8 Bit CISC Description using Verilog & Synthesis

K.Narayanan

ECE Department
George Washington University
Spring 2002

The Big Picture

The ECE 128 course is a VLSI testing course that introduces us to an important phase in the process of chip manufacture -*Testing*. A designed chip typically undergoes rigorous testing before reaching the customer. The hierarchy of testing begins with the designer verifying the design for functionality – *Functional testing.* The designer keeps in mind the testing requirements after manufacture i.e. his design should reflect the ease with which the manufactured chip can be tested. Thus the designer's objective is not only limited to design for manufacture but also design for testing.

The modern day designers have a plethora of tools to choose from. Sophisticated programs automate and abstract various stages of design. Logic or Layout realization using these CAD tools has lead to smaller lead-time in product development.

Virtual testing and simulation has been the order of the day in the IT era. Spice, Cadence has evolved to such a sophistication, making virtual testing and simulation a reality.

The design of a CISC chip using Verilog HDL exploits the above-mentioned advantage of testing the design extensively using Cadence. The design can be used as an input to other software ATPG (Synopsis) for test vector generation. The test vectors thus chosen can be used to detect faults using single stuck at models – Verifault is a fault simulator that gives the fault coverage of a HDL model of a design. Any sane person would now be wondering why do all this?

It is to deliver a product that meets the specifications not just on paper, but a design, that has undergone the rigors of simulation in virtual testbenches, a design that has been modeled for faults to identify test patterns for conformance tests for the manufactured chips.

In the tandem of events that any product undergoes in its development called – Rapid Prototyping. HDL plays a vital role in system description. Description of a system can be categorized into – description for testing, description for synthesis (ciruit realisation).As described earlier since both go hand in hand. This project involves understanding Verilog for testing and synthesis.

Test modules and specifications for the 8Bit CISC processor was provided. The ALU, PC, CU, IO modules were implemented in Verilog. The individual modules were simulated and synthesized before wiring them up into a chip. The chip in turn was tested after integration.

The Big Picture 2

1. 0 Introduction 5

2. 0 Modular Hierarchy 6

2.1 Modularization 6

2.2 Specification & Requirements 6

3.0 Verilog Implementation 12

3.1 Integrated CPU 12

3.2 ALU Implementation 13

3.3 Program Counter Implementaion 20

3.4 Control Unit Implementation 23

3.5 Input Output Implementation 30

4.0 Simulation Results 32

4.1 ALU Simulation 32

4.2 Simulation of Program Counter 34

4.3 Control Unit Simulation 36

4.4 Simulation of CPU – CPU_test 37

5.0 Synthesized Area of Chip, Modules with constraint on optimize area 39

5.1 Integrated CPU Area 39

5.2 Integrated ALU module area 40

5.3 Integrated Program Counter module area 40

5.4 Input-Output module area 41

5.5 Control Unit Module Area 41

6.0 Synthesis Snapshots & Wave Form Snapshots 42

6.1.1 ALU Synthesized 42

6.1.2 ALU Symbol 42

6.1.3 Register Symbol....42

6.1.4 Adder-Subtractor Symbol....43

6.1.5 Adder-Subtractor Schematic....43

6.1.6 Carry Register Schematic....44

6.1.7 ALU Wave Form Output....44

6.1.8 ALU Integrated....45

6.2.1 Program Control Integrated Symbol....45

6.2.2 Next PC Generation Symbol....46

6.2.3 PC-Integrated Schematic....46

6.2.4 Program Control – Branch Genration Schematic....47

6.2.5 Program Control Waveform Output....48

6.3.1 Control Unit CU – MOD & Decode symbol....48

6.3.2 Control unit Integrated Symbol....49

6.3.3 Decoder Schematic....50

6.3.4 CU Waveform Output....50

6.4.0 CPU Integrated Symbol....51

6.4.1 CPU_test Waveform....51

6.4.2 Script File for CPU synthesis....52

7.0 Timing Report....53

7.1 CPU Timing Worst &Best....53

7.2 Control Unit Timing Worst &Best....54

7.3 ALU timing....56

7.4 Program Control timing Worst & Best....57

7.5 Input Output Timing Worst & Best....59

8.0 Scope , Limitations of the project and Future work....60

1. 0 Introduction

HDL like English, C or Engineering drawing is a means of communicating ideas. It has its rules and grammar that gives it a structure and still affords the flexibility for every individual to establish his or her style of expression.

Verilog that way affords a deceptive familiarity with C, but any one who has worked long enough with it would tell tales, of how it differs. Yet for beginners this familiarity helps in learning the syntax of the language.

The project given to us by Prof. Ms. M.E.Zaghloul and teaching assistant Mr. Stephen Arnold was like a crash course. Four Modules the ALU, Control, Input-Output and Program Control were identified. Detailed specifications in terms of timing diagrams, opcode and instruction set and block diagrams facilitated us in our maiden attempt in describing hardware using a HDL. (See Sec.2.2 for specifications)

Every module was provided with rigorous test-benches that exercised and helped ratify the functionality of the design.

Every test bench unveiled hidden problems. And lead to successive refinement of the description, to overcome the encountered problems. For instance race conditions in combinational logic that would otherwise be never touched upon in normal coursework were encountered.

Each module was simulated using the test-bench and synthesized – synthesis enforces additional rules that adds to the hardware flavor.

Synthesis with constraints to minimize area was used. Area and timing reports were generated and used as the yardstick in evaluating a design.

2. 0 Modular Hierarchy

2.1 Modularization

A complex system is organized into functional blocks and is inherently structured to promote division of labor. Be it the organization and differentiation of cells into tissues, tissues to organs in the human body, or the organization of the basic transistor switch into caches, registers, memory, both depict the underlying theme of structure and hierarchy, specialization amongst systems into functional blocks or modules.

Modularization and structure is thus the heart of any system. And any tool like the HDL used in creation and description of such a system draws heavily on this underlying concept.

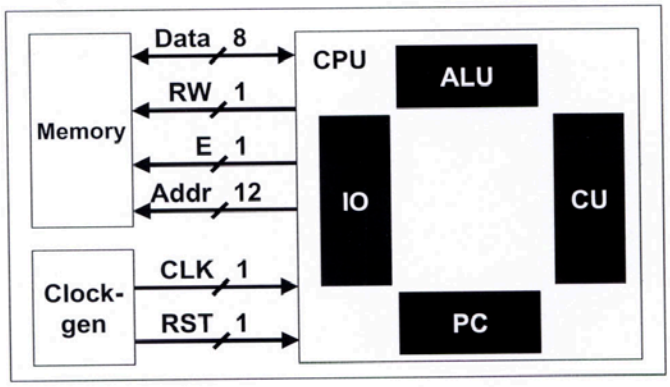
2.2 Specification & Requirements

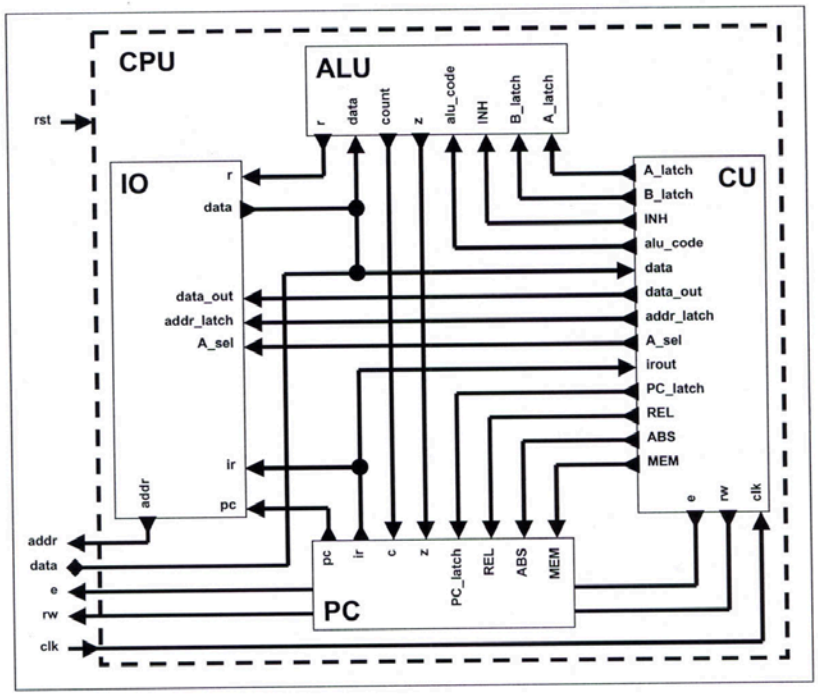
Detailed specifications for the CISC processor was provided in terms of instruction set, timing diagrams, interface diagrams to external modules. Test modules and timing diagrams greatly aided us in understanding the essence and interplay of control and data paths in the implementation of the computer system as a finely orchestrated system.

