, FAULT_PENDING);



endmodule
```

Field Decode of control card

```
module fielddec ( EL, IR, MISC, _SYSCALL, _HALT, _BKPT, _TRAP0,
_E_PTE, _SET_FLAGS,_DO_RSHIFT, _DMA_ACK, _DO_BRANCH, _CLR_TRAP,
RL_IE, CLKM, RINIT_INST, R_L_PTE, RCOMMIT, FP_L,_EL_MAR, _EL_MSW,
_EL_C, _EL_PC, _EL_DP, _EL_SP, _EL_A, _EL_B, _EL_MDR, _EL_SSP,
_EL_TPC, _EL_FCODE,
RL_FPL, _ER_MDR, _ER_IMM, LATCH, CLKS, FAULT_PENDING, RL_MDR,
RL_PTB, LATCH_SZ,RL_B_LO, RL_A_LO, RL_SP, RL_DP, RL_PC, RL_C,
RL_MSW, RL_SSP, MSWM, ER, _STOP_CLK

);

input       [7:0] IR;
input       [3:0] EL, MISC, FP_L, LATCH;
input       [1:0] ER;
input       CLKM, _STOP_CLK, CLKS, FAULT_PENDING, LATCH_SZ, MSWM;
output      _SYSCALL, _HALT, _BKPT, _TRAP0, _E_PTE, _SET_FLAGS,
            _DO_RSHIFT, _DMA_ACK, _DO_BRANCH;
output      _CLR_TRAP, RL_IE, RINIT_INST, R_L_PTE, RCOMMIT, _EL_MAR,
            _EL_MSW, _EL_C, _EL_PC, _EL_DP,
             _EL_SP, _EL_A, _EL_B, _EL_MDR, _EL_SSP, _EL_TPC,
            _EL_FCODE, RL_FPL, _ER_MDR, _ER_IMM,
            RL_MDR, RL_PTB, RL_B_LO, RL_A_LO, RL_SP, RL_DP, RL_PC,
            RL_C, RL_MSW, RL_SSP;
wire        w1, w2, w3, w4, w5, w6, w7, w8, w9, w10, w11, w12, w13,
            w14, w15, w16, w17, w18, w19,w20,w21,
            w22, w23, w24, w25, _SET_FLAGS, E_PTE, _DMA_ACK, eq,
            A_gt_B, A_lt_B, RL_A_LO, RL_B_LO, RL_SP;
wire        [3:0] Y, out, result;
wire        [7:0] con, conB;


nand U21B (w1, EL[3], EL[2], EL[1], EL[0]);
mux157 U37 ({2'b01,IR[1],IR[0]}, EL, w1, Y);

dec138 U33 (MISC[2:0], 1'b1, MISC[3], 1'b0, w2, _SYSCALL, _HALT,
_BKPT, _TRAP0, _E_PTE, _SET_FLAGS, w3);
dec138 U32 (MISC[2:0], MISC[3], 1'b0, 1'b0, _DO_RSHIFT, _DMA_ACK,
w6, _DO_BRANCH, w7, w5, w4, w8);

nor U38C (w9, CLKM, _SET_FLAGS);
or U40A (w10, w9, w22);
not U39C (w24, FAULT_PENDING);
and U41B (RL_MSW, w24, w10);

or U40D (_CLR_TRAP, CLKM, w7);
nor U38A (RL_IE, CLKM, w6);
nor U38B (RINIT_INST, CLKM, w3);
or U40B (R_L_PTE, CLKM, E_PTE);

and U41A (w11, w3, w5);
nor U38D (RCOMMIT, CLKM, w11);

nand U22C (w12, _STOP_CLK, _DMA_ACK);
mux157 U34 (Y, FP_L, w12, out);

dec138 U31 (out[2:0], 1'b1, out[3], 1'b0, _EL_MAR, _EL_MSW, _EL_C,
_EL_PC, _EL_DP, _EL_SP, _EL_A, _EL_B);
dec138 U30 (out[2:0], out[3], 1'b0, 1'b0, _EL_MDR, w13, _EL_SSP,
_EL_TPC, _EL_FCODE, w14, w15, w16);