The following requirements can be extracted from the given specifications:

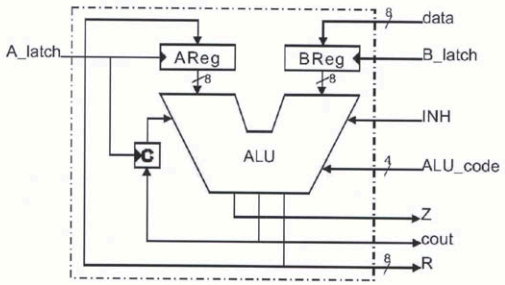
- ✓ The processor should implement arithmetic, logical and shifting operations as per the instruction set.
- ✓ Depending on the instruction the processor should implement instruction *Fetch-Decode-Execute* in 1, 2, 3 cycles.
- ✓ Each cycle is further divided into 4 phases that enable triggering events that ensure setup and hold requirement for a flip-flop and pre-empting asynchronous events that can cause race and hazards in operation.
- ✓ The 8 bit CISC processor should provide suitable interface to the provided memory module. The individual modules should pass the test-benches that check the functionality of the module.

• Figure 1 Specification - Birds View



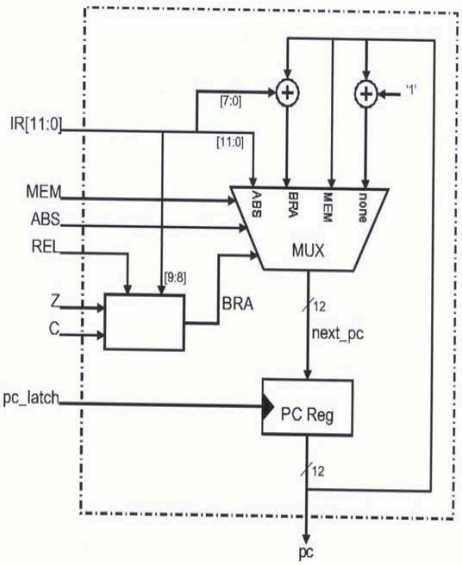


ALU Section



INH	ALU_code	Function (R,cout,Z)
0	X000	ADD
0	X001	SUBTRACT
0	X010	AND
0	X011	OR
0	X100	XOR
0	X101	B or LOAD
0	X110	A or NoOp
0	X111	A or NoOp
1	0100	Rotate Left
1	0101	Rotate Right
1	0110	Arth. Shift Left
1	0111	Arth. Shift Right
1	1000	A + 1
1	1001	A - 1
1	1010	0
1	1011	A or NoOp
1	1100	1's Comp. A
1	1101	2's Comp. A
1	1110	A, C=0
1	1111	A, C=1

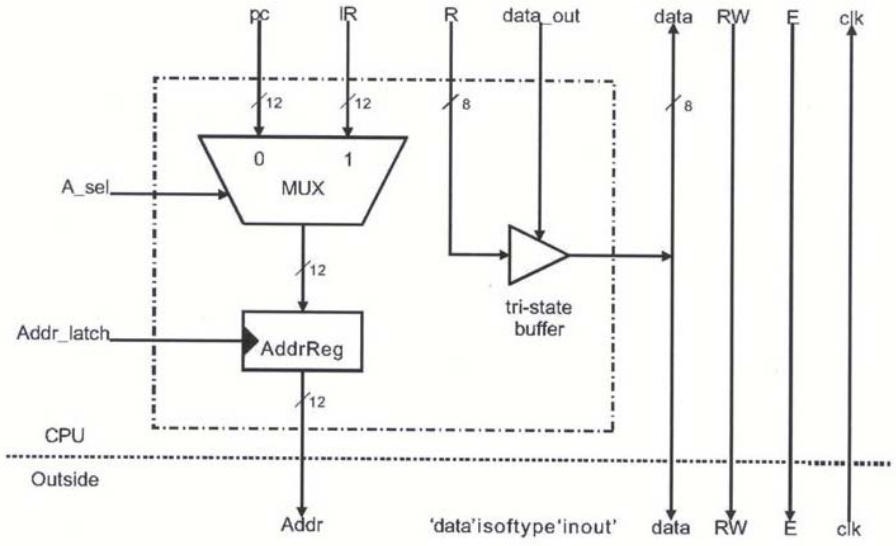
PC Section

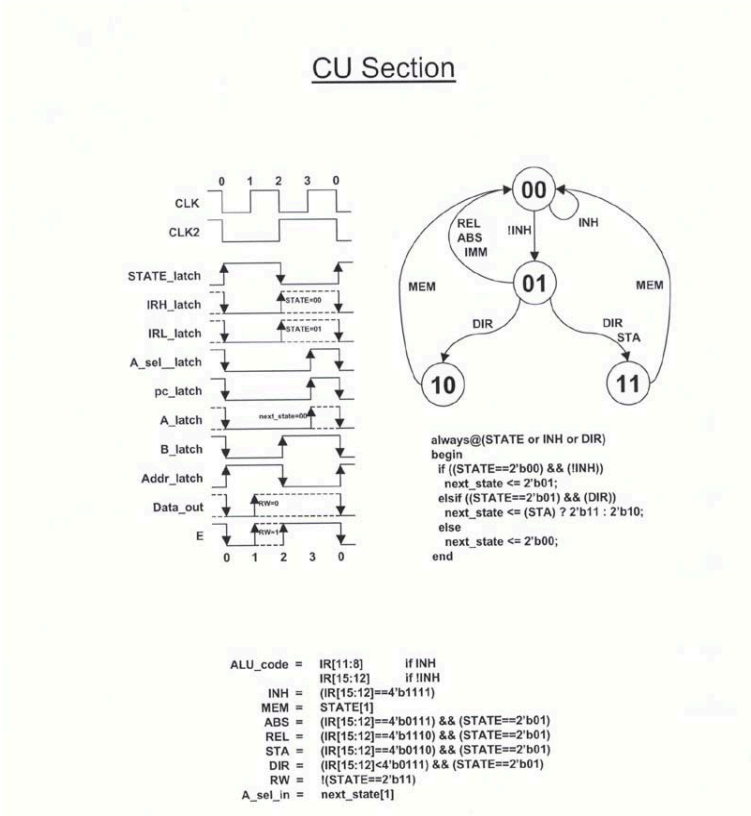


next_pc =		
	pc	if MEM
	BranchAddr(PC,IR)	if BRA
	IR	if ABS
	pc + 1	otherwise

Z	C	IR_9	IR_8	REL	BRA
X	0	0	0	1	1
X	1	0	1	1	1
0	X	1	0	1	1
1	X	1	1	1	1
otherwise					0

IO Section





The ALU Specifications maps the alu_code and type of instruction to the operation.

PC – Program Counter holds the address of next instruction.

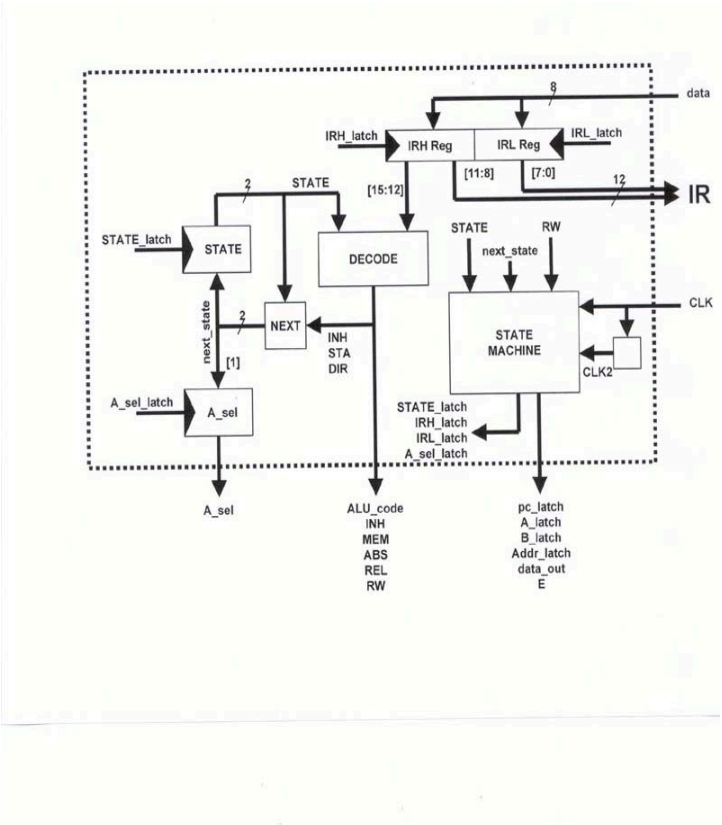
The next instruction address can be just a PC increment if the program execution is sequential. On a branching operation PC is added with the relative offset to calculate the new address. In case of Absolute addressing the PC contents reflects the Instruction Register contents.

Input Output module is the traffic light or police man of the system ensuring smooth flow of data to and from the CPU. It implements a tri-state buffer for multiple drivers driving the same net.

CU – Control Unit. Is responsible for the generation of signals that co-ordinate and synchronize the activity of the CPU.

It generates various latching signals which are active high signals that act as a clock for flip-flops to trigger the data to be latched.

The latches are generated based on the instruction decoded and the state of execution of the instruction.



Instruction Set and Timing

CISC Accumulator Based.
1-3 Cycle Instructions: 4 Phase Cycle. Instructions are 8-bits or 16-bits wide.
Data Bus: 8-bits wide.
Address Bus: 12-bits wide.
Status bits: Z (zero), C (carry)
CPU I/O pins: addr [11:0], data [7:0], rd, wr, clk

Instruction Set

Addressing Modes: Inherent, Immediate, Direct
Branching Modes: Absolute, Relative

Instruction Format:

Instruction	Function	Status Code	Mode	Op Code FEDC	BA98	7654	3210
ADDM	A ⇐ A + (M)	Z, C	DIR	0000	Address		
SUBM	A ⇐ A - (M)	Z, C	DIR	0001			
ANDM	A ⇐ A & (M)	Z	DIR	0010			
ORM	A ⇐ A | (M)	Z	DIR	0011			
XORM	A ⇐ A ^ (M)	Z	DIR	0100			
LDAM	A ⇐ (M)	Z	DIR	0101			
STA	(M) ⇐ A	---	DIR	0110			
JMP	PC ⇐ M	---	ABS	0111			
ADDI	A ⇐ A + K	Z, C	IMM	1000	0000	Data	
SUBI	A ⇐ A - K	Z, C	IMM	1001	0000		
ANDI	A ⇐ A & K	Z	IMM	1010	0000		
ORI	A ⇐ A | K	Z	IMM	1011	0000		
XORI	A ⇐ A ^ K	Z	IMM	1100	0000		
LDAI	A ⇐ K	Z	IMM	1101	0000		
BCC	PC ⇐ PC + K + 2	(C = 0)	REL	1110	0000	Relative Address	
BCS	PC ⇐ PC + K + 2	(C = 1)	REL	1110	0001		
BNE	PC ⇐ PC + K + 2	(Z = 0)	REL	1110	0010		
BEQ	PC ⇐ PC + K + 2	(Z = 1)	REL	1110	0011		
ROL	Rotate Left	Z, C	INH	1111	0100		
ROR	Rotate Right	Z, C	INH	1111	0101		
ASL	Arithmetic Shift Left	Z, C	INH	1111	0110		
ASR	Arithmetic Shift Right	Z, C	INH	1111	0111		
INC	A ⇐ A + 1	Z, C	INH	1111	1000		
DEC	A ⇐ A - 1	Z, C	INH	1111	1001		
CLR	A ⇐ 0	Z	INH	1111	0101		
NOP	---	---	INH	1111	1011		
NOT	A ⇐ (~A)	Z	INH	1111	1100		
NEG	A ⇐ (-A)	Z, C	INH	1111	1101		
CLC	---	C ⇐ 0	INH	1111	1110		
SEC	---	C ⇐ 1	INH	1111	1111		

Instruction set specification

3.0 Verilog Implementation

3.1 Integrated CPU

```
/*************************************************************************/
/*      Module name: my_cpu.v                                            */
/*      Arguments  :                                                     */
/*      Description:                                                     */
/*      Returns    :                                                     */
/*      Author     : K.Narayanan                                         */
/*      Bugs       :                                                     */
/*      Date       : April 29 2002                                       */
/*************************************************************************/
module my_cpu (data,addr,rw,E,clk,rst);

input        clk,rst;
inout  [7:0]   data;
output [11:0]  addr;
output       rw,E;

//reg rw,E;
//wire         clk,rst;
//reg   [7:0]   data;
//reg   [11:0]  addr;

wire pc_l,a_l,b_l,addr_l,data_out,E,inh,mem,abs,rel,rw,a_sel;
wire [3:0] alu_code;
wire [11:0] ir;
wire      zero,cout;
wire [7:0] result;
wire [11:0] pc;

/* instantiating individual modules */

alu_intg alu1
(.z(zero),.cout(cout),.r(result),.data(data),.a_latch(a_l),.b_latch(b_l),.inh(inh),.alu_code(alu_code),.rst(rst));


pc_intg pc1 (.pc(pc),.ir(ir),.mem(mem),.abs(abs),.rel(rel),.z(zero),.c(cout),.pc_latch(pc_l),.rst(rst));


io_intg io1
(.data(data),.addr(addr),.data_out(data_out),.result(result),.ir(ir),.pc(pc),.a_sel(a_sel),.addr_l(addr_l),.rst(rst)
);

cu_intg cu1
(.ir(ir),.pc_l(pc_l),.a_l(a_l),.b_l(b_l),.addr_l(addr_l),.data_out(data_out),.E(E),.alu_code(alu_code),.inh(inh
),.mem(mem),.abs(abs),.rel(rel),.rw(rw),.a_sel(a_sel),.clk(clk),.rst(rst),.data(data));


endmodule
```