nor U43A (w17, FP_L[2], FP_L[1]);
and U44A (w18, FP_L[0], FP_L[3]);

comp7485 U35 (out, FP_L, eq, A_gt_B, A_lt_B);
and U60C (w19, eq, CLKM);

or U40C (RL_FPL, w18, w19);

assign _ER_MDR = ER[0];
not U39B (_ER_IMM,ER[0]);

nand U42A (w20, LATCH[3], LATCH[2], LATCH[1], LATCH[0]);
mux157 U36 ({1'b0,IR[2:0]}, LATCH, w20, result);
not U14F (w21, result[3]);

dec138 U27 (result[2:0], CLKS, result[3], FAULT_PENDING, con[0],
con[1], con[2], con[3], con[4], con[5], con[6], con[7]);
dec138 U26 (result[2:0], CLKS, w21, FAULT_PENDING, conB[0], conB[1],
conB[2], conB[3], conB[4], conB[5], conB[6], conB[7]);

xor U12C (RL_PTB, conB[1], 1'b1);
xor U12D (RL_MDR, conB[0], 1'b1);

buf240 U29 (con, 1'b0, {RL_B_LO, RL_A_LO, RL_SP, RL_DP, RL_PC, RL_C,
w22, w23});
and U23B (RL_A_HI, RL_A_LO, LATCH_SZ);
and U23C (RL_B_HI, RL_B_LO, LATCH_SZ);

not U39A (w25, MSWM);
and U23D (RL_SSP, w25, RL_SP);

endmodule
```

Field decode 2 of control card

```
module fielddec2 (XL_MODE, CLKS, XL_PAGING, XL_MDR_LO, XL_MDR_HI,
XL_MAR, R_RW, FAULT_PENDING, RL_MDR, RL_PTB,RL_FAULT,L_MODE,
L_PAGING, CLKM, L_MDR_LO, L_MDR_HI, L_MAR, _RW, _WR, L_PTB, L_FAULT,
L_MAR1,
COMMIT, L_FPL, L_A_HI, L_B_HI, L_SSP, L_C, L_PC, L_DP, L_SP, L_A_LO,
L_B_LO, L_MSW, INIT_INST,L_IE, L_PTE, RIMMVAL, RCOMMIT, RL_FPL,
RL_A_HI, RL_B_HI, RL_SSP, RL_C, RL_PC, RL_DP, RL_SP, RL_A_LO,
RL_B_LO,
RL_MSW, RINIT_INST, RL_IE, R_L_PTE, IMMVAL);

input        XL_MODE, CLKS, XL_PAGING, XL_MDR_LO, XL_MDR_HI, XL_MAR,
             R_RW, FAULT_PENDING, RL_MDR, RL_PTB,RL_FAULT, RCOMMIT,
             RL_FPL, RL_A_HI, RL_B_HI, RL_SSP, RL_C, RL_PC, RL_DP,
             RL_SP, RL_A_LO, RL_B_LO,RL_MSW, RINIT_INST, RL_IE,
             R_L_PTE;
input        [1:0] RIMMVAL;

output       L_MODE, L_PAGING, L_MDR_LO, L_MDR_HI, L_MAR, _RW, _WR,
             L_PTB, L_FAULT, CLKM, L_MAR1,COMMIT, L_FPL, L_A_HI,
             L_B_HI, L_SSP, L_C, L_PC, L_DP, L_SP, L_A_LO, L_B_LO,
             L_MSW, INIT_INST,L_IE, L_PTE;
output       [1:0] IMMVAL;
wire         W1, W2, W3, W4, W5, W6, CLKM, L_MAR1, COMMIT,L_FPL,
             L_MAR1, COMMIT, L_FPL, L_A_HI, L_B_HI, L_SSP, L_C, L_PC,
             L_DP, L_SP, L_A_LO, L_B_LO, L_MSW,INIT_INST, L_IE,
             L_PTE;

and U59C (L_MODE, XL_MODE, CLKS);
and U59D (L_PAGING, XL_PAGING, CLKS);
and U60A (W1, XL_MDR_LO, CLKS);
and U60B (W2, XL_MDR_HI, CLKS);
nand U62A (W3, XL_MAR, CLKS);