3.2 ALU Implementation

```
/*************************************************************************/
/*        Module name: my_alu.v                                          */
/*        Arguments  :                                                   */
/*        Description:                                                   */
/*        Returns    :                                                   */
/*        Author     : K.Narayanan                                       */
/*        Bugs       :                                                   */
/*        Date       : April 29 2002                                     */
/*************************************************************************/
module alu_intg (z,cout,r,data,a_latch,b_latch,inh,alu_code,rst);

input [7:0] data;
input a_latch,b_latch,inh,rst;
input [3:0] alu_code;

output [7:0] r;
output z,cout;
//reg    z,cout;
//reg    [7:0] r;

wire [7:0] a_8,b_8;
wire cin;

a_reg a (.a_8(a_8),.rst(rst),.result(r),.a_latch(a_latch));
b_reg b (.b_8(b_8),.rst(rst),.data(data),.b_latch(b_latch));
carry c (.c(cin),.rst(rst),.cout(cout),.a_latch(a_latch));

my_alu alu1 (.result(r),.zero(z),.cout(cout),.inh(inh),.ALU_code(alu_code),.a_8(a_8),.b_8(b_8),.c(cin));

endmodule
/*************************************************************************/
/*        Module name: a_reg.v                                           */
/*        Arguments  :                                                   */
/*        Description:                                                   */
/*        Returns    :                                                   */
/*        Author     : K.Narayanan                                       */
/*        Bugs       :                                                   */
/*        Date       : April 29 2002                                     */
/*************************************************************************/

module a_reg (a_8,rst,result,a_latch);
input       rst,a_latch;
input  [7:0] result;

output [7:0] a_8;

wire [7:0] result;
reg  [7:0] a_8;

always @(posedge a_latch or posedge rst) begin
if(rst == 1'b1)begin                // reset is active high
  a_8 <= 8'h00;
end
```

```
else begin
//   $display("latching result %h",result);  // latch data as a_latch active
   a_8 <= result;
end
end                             // end always

endmodule
/*********************************************************************/
/*        Module name: b_reg.v                                       */
/*        Arguments  :                                               */
/*        Description:                                               */
/*        Returns    :                                               */
/*        Author     : K.Narayanan                                   */
/*        Bugs       :                                               */
/*        Date       : April 29 2002                                 */
/*********************************************************************/

module b_reg(b_8,rst,data,b_latch);

input rst,b_latch;
input  [7:0] data;

output [7:0] b_8;

wire [7:0] data;
reg  [7:0] b_8;


always @(posedge b_latch or posedge rst ) begin
if(rst == 1'b1 ) begin              // reset is active high
  b_8 <= 8'h00;
end
else begin
  b_8 <= data;                    // latch data as b_latch active

end
end                             // end always

endmodule
/*********************************************************************/
/*        Module name: c_reg.v                                       */
/*        Arguments  :                                               */
/*        Description:                                               */
/*        Returns    :                                               */
/*        Author     : K.Narayanan                                   */
/*        Bugs       :                                               */
/*        Date       : April 29 2002                                 */
/*********************************************************************/

module carry(c,rst,cout,a_latch);

input rst,cout,a_latch;

output  c;

reg     c;
```

```
always @(posedge a_latch or posedge rst ) begin
if(rst== 1'b1) begin
  c <= 1'b0;
end
else begin
  c <= cout;
//end                                   // a_latch check
end

end                                     // end always
endmodule
/*************************************************************************/
/*        Module name: myalu.h                                           */
/*        Arguments  :                                                   */
/*        Description:                                                   */
/*        Returns    :                                                   */
/*        Author     : K.Narayanan                                       */
/*        Bugs       :                                                   */
/*        Date       : April 29 2002                                     */
/*************************************************************************/
/* ALU operation codes */
`define INH_0 1'b0
`define INH_1 1'b1
`define INH_x 1'bx

/* INH_0 */
`define ADD   4'bx000
`define SUB   4'bx001
`define AND   4'bx010
`define OR    4'bx011
`define XOR   4'bx100
`define B_LD  4'bx101
`define A1NOP 4'bx110
`define A2NOP 4'bx111

/* INH_1 */
`define RLFT  4'b0100
`define RRGT  4'b0101
`define AS_LFT 4'b0110
`define AS_RGT 4'b0111
`define INCR  4'b1000
`define DECR  4'b1001
`define ZERO  4'b1010
`define A3NOP 4'b1011
`define ONESC 4'b1100
`define TWOSC 4'b1101
`define A_C_0 4'b1110
`define A_C_1 4'b1111
```

```
/*************************************************************************/
/*        Module name: my_alu.v                                          */
/*        Arguments  :                                                   */
/*        Description:                                                   */
/*        Returns    :                                                   */
/*        Author     : K.Narayanan                                       */
/*        Bugs       :                                                   */
/*        Date       : April 29 2002                                     */
/*************************************************************************/
/*  ALU Module                                                           */
/* operations based on ALU_code and INH                                  */

`include "myalu.h"

module my_alu (result,zero,cout,inh,ALU_code,a_8,b_8,c);


input  [3:0] ALU_code;
input        inh,c;

input  [7:0] a_8,b_8;

output [7:0] result;
output       zero,cout;


reg  [7:0] result;
reg        zero,cout;

wire  [7:0] a_8,b_8;
wire [3:0] ALU_code;


//wire      c,a_latch,b_latch;
// all inputs required in sensitivity list for same operation but diff inputs

always @(inh or ALU_code or a_8 or b_8 or c) begin

//$display ("inh:alu:%h a_8:%h b_8:%h",{inh,ALU_code},a_8,b_8);
casex ({inh,ALU_code})
    /* Addition affects the carry and zero status lines */

{`INH_0,`ADD}: begin
       {result} <= {a_8 + b_8 + c};
end

{`INH_0,`SUB}: begin
       {result} <= {a_8 - b_8 - c };

end

{`INH_0,`AND}:begin
     result <= a_8 & b_8;
end
```

```
{`INH_0,`OR}:begin
    result <= a_8 | b_8;
end
{`INH_0,`XOR}:begin
    result <= a_8 ^ b_8;
end
{`INH_0,`B_LD}:begin
    result <=  b_8;
end
{`INH_0,`A1NOP}:begin
    result <= a_8;
end

{`INH_0,`A2NOP}:begin
    result <= a_8;
end

{`INH_1,`RLFT}:begin
    {result} <= {a_8[6:0],c};
end

{`INH_1,`RRGT}:begin
    {result} <= {c,a_8[7:1]};
end


{`INH_1,`AS_LFT}:begin
    {result} <= {a_8[6:0],1'b0};
end


{`INH_1,`AS_RGT}:begin
    {result} <= {a_8[7],a_8[7:1]};
end


{`INH_1,`INCR}:begin
     {result} <= {a_8 + 1'b1};
end


{`INH_1,`DECR}:begin
     {result} <= {a_8 + 8'hff}  ;

end

{`INH_1,`ZERO}:begin
    result <= 0;
end
{`INH_1,`A3NOP}:begin
    result <= a_8;
end

{`INH_1,`ONESC}:begin
    result <= 8'hff^a_8 ;
end
```

```
{`INH_1,`TWOSC}:begin
    {result} <= {(8'hff^a_8) + 1'b1} ;

end
{`INH_1,`A_C_0}:begin
    result <= a_8 ;
end

{`INH_1,`A_C_1}:begin
    result <= a_8 ;
end

endcase

end // always

always @(result or a_8 or b_8 or c) begin //zero_calc and carry calculation
  if(result == 8'h00) begin
        zero <= 1'b1;
  end
  else begin
     zero <= 1'b0;
  end

casex ({inh,ALU_code})
   /* Addition affects the carry and zero status lines */

{`INH_0,`ADD}: begin
        cout <= ((a_8[7]&b_8[7]) | ((1'b1^result[7])&b_8[7]) | (a_8[7]&(1'b1^result[7])));
end

{`INH_0,`SUB}: begin
        cout <= ((a_8[7]&b_8[7]) | ((1'b1^result[7])&b_8[7]) | (a_8[7]&(1'b1^result[7])));

end

{`INH_0,`AND}:begin
        cout <=c;
end
{`INH_0,`OR}:begin
        cout <=c;
end
{`INH_0,`XOR}:begin
        cout <=c;
end
{`INH_0,`B_LD}:begin
        cout <=c;
end
{`INH_0,`A1NOP}:begin
        cout <=c;
end

{`INH_0,`A2NOP}:begin
        cout <=c;
end
```

```
{`INH_1,`RLFT}:begin
      cout <= a_8[7];
end

{`INH_1,`RRGT}:begin
      cout <= a_8[0];
end

{`INH_1,`AS_LFT}:begin
      cout<= a_8[7];
end

{`INH_1,`AS_RGT}:begin
      cout<= a_8[0];
end

{`INH_1,`INCR}:begin
      cout   <= ((a_8[7])&(a_8[6])&(a_8[5])&(a_8[4])&(a_8[3])&(a_8[2])&(a_8[1])&(a_8[0]));

end

{`INH_1,`DECR}:begin

           cout   <= ((!a_8[7])&(!a_8[6])&(!a_8[5])&(!a_8[4])&(!a_8[3])&(!a_8[2])&(!a_8[1])&(!a_8[0]));

end

{`INH_1,`ZERO}:begin
      cout <= 0;
end
{`INH_1,`A3NOP}:begin
           cout <= c;
end

{`INH_1,`ONESC}:begin
           cout <=c;
end

{`INH_1,`TWOSC}:begin
     cout   <= ((result[7])|(result[6])|(result[5])|(result[4])|(result[3])|(result[2])|(result[1])|(result[0]));
end

{`INH_1,`A_C_0}:begin
           cout <= 1'b0;
end

{`INH_1,`A_C_1}:begin
           cout <= 1'b1;
end

endcase

end

endmodule
```

3.3 Program Counter Implementaion

```
/*********************************************************************/
/*      Module name: pc_intg.v                                       */
/*      Arguments  :                                                 */
/*      Description:                                                 */
/*      Returns    :                                                 */
/*      Author     : K.Narayanan                                     */
/*      Bugs       :                                                 */
/*      Date       : April 29 2002                                   */
/*********************************************************************/

module pc_intg (pc,ir,mem,abs,rel,z,c,pc_latch,rst);
output [11:0] pc;
input  [11:0] ir;
input         mem,abs,rel,z,c,pc_latch,rst;

wire   [11:0] ir,next_pc;
wire          mem,abs,bra;
wire   [11:0] pc;

pc_reg pc_reg1 (.pc(pc),.rst(rst),.next_pc(next_pc),.pc_latch(pc_latch));

bra_gen bra_gen1(.bra(bra),.z(z),.c(c),.rel(rel),.ir9(ir[9]),.ir8(ir[8]));

next_pc_gen next_pc1(.next_pc(next_pc),.pc(pc),.ir(ir),.mem(mem),.abs(abs),.bra(bra));

endmodule

/*********************************************************************/
/*      Module name: bra.h                                           */
/*      Arguments  :                                                 */
/*      Description: Contains defines fro branch generation          */
/*      Returns    :                                                 */
/*      Author     : K.Narayanan                                     */
/*      Bugs       :                                                 */
/*      Date       : April 29 2002                                   */
/*********************************************************************/

// Branch opcode MSW 1110 00XX
// decoding for BRA
// {z,c,ir9,ir8,rel}
// branch only if relative addressing asserted.

`define BCC 5'bx0001    // pc <= pc + 2 if(c==0)
`define BCS 5'bx1011    // pc <= pc + 2 if(c==1)
`define BNE 5'b0x101    // pc <= pc + 2 if(z==0)
`define BEQ 5'b1x111    // pc <= pc + 2 if(z==1)
`include "bra.h"
```

```
/*********************************************************************/
/*      Module name: bra_gen.v                                       */
/*      Arguments  :                                                 */
/*      Description: Contains defines fro branch generation          */
/*      Returns    :                                                 */
/*      Author     : K.Narayanan                                     */
/*      Bugs       :                                                 */
/*      Date       : April 29 2002                                   */
/*********************************************************************/
module bra_gen (bra,z,c,rel,ir9,ir8);
output bra;
reg    bra;
input  z,c,rel,ir9,ir8;

always @(z or c or rel or ir9 or ir8) begin
//$display("bra input:%b",{z,c,ir9,ir8,rel});
casex ({z,c,ir9,ir8,rel})
{`BCC}: begin
//$display("bcc");
  bra <= 1'b1;
end
{`BCS}: begin
//$display("bcs");
  bra <= 1'b1;
end
{`BNE}: begin
//$display("bne");
  bra <= 1'b1;
end
{`BEQ}: begin
//$display("beq");
  bra <= 1'b1;
end
default:begin
//$display("other");
  bra <= 1'b0;
end
endcase
end
endmodule
/*********************************************************************/
/*      Module name: pc_reg.v                                        */
/*      Arguments  :                                                 */
/*      Description:                                                 */
/*      Returns    :                                                 */
/*      Author     : K.Narayanan                                     */
/*      Bugs       :                                                 */
/*      Date       : April 29 2002                                   */
/*********************************************************************/
module pc_reg (pc,rst,next_pc,pc_latch);
input        rst,pc_latch;
input  [11:0] next_pc;

output [11:0] pc;
wire [11:0] next_pc;
```

```
reg  [11:0] pc;

always @(posedge pc_latch or posedge rst) begin
if(rst == 1'b1)begin                    // reset is active high
 pc <= 12'h000;

end
else begin
 pc <= next_pc;

end
end                                     // end always

endmodule
/*************************************************************************/
/*        Module name: next_pc_gen.v                                     */
/*        Arguments  :                                                   */
/*        Description:                                                   */
/*        Returns    :                                                   */
/*        Author     : K.Narayanan                                       */
/*        Bugs       :                                                   */
/*        Date       : April 29 2002                                     */
/*************************************************************************/
module next_pc_gen (next_pc,pc,ir,mem,abs,bra);
input  [11:0] ir,pc;
input         mem,abs,bra;

output [11:0] next_pc;

wire   [11:0] ir,pc;

reg    [11:0] next_pc;

always @(ir or abs or mem or bra or pc) begin

if(mem == 1'b1)     begin
   next_pc <= pc;
end
else if(bra == 1'b1)begin
   next_pc <= {pc+1+ir[7:0]};
end
else if(abs == 1'b1)begin
   next_pc <= {ir[11:0]};
end
else begin
   next_pc <= {pc + 1};
end


end                                     //end always

endmodule
```