nand U62C (CLKM, CLKS, CLKS);
nand U62B (W4, R_RW, CLKS);
or U61C (W5, W4, FAULT_PENDING);

or U49D (L_MDR_LO, W1, RL_MDR);
or U61A (L_MDR_HI, W2, RL_MDR);
nor U43D (L_MAR, W3, FAULT_PENDING);

buf244 U63 ({RIMMVAL, R_RW, W5, RL_PTB, 1'b0, RL_FAULT, CLKM}, 1'b0,
{IMMVAL, _RW, _WR, L_PTB, W6, L_FAULT, CLKM});

assign L_MAR1 = CLKM;

assign COMMIT = RCOMMIT;
assign L_FPL = RL_FPL;
assign L_A_HI = RL_A_HI;
assign L_B_HI = RL_B_HI;
assign L_SSP = RL_SSP;
assign L_C = RL_C;
assign L_PC = RL_PC;
assign L_DP = RL_DP;
assign L_SP = RL_SP;
assign L_A_LO = RL_A_LO;
assign L_B_LO = RL_B_LO;
assign L_MSW = RL_MSW;
assign INIT_INST = RINIT_INST;
assign L_IE = RL_IE;
assign L_PTE = R_L_PTE;


endmodule
```

Faults and Interrupts

```
module interrupts (_IRQ0, _IRQ1, _IRQ2, _IRQ3, _IRQ4, _IRQ5, _RESET,
_DMA_REQ, MSWE, CLK_M, _EL_FCODE, CLKM,_TRAPO, MSWM, PRIV, MSWV,
ENCODER, RL_FAULT, FAULT_PENDING, L, _NEXT0, CLKS, _CLR_TRAP,_NP,
_NW, _BKPT,_SYSCALL);

input        _IRQ0, _IRQ1, _IRQ2, _IRQ3, _IRQ4, _IRQ5, _RESET,
             _DMA_REQ, MSWE, CLK_M, _EL_FCODE, CLKM,_TRAPO, MSWM,
             PRIV, MSWV, _NEXT0, CLKS, _CLR_TRAP,_NP, _NW,
             _BKPT,_SYSCALL;

output       RL_FAULT, FAULT_PENDING;
output       [15:0] L;
output       [3:0] ENCODER;

wire         q1, q2, q3, q4, q5, q6, RL_FAULT, FAULT_PENDING, W1, W2,
             W3,W4,W5,W6,W7,W8,W9,W10,W11,W12,W13,W14,W15,W16,W17,W18
             ,W19,W20,W21,W22,W23,W24,W25,W26,W27,W28,W29,W30,W31,W32
             ,W33,W34,W35,W36,W37,W38;

wire  [3:0]wai, bai, ENCODER, out, A, B;
wire  [7:0] Q;

not U53D (W1, _IRQ0);
not U53C (W2, _IRQ1);
not U53B (W3, _IRQ2);
not U53A (W4, _IRQ3);
not U39F (W5, _IRQ4);
not U39E (W6, _IRQ5);

and U54D (W7, _RESET, W13);
and U54C (W8, _RESET, W14);
and U54B (W9, _RESET, W15);
and U54A (W10, _RESET, W16);
and U41D (W11, _RESET, W17);
and U41C (W12, _RESET, W18);

flipflop7474 U51A (1'b1, W1, 1'b1, W7, q1, W19);
flipflop7474 U52A (1'b1, W2, 1'b1, W8, q2, W20);
flipflop7474 U50A (1'b1, W3, 1'b1, W9, q3, W21);
flipflop7474 U50B (1'b1, W4, 1'b1, W10, q4, W22);
flipflop7474 U51B (1'b1, W5, 1'b1, W11, q5, W23);
flipflop7474 U52B (1'b1, W6, 1'b1, W12, q6, W24);
```