3.4 Control Unit Implementation

```
/*************************************************************************/
/*      Module name: cu_intg.v                                           */
/*      Arguments  :                                                     */
/*      Description:                                                     */
/*      Returns    :                                                     */
/*      Author     : K.Narayanan                                         */
/*      Bugs       :                                                     */
/*      Date       : April 29 2002                                       */
/*************************************************************************/

module cu_intg (ir,pc_l,a_l,b_l,addr_l,data_out,E,alu_code,inh,mem,abs,rel,rw,a_sel,clk,rst,data);

input clk,rst;
input [7:0] data;

output pc_l,a_l,b_l,addr_l,data_out,E,inh,mem,abs,rel,rw,a_sel;
output [11:0] ir;
output [3:0] alu_code;


wire pc_l,a_l,b_l,addr_l,data_out,E,inh,mem,abs,rel,rw,a_sel;
wire [11:0] ir;
wire [3:0] alu_code;

wire irl_l,irh_l,state_l,a_sel_l,sta,dir,clkby2 ;
wire [1:0]state,nxt_st;
wire [3:0]irh_msn;
wire clkdivby2_rst;

// MODULE INSTANTIATION

invert_rst i1clkdivby2_rst(.rst_bar(clkdivby2_rst),.rst(rst));

cu_mod cu_mod1
(.pc_l(pc_l),.a_l(a_l),.b_l(b_l),.addr_l(addr_l),.data_out(data_out),.addr_sel_l(a_sel_l),.irh_l(irh_l),.irl_l(irl
_l),.E(E),.state_l(state_l),.rw(rw),.nxt_st(nxt_st),.state(state),.clk(clk),.clkby2(clkby2),.rst(rst));

decode decode1
(.alu_code(alu_code),.inh(inh),.mem(mem),.abs(abs),.rel(rel),.rw(rw),.sta(sta),.dir(dir),.irh_msn(irh_msn),.s
tate(state),.ir(ir));

ir_reg ir1 (.ir(ir),.irh_msn(irh_msn),.rst(rst),.data(data),.irl_latch(irl_l),.irh_latch(irh_l));

next_state nxt_state1 (.nxt_st(nxt_st),.inh(inh),.sta(sta),.dir(dir),.state(state));

state_reg state_reg1 (.state(state),.rst(rst),.nxt_st(nxt_st),.state_l(state_l));

a_sel_reg a_sel_reg1 (.a_sel(a_sel),.rst(rst),.nxt_st1(nxt_st[1]),.a_sel_l(a_sel_l));

clk_div_by2 ck2(.clkby2(clkby2),.clk(clk),.rst(clkdivby2_rst));

//clock_gen ck1 (.clock(clk));
endmodule
```

```
/*************************************************************************/
/*      Module name: ir_reg.v                                            */
/*      Arguments  :                                                     */
/*      Description:                                                     */
/*      Returns    :                                                     */
/*      Author     : K.Narayanan                                         */
/*      Bugs       :                                                     */
/*      Date       : April 29 2002                                       */
/*************************************************************************/
module ir_reg (ir,irh_msn,rst,data,irl_latch,irh_latch);
input       rst,irl_latch,irh_latch;
input  [7:0]  data;
output [11:0] ir;
output [3:0]  irh_msn;

wire [7:0] irl;
wire [3:0] irh_msn,irh_lsn;

assign ir = {irh_lsn,irl};

irl_reg irl1 (.irl(irl),.rst(rst),.data(data),.irl_latch(irl_latch));

irh_reg irh1 (.irh_msn(irh_msn),.irh_lsn(irh_lsn),.rst(rst),.data(data),.irh_latch(irh_latch));

endmodule
/*************************************************************************/
/*      Module name: irh_reg.v                                           */
/*      Arguments  :                                                     */
/*      Description:                                                     */
/*      Returns    :                                                     */
/*      Author     : K.Narayanan                                         */
/*      Bugs       :                                                     */
/*      Date       : April 29 2002                                       */
/*************************************************************************/
module irh_reg (irh_msn,irh_lsn,rst,data,irh_latch);
input       rst,irh_latch;
input  [7:0]  data;

output [3:0]  irh_msn,irh_lsn;

reg    [3:0]  irh_msn,irh_lsn;
wire          rst,irh_latch;

always @(posedge rst or posedge irh_latch)begin
if(rst) begin                          // rst is active high
 irh_msn <= 4'hf;
 irh_lsn <= 4'h0;
end
else begin
 {irh_msn,irh_lsn} <= {data};
end

end
endmodule
```

```
/*************************************************************************/
/*        Module name: irl_reg.v                                         */
/*        Arguments  :                                                   */
/*        Description:                                                   */
/*        Returns    :                                                   */
/*        Author     : K.Narayanan                                       */
/*        Bugs       :                                                   */
/*        Date       : April 29 2002                                     */
/*************************************************************************/
module irl_reg (irl,rst,data,irl_latch);
input        rst,irl_latch;
input  [7:0]  data;

output [7:0]  irl;

reg    [7:0]  irl;
wire         irl_latch;

always @(posedge rst or posedge irl_latch)begin
if(rst) begin                              // rst is active high
 irl <= 8'h00;
end
else begin
 irl <= data;
end

end
endmodule

/*************************************************************************/
/*        Module name: cu_mod.v                                          */
/*        Arguments  :                                                   */
/*        Description:                                                   */
/*        Returns    :                                                   */
/*        Author     : K.Narayanan                                       */
/*        Bugs       :                                                   */
/*        Date       : April 29 2002                                     */
/*************************************************************************/
module cu_mod (pc_l,a_l,b_l,addr_l,data_out,addr_sel_l,irh_l,irl_l,E,state_l,rw,nxt_st,state,clk,clkby2,rst);

input [1:0] nxt_st,state;
input       rw,clk,clkby2,rst;
output      pc_l,a_l,b_l,addr_l,data_out,addr_sel_l,irh_l,irl_l,E,state_l;
reg         pc_l,a_l,b_l,addr_l,data_out,addr_sel_l,irh_l,irl_l,E,state_l;

reg   [1:0]   phase;

always @ (posedge rst)
 phase <= 2'b11;
```

```
always @( clk or  clkby2  ) begin
case ({clk,clkby2})

2'b00: begin
//$display("phase 00");
// generation of latches in {clk,clkby2} 00 state

if(phase == 2'b11) begin
state_l    <= !clkby2;
irh_l      <=  clkby2;
irl_l      <=  clkby2;
addr_sel_l          <=  clk;
pc_l                <=  clk;
a_l                 <=  clk;
b_l                 <=  clkby2;
addr_l              <= !clkby2;
data_out <=  clk;
E                   <=  clk;
phase          <=  2'b10;

end
end

2'b10: begin
//$display("phase 10");
// generation of latches in {clk,clkby2} 10 state
if (phase == 2'b10); begin

//state_l  <= !clkby2;                not valid edge dont do anything
//irh_l    <=  clkby2;                not valid edge dont do anything
//irl_l    <=  clkby2;                not valid edge dont do anything
//addr_sel_l          <= !clk;        not valid edge dont do anything
//pc_l                <= !clk;        not valid edge dont do anything
//a_l                 <= !clk;        not valid edge dont do anything
//b_l                 <=  clkby2;     not valid edge dont do anything
//addr_l  <= !clkby2;                 not valid edge dont do anything

data_out <=  (!rw)?clk:data_out;
E                   <=  (rw)?clk:E;
phase          <= 2'b01;

end
end


2'b01: begin
//$display("phase 01");
// generation of latches in {clk,clkby2} 01 state

if(phase == 2'b01) begin
state_l    <= !clkby2;
irh_l      <=  (state == 2'b00)?clkby2:!clkby2;
irl_l      <=  (state == 2'b01)?clkby2:!clkby2;
```

```
//addr_sel_l      <=  clk;            not valid edge dont do anything
//pc_l            <=  clk;            not valid edge dont do anything
//a_l             <=  clk;            not valid edge dont do anything

b_l               <=  clkby2;
addr_l            <= !clkby2;

//data_out        <=  (data_out)? 1'b1:1'b0;//not valid edge dont do anything

E                 <=  (rw)?!clk:E;
phase             <= 2'b00;
end
end

2'b11: begin
// generation of latches in {clk,clkby2} 11 state
//$display("phase 11");

if(phase == 2'b00) begin
//state_l <= !clkby2;       not valid edge dont do anything
//irh_l   <=  clkby2;       not valid edge dont do anything
//irl_l   <=  clkby2;       not valid edge dont do anything

addr_sel_l        <=  clk;
pc_l              <=  clk;
a_l               <=  (nxt_st==2'b00)?clk:!clk;

//b_l             <=  clkby2;       not valid edge dont do anything
//addr_l <= !clkby2;        not valid edge dont do anything
//data_out        <=  clk;  not valid edge dont do anything
//E               <=  clk;  not valid edge dont do anything
phase             <=  2'b11;
end
end
endcase
end                                     // always
endmodule
/*************************************************************************/
/*      Module name: decode.v                                            */
/*      Arguments  :                                                     */
/*      Description:                                                     */
/*      Returns    :                                                     */
/*      Author     : K.Narayanan                                         */
/*      Bugs       :                                                     */
/*      Date       : April 29 2002                                       */
/*************************************************************************/
module decode (alu_code,inh,mem,abs,rel,rw,sta,dir,irh_msn,state,ir );
input [1:0] state;
input [3:0] irh_msn;
input [11:0] ir;

output [3:0] alu_code;
output       inh,mem,abs,rel,rw,sta,dir;
wire         inh,mem,abs,rel,rw,sta,dir;
wire   [3:0] alu_code;
```

```
// coding for alu_code generate

assign alu_code = (inh)?ir[11:8]:irh_msn;

// Coding to generate control signals

assign dir =  ((irh_msn < 4'b0111)&& (state==2'b01))?1'b1:1'b0;
assign inh =  (irh_msn == 4'b1111)?1'b1:1'b0;
assign mem =  state[1];
assign abs =  ((irh_msn == 4'b0111)&& (state==2'b01))?1'b1:1'b0;
assign rel =  ((irh_msn == 4'b1110)&& (state==2'b01))?1'b1:1'b0;
assign sta =  ((irh_msn == 4'b0110)&& (state==2'b01))?1'b1:1'b0;
assign rw  =  (state == 2'b11)?1'b0:1'b1;

endmodule
/*************************************************************************/
/*      Module name: next_state.v                                        */
/*      Arguments  :                                                     */
/*      Description:                                                     */
/*      Returns    :                                                     */
/*      Author     : K.Narayanan                                         */
/*      Bugs       :                                                     */
/*      Date       : April 29 2002                                       */
/*************************************************************************/

module next_state(nxt_st,inh,sta,dir,state);
input[1:0]          state;
input               inh,sta,dir;
output[1:0]         nxt_st;

reg[1:0]            nxt_st;

always @ (state or inh or sta or dir) begin
if ((state == 2'b00) && (!inh))
          nxt_st <= 2'b01;
else if ((state == 2'b01) && (dir))
      nxt_st <= (sta)?2'b11:2'b10;
else
      nxt_st <= 2'b00;
end
endmodule
/*************************************************************************/
/*      Module name: state_reg.v                                         */
/*      Arguments  :                                                     */
/*      Description:                                                     */
/*      Returns    :                                                     */
/*      Author     : K.Narayanan                                         */
/*      Bugs       :                                                     */
/*      Date       : April 29 2002                                       */
/*************************************************************************/
module state_reg(state,rst,nxt_st,state_l);
input               rst,state_l;
input  [1:0]        nxt_st;
```

```
output [1:0]         state;
reg    [1:0]         state;

always @(posedge rst or posedge state_l)begin
if(rst== 1'b1) begin
//$display("reset");
//rst active clear register
state <= 2'b00;
end // if end
else
state <= nxt_st;

end // end always
endmodule
/*************************************************************************/
/*        Module name: a_sel_reg.v                                       */
/*        Arguments  :                                                   */
/*        Description:                                                   */
/*        Returns    :                                                   */
/*        Author     : K.Narayanan                                       */
/*        Bugs       :                                                   */
/*        Date       : April 29 2002                                     */
/*************************************************************************/

module a_sel_reg (a_sel,rst,nxt_st1,a_sel_l);

input rst,nxt_st1,a_sel_l;
output  a_sel;
reg     a_sel;

always @(posedge rst or posedge a_sel_l )begin

if(rst== 1'b1) begin
 a_sel <= 1'b0;
end
else begin
 a_sel <= nxt_st1;
end

end                                             // end always
endmodule
/*************************************************************************/
/*        Module name: clk_by_2.v                                        */
/*        Arguments  :                                                   */
/*        Description:                                                   */
/*        Returns    :                                                   */
/*        Author     : K.Narayanan                                       */
/*        Bugs       :                                                   */
/*        Date       : April 29 2002                                     */
/*************************************************************************/

module clk_div_by2(clkby2,clk,rst);
input clk,rst;
output clkby2;

reg clkby2;
```

```
always @(negedge clk or  negedge rst)begin
if(!rst)
        clkby2 <= 1'b0;
else
        clkby2 <= !clkby2;
end


endmodule
/*****************************************************************************/
/*      Module name: invert_rst.v                                            */
/*      Arguments  :                                                         */
/*      Description:                                                         */
/*      Returns    :                                                         */
/*      Author     : K.Narayanan                                             */
/*      Bugs       :                                                         */
/*      Date       : April 29 2002                                           */
/*****************************************************************************/
module invert_rst(rst_bar, rst);

input rst;
output rst_bar;

wire rst,rst_bar;

assign rst_bar = !rst;

endmodule
```

3.5 Input Output Implementation

```
/*****************************************************************************/
/*      Module name: io_intg.v                                               */
/*      Arguments  :                                                         */
/*      Description:                                                         */
/*      Returns    :                                                         */
/*      Author     : K.Narayanan                                             */
/*      Bugs       :                                                         */
/*      Date       : April 29 2002                                           */
/*****************************************************************************/
module io_intg (data,addr,data_out,result,ir,pc,a_sel,addr_l,rst);
input [11:0] ir,pc;
input [7:0]  result;
input        a_sel,addr_l,data_out,rst;

output [7:0]  data;
output [11:0] addr;

wire [11:0]  addr_in;

io_tri   tri_1  (.data(data),.result(result), .data_out(data_out));
addr_reg addr1  (.addr(addr),.rst(rst),.addr_in(addr_in),.addr_l(addr_l));
a_select a_sel1 (.addr(addr_in),.pc(pc),.ir(ir),.a_sel(a_sel));

endmodule
```

```
/*************************************************************************/
/*      Module name: io_areg.v                                           */
/*      Arguments  :                                                     */
/*      Description:                                                     */
/*      Returns    :                                                     */
/*      Author     : K.Narayanan                                         */
/*      Bugs       :                                                     */
/*      Date       : April 29 2002                                       */
/*************************************************************************/
module addr_reg (addr,rst,addr_in,addr_l);
input       rst,addr_l;
input  [11:0] addr_in;
output [11:0] addr;


reg    [11:0] addr;

always @(posedge addr_l or posedge rst) begin
if(rst) begin                       // rst is active high
 addr  <= 16'h0000;
end
else begin
 addr <= addr_in;
end

end                                 // end always
endmodule

/*************************************************************************/
/*      Module name: io_asel.v                                           */
/*      Arguments  :                                                     */
/*      Description:                                                     */
/*      Returns    :                                                     */
/*      Author     : K.Narayanan                                         */
/*      Bugs       :                                                     */
/*      Date       : April 29 2002                                       */
/*************************************************************************/
module a_select (addr,pc,ir,a_sel);

input a_sel;
input [11:0] ir,pc;

output [11:0] addr;

wire [11:0] ir,pc,addr;

assign addr = (a_sel)? ir:pc;

endmodule
```

```
/*********************************************************************/
/*        Module name: io_tri.v                                      */
/*        Arguments  :                                               */
/*        Description:                                               */
/*        Returns    :                                               */
/*        Author     : K.Narayanan                                   */
/*        Bugs       :                                               */
/*        Date       : April 29 2002                                 */
/*********************************************************************/
module io_tri (data,result, data_out);
output [7:0] data;
input  [7:0] result;
input        data_out;

wire   [7:0] data;

assign data = (data_out)? result:8'bzzzz_zzzz;

endmodule
```

4.0 Simulation Results

4.1 ALU Simulation

Host command: /apps/cadence/LDV32/tools/verilog/bin/verilog.exe
Command arguments:
+access+r
alu_test.v
a_reg.v
b_reg.v
c_reg.v
my_alu.v
alu_intg.v

VERILOG-XL 3.20.s004 log file created Apr 28, 2002 15:02:17
VERILOG-XL 3.20.s004 Apr 28, 2002 15:02:17

```
For technical assistance please contact the Cadence Response Center at
1-877-CDS-4911 or send email to support@cadence.com
For more information on Cadence's Verilog-XL product line send email to
talkv@cadence.com

Compiling source file "alu_test.v"
Compiling source file "a_reg.v"
Compiling source file "b_reg.v"
Compiling source file "c_reg.v"
Compiling source file "my_alu.v"
Compiling included source file "myalu.h"
Continuing compilation of source file "my_alu.v"
Compiling source file "alu_intg.v"
Highest level modules:
alu_test
INH=1 ALU_code=b,NOP B=ff,  Expect Z=1 C=0 R=00,  Measured Z=1 C=0 R=00
INH=1 ALU_code=8,INC B=ee,  Expect Z=0 C=0 R=01,  Measured Z=0 C=0 R=01
INH=1 ALU_code=8,INC B=dd,  Expect Z=0 C=0 R=02,  Measured Z=0 C=0 R=02
INH=1 ALU_code=5,ROR B=cc,  Expect Z=0 C=0 R=01,  Measured Z=0 C=0 R=01
INH=1 ALU_code=5,ROR B=cc,  Expect Z=1 C=1 R=00,  Measured Z=1 C=1 R=00
INH=1 ALU_code=5,ROR B=cc,  Expect Z=0 C=0 R=80,  Measured Z=0 C=0 R=80
INH=1 ALU_code=9,DEC B=bb,  Expect Z=0 C=0 R=7f,  Measured Z=0 C=0 R=7f
INH=1 ALU_code=9,DEC B=bb,  Expect Z=0 C=0 R=7e,  Measured Z=0 C=0 R=7e
INH=1 ALU_code=d,NEG B=aa,  Expect Z=0 C=1 R=82,  Measured Z=0 C=1 R=82
INH=1 ALU_code=c,NOT B=99,  Expect Z=0 C=1 R=7d,  Measured Z=0 C=1 R=7d
INH=1 ALU_code=6,ASL B=88,  Expect Z=0 C=0 R=fa,  Measured Z=0 C=0 R=fa
INH=1 ALU_code=7,ASR B=77,  Expect Z=0 C=0 R=fd,  Measured Z=0 C=0 R=fd
INH=1 ALU_code=7,ASR B=77,  Expect Z=0 C=1 R=fe,  Measured Z=0 C=1 R=fe
INH=1 ALU_code=c,NOT B=66,  Expect Z=0 C=1 R=01,  Measured Z=0 C=1 R=01
INH=1 ALU_code=4,ROL B=55,  Expect Z=0 C=0 R=03,  Measured Z=0 C=0 R=03
INH=1 ALU_code=f,SEC B=44,  Expect Z=0 C=1 R=03,  Measured Z=0 C=1 R=03
INH=1 ALU_code=4,ROL B=33,  Expect Z=0 C=0 R=07,  Measured Z=0 C=0 R=07
INH=1 ALU_code=5,ROR B=22,  Expect Z=0 C=1 R=03,  Measured Z=0 C=1 R=03
INH=1 ALU_code=e,CLC B=11,  Expect Z=0 C=0 R=03,  Measured Z=0 C=0 R=03
INH=0 ALU_code=7,NOP B=ab,  Expect Z=0 C=0 R=03,  Measured Z=0 C=0 R=03
INH=0 ALU_code=0,ADD B=21,  Expect Z=0 C=0 R=24,  Measured Z=0 C=0 R=24
INH=0 ALU_code=c,XOR B=ff,  Expect Z=0 C=0 R=db,  Measured Z=0 C=0 R=db
INH=0 ALU_code=8,ADD B=27,  Expect Z=0 C=1 R=02,  Measured Z=0 C=1 R=02
INH=0 ALU_code=9,SUB B=00,  Expect Z=0 C=0 R=01,  Measured Z=0 C=0 R=01
INH=0 ALU_code=3,OR  B=70,  Expect Z=0 C=0 R=71,  Measured Z=0 C=0 R=71
INH=0 ALU_code=2,AND B=40,  Expect Z=0 C=0 R=40,  Measured Z=0 C=0 R=40
INH=0 ALU_code=e,NOP B=aa,  Expect Z=0 C=0 R=40,  Measured Z=0 C=0 R=40
INH=0 ALU_code=5,LDA B=fe,  Expect Z=0 C=0 R=fe,  Measured Z=0 C=0 R=fe
INH=0 ALU_code=0,ADD B=02,  Expect Z=1 C=1 R=00,  Measured Z=1 C=1 R=00
-- Simulation End
hello finally there
L56 "alu_test.v": $finish at simulation time 1885
```