```
reg273 U ({_DMA_REQ, W19, W20, W21, W22, W23, W24, MSWE}, CLK_M,
_RESET, Q);

not U53F (W25, _NEXT0);
or U49C (W26, W25, FAULT_PENDING);
and U59B (RL_FAULT, CLKS, W26);
and U59A (W27, _CLR_TRAP, _RESET);

reg273 U56 ({wai,ENCODER} , RL_FAULT, W27, {bai,out});
buf244 U57 ({3'b0, out, 1'b0}, _EL_FCODE, L[7:0]);
buf244 U58 ({8'b0}, _EL_FCODE, L[15:8]);

dec138 U45 (out[2:0], CLKM, out[3], out[3], W29, W18, W17, W16, W15,
W14, W13, W28);

not U53E (W30, _TRAP0);
nand U48D (W31, MSWM, PRIV);
nand U48C (W32, MSWV, W30);

enc148 U46 (1'b0, {1'b1, _NP, _NW, _BKPT, W31, W32,1'b1,_SYSCALL},
FAULT_PENDING, W34, A);
enc148 U47 (W35, {Q[7:1],1'b0}, W36, W37, B);

not U39D (ENCODER[3], W34);
nand U22D (ENCODER[2], A[2], B[2]);
nand U48B (ENCODER[1], A[1], B[1]);
nand U48A (ENCODER[0], A[0], B[0]);

nor U43C (W38, Q[7], Q[7]);
nor U43B (W33, W38, Q[0]);
or U49A (W35, W33, FAULT_PENDING);

endmodule
```

APPENDIX C – PICTURES

Figure 40: A working MDR module on test rig

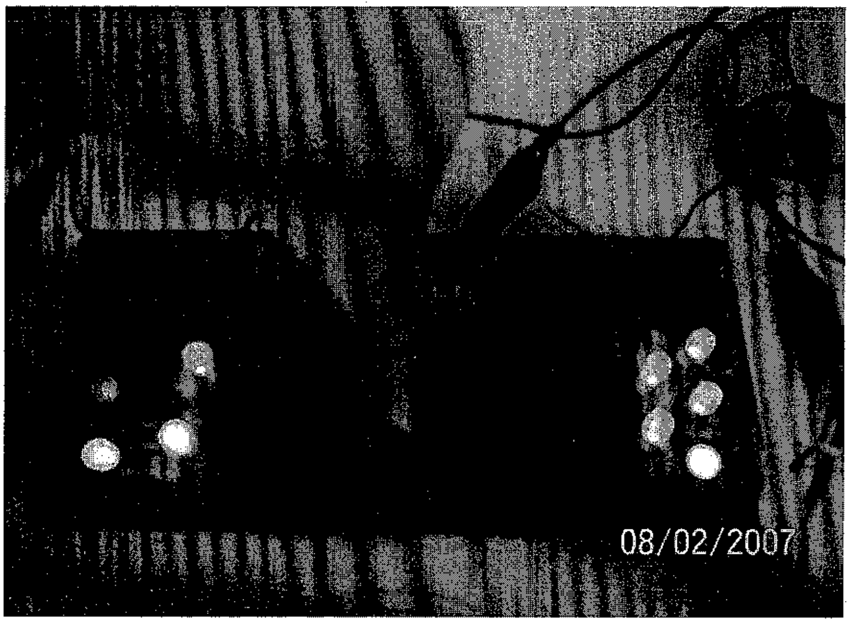


Figure 41: AND operation of ALU module

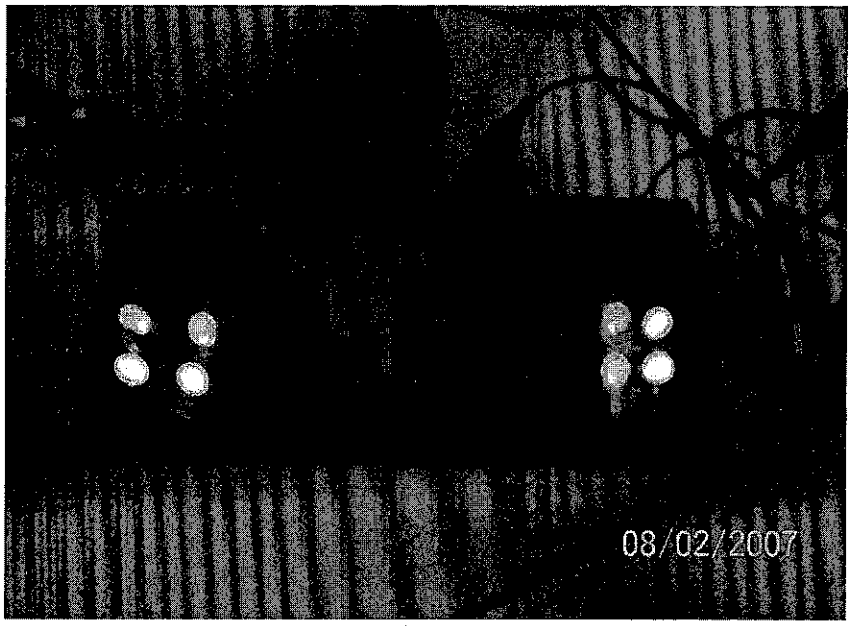


Figure 42: Minus operation of ALU module

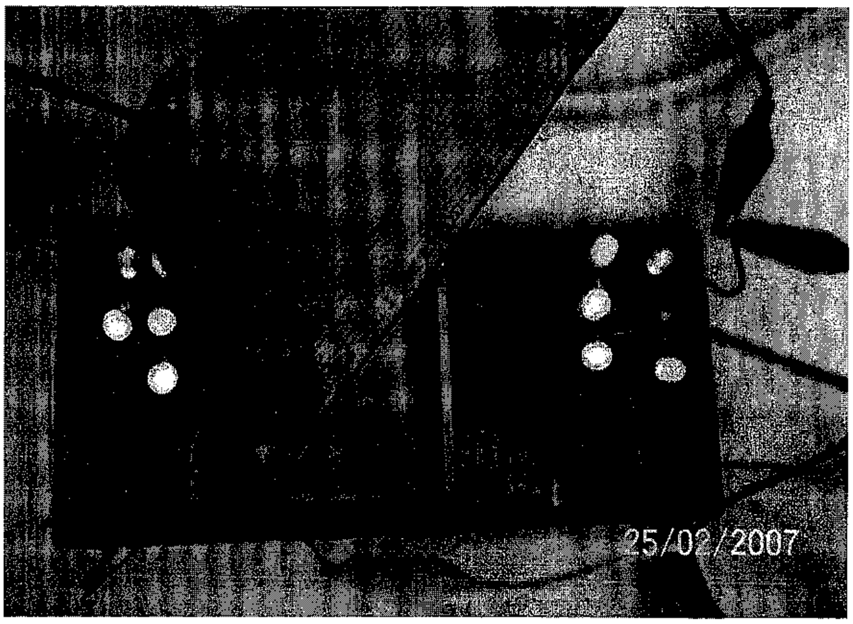


Figure 43: AND operation of integrated ALU with both carry and overflow lighted

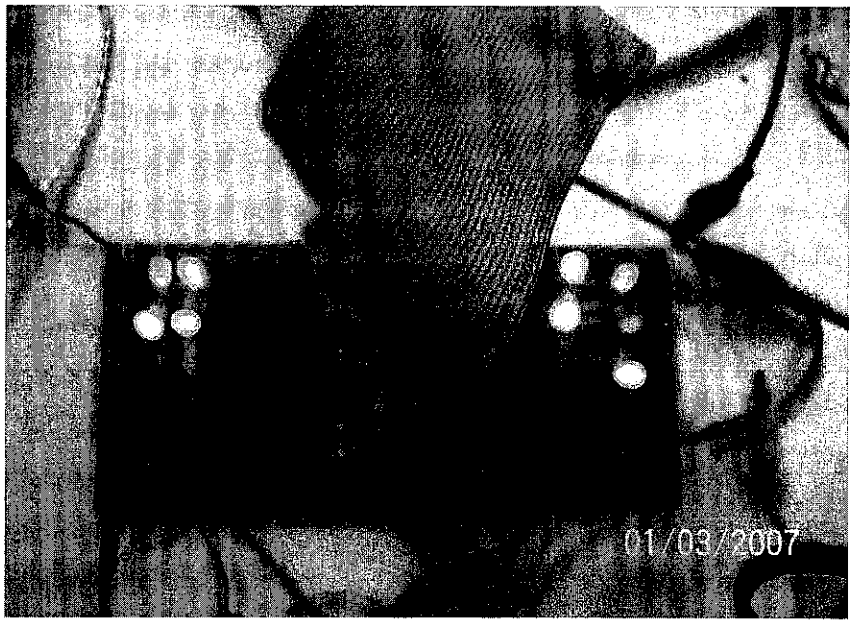


Figure 44: Add operation of integrated ALU with sign lighted

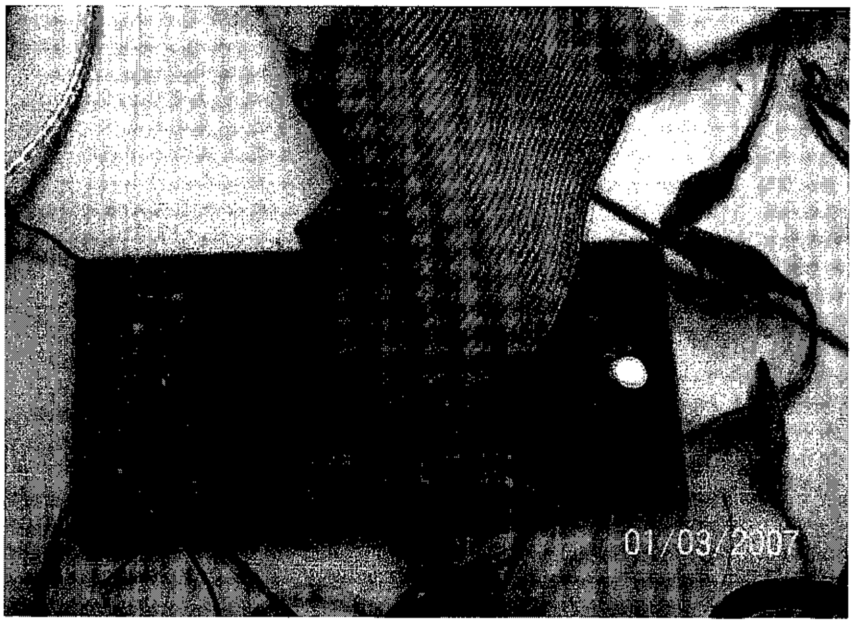


Figure 45 : Minus operation of integrated ALU with zero flag lighted

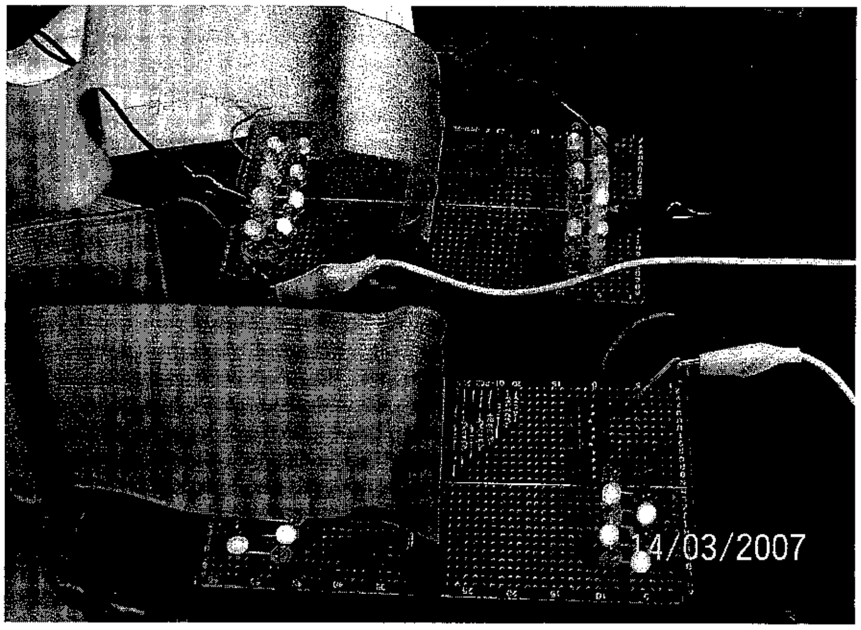


Figure 46: Results in L, R and Z busses

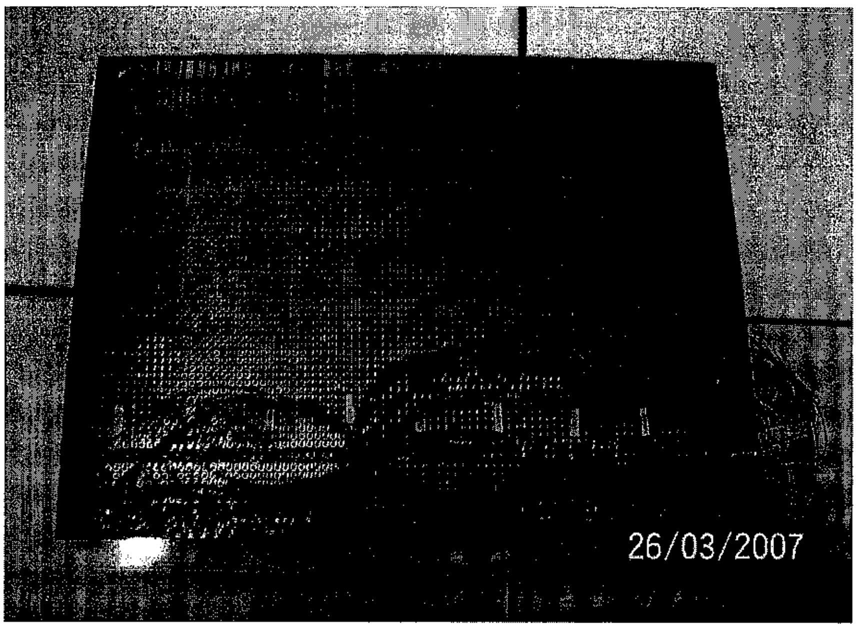


Figure 47: Wire wrap connections of interface

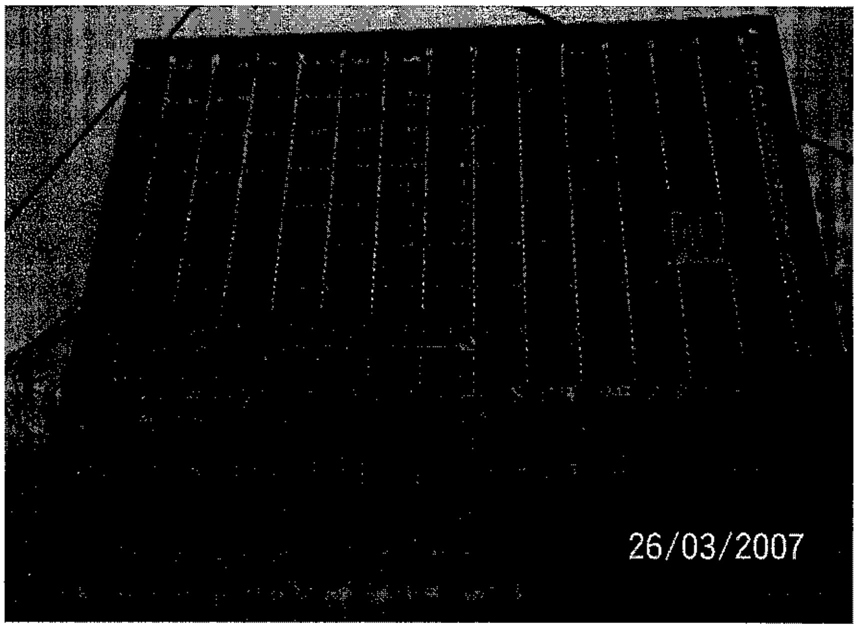


Figure 48: Top view of interface

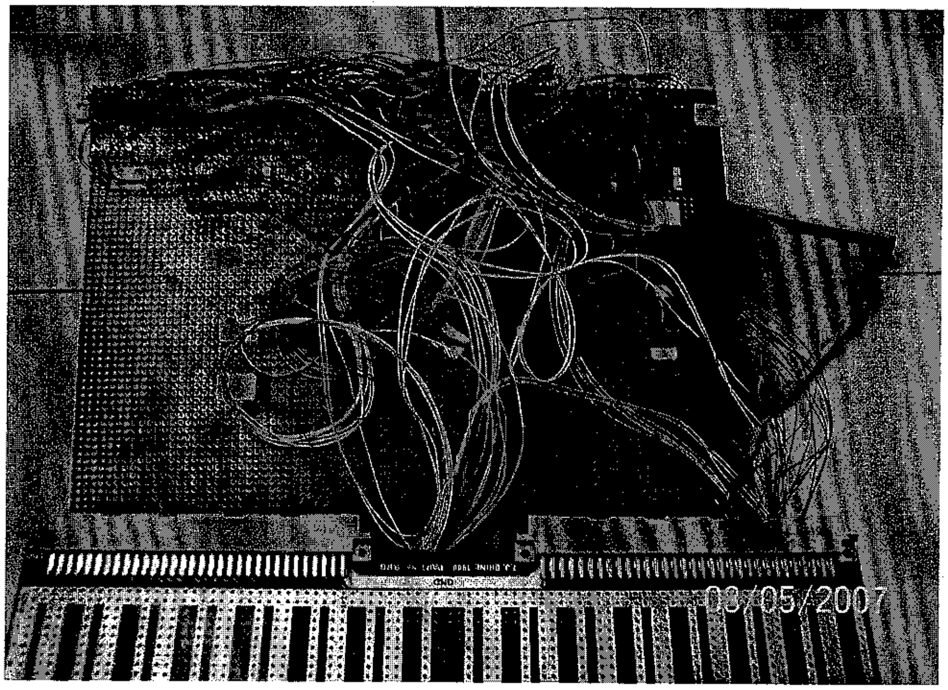


Figure 49: Final interface

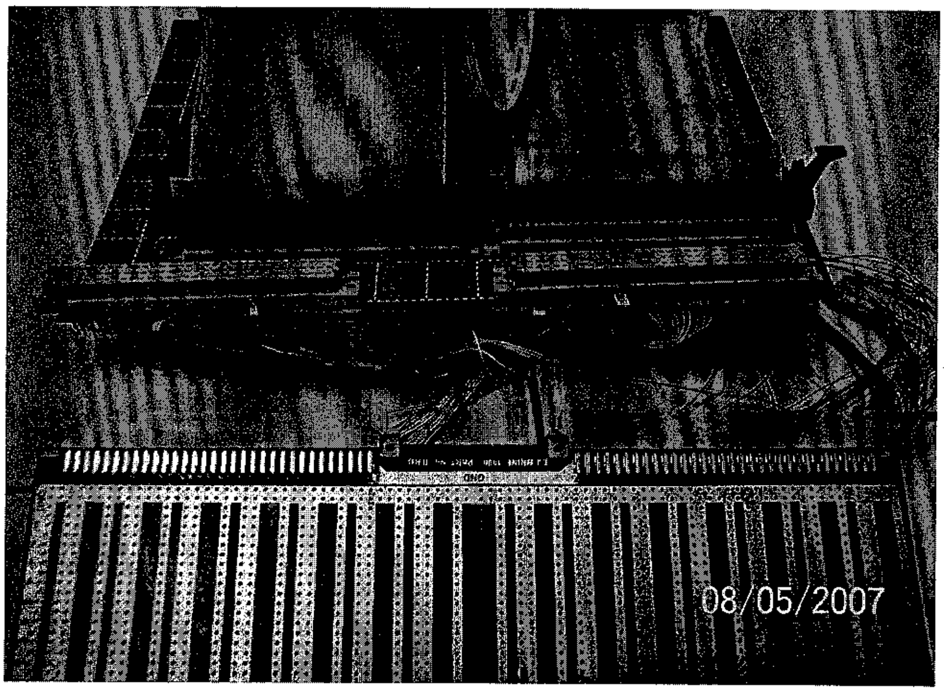


Figure 50: Top view of final interface