4.2 Simulation of Program Counter

```
0 simulation events (use +profile or +listcounts option to count)
CPU time: 0.1 secs to compile + 0.0 secs to link + 0.1 secs in simulation
End of VERILOG-XL 3.20.s004   Apr 28, 2002  15:02:23
Host command: /apps/cadence/LDV32/tools/verilog/bin/verilog.exe
Command arguments:
   +access+r
   bra_gen.v
   next_pc_gen.v
   pc_reg.v
   pc_intg.v
   clock_gen.v
   pc_test.v

VERILOG-XL 3.20.s004 log file created Apr 28, 2002  15:08:39
VERILOG-XL 3.20.s004   Apr 28, 2002  15:08:39
Copyright (c) 1995 Cadence Design Systems, Inc.  All Rights Reserved.
Unpublished -- rights reserved under the copyright laws of the United States.
Copyright (c) 1995 UNIX Systems Laboratories, Inc.  Reproduced with Permission.
THIS SOFTWARE AND ON-LINE DOCUMENTATION CONTAIN CONFIDENTIAL INFORMATION
AND TRADE SECRETS OF CADENCE DESIGN SYSTEMS, INC.  USE, DISCLOSURE, OR
REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF
CADENCE DESIGN SYSTEMS, INC.
RESTRICTED RIGHTS LEGEND
Use, duplication, or disclosure by the Government is subject to
restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in
Technical Data and Computer Software clause at DFARS 252.227-7013 or
subparagraphs (c)(1) and (2) of Commercial Computer Software -- Restricted
Rights at 48 CFR 52.227-19, as applicable.
            Cadence Design Systems, Inc.
            555 River Oaks Parkway
            San Jose, California  95134
For technical assistance please contact the Cadence Response Center at
1-877-CDS-4911 or send email to support@cadence.com

For more information on Cadence's Verilog-XL product line send email to
talkv@cadence.com

Compiling source file "bra_gen.v"
Compiling included source file "bra.h"
Continuing compilation of source file "bra_gen.v"
Compiling source file "next_pc_gen.v"
Compiling source file "pc_reg.v"
Compiling source file "pc_intg.v"
Compiling source file "clock_gen.v"
Compiling source file "pc_test.v"
Highest level modules:
clock_gen
pc_test

IR=xxx  ZC=xx  MEM,ABS,REL=xxx  MUX= +1,  Expect PC=000,  Measured PC=000
IR=088  ZC=00  MEM,ABS,REL=000  MUX= +1,  Expect PC=001,  Measured PC=001
IR=088  ZC=01  MEM,ABS,REL=000  MUX= +1,  Expect PC=002,  Measured PC=002
IR=199  ZC=00  MEM,ABS,REL=000  MUX= +1,  Expect PC=003,  Measured PC=003
```

```
IR=199  ZC=01  MEM,ABS,REL=000  MUX= +1,  Expect PC=004,  Measured PC=004
IR=277  ZC=00  MEM,ABS,REL=000  MUX= +1,  Expect PC=005,  Measured PC=005
IR=277  ZC=01  MEM,ABS,REL=000  MUX= +1,  Expect PC=006,  Measured PC=006
IR=366  ZC=00  MEM,ABS,REL=000  MUX= +1,  Expect PC=007,  Measured PC=007
IR=366  ZC=01  MEM,ABS,REL=000  MUX= +1,  Expect PC=008,  Measured PC=008
IR=088  ZC=10  MEM,ABS,REL=000  MUX= +1,  Expect PC=009,  Measured PC=009
IR=088  ZC=11  MEM,ABS,REL=000  MUX= +1,  Expect PC=00a,  Measured PC=00a
IR=199  ZC=10  MEM,ABS,REL=000  MUX= +1,  Expect PC=00b,  Measured PC=00b
IR=199  ZC=11  MEM,ABS,REL=000  MUX= +1,  Expect PC=00c,  Measured PC=00c
IR=277  ZC=10  MEM,ABS,REL=000  MUX= +1,  Expect PC=00d,  Measured PC=00d
IR=277  ZC=11  MEM,ABS,REL=000  MUX= +1,  Expect PC=00e,  Measured PC=00e
IR=366  ZC=10  MEM,ABS,REL=000  MUX= +1,  Expect PC=00f,  Measured PC=00f
IR=366  ZC=11  MEM,ABS,REL=000  MUX= +1,  Expect PC=010,  Measured PC=010
IR=088  ZC=00  MEM,ABS,REL=100  MUX=MEM,  Expect PC=010,  Measured PC=010
IR=088  ZC=01  MEM,ABS,REL=100  MUX=MEM,  Expect PC=010,  Measured PC=010
IR=199  ZC=00  MEM,ABS,REL=100  MUX=MEM,  Expect PC=010,  Measured PC=010
IR=199  ZC=01  MEM,ABS,REL=100  MUX=MEM,  Expect PC=010,  Measured PC=010
IR=277  ZC=00  MEM,ABS,REL=100  MUX=MEM,  Expect PC=010,  Measured PC=010
IR=277  ZC=01  MEM,ABS,REL=100  MUX=MEM,  Expect PC=010,  Measured PC=010
IR=366  ZC=00  MEM,ABS,REL=100  MUX=MEM,  Expect PC=010,  Measured PC=010
IR=366  ZC=01  MEM,ABS,REL=100  MUX=MEM,  Expect PC=010,  Measured PC=010
IR=088  ZC=10  MEM,ABS,REL=100  MUX=MEM,  Expect PC=010,  Measured PC=010
IR=088  ZC=11  MEM,ABS,REL=100  MUX=MEM,  Expect PC=010,  Measured PC=010
IR=199  ZC=10  MEM,ABS,REL=100  MUX=MEM,  Expect PC=010,  Measured PC=010
IR=199  ZC=11  MEM,ABS,REL=100  MUX=MEM,  Expect PC=010,  Measured PC=010
IR=277  ZC=10  MEM,ABS,REL=100  MUX=MEM,  Expect PC=010,  Measured PC=010
IR=277  ZC=11  MEM,ABS,REL=100  MUX=MEM,  Expect PC=010,  Measured PC=010
IR=366  ZC=10  MEM,ABS,REL=100  MUX=MEM,  Expect PC=010,  Measured PC=010
IR=366  ZC=11  MEM,ABS,REL=100  MUX=MEM,  Expect PC=010,  Measured PC=010
IR=123  ZC=00  MEM,ABS,REL=010  MUX=ABS,  Expect PC=123,  Measured PC=123
IR=103  ZC=00  MEM,ABS,REL=010  MUX=ABS,  Expect PC=103,  Measured PC=103
IR=002  ZC=00  MEM,ABS,REL=001  MUX=BRA,  Expect PC=106,  Measured PC=106
IR=110  ZC=01  MEM,ABS,REL=001  MUX=BRA,  Expect PC=117,  Measured PC=117
IR=002  ZC=01  MEM,ABS,REL=001  MUX=+1 ,  Expect PC=118,  Measured PC=118
IR=010  ZC=11  MEM,ABS,REL=001  MUX=+1 ,  Expect PC=119,  Measured PC=119
IR=111  ZC=10  MEM,ABS,REL=001  MUX=+1 ,  Expect PC=11a,  Measured PC=11a
IR=102  ZC=00  MEM,ABS,REL=001  MUX=+1 ,  Expect PC=11b,  Measured PC=11b
IR=304  ZC=10  MEM,ABS,REL=001  MUX=BRA,  Expect PC=120,  Measured PC=120
IR=210  ZC=00  MEM,ABS,REL=001  MUX=BRA,  Expect PC=131,  Measured PC=131
IR=210  ZC=10  MEM,ABS,REL=001  MUX=+1 ,  Expect PC=132,  Measured PC=132
IR=312  ZC=00  MEM,ABS,REL=001  MUX=+1 ,  Expect PC=133,  Measured PC=133
IR=202  ZC=11  MEM,ABS,REL=001  MUX=+1 ,  Expect PC=134,  Measured PC=134
IR=311  ZC=01  MEM,ABS,REL=001  MUX=+1 ,  Expect PC=135,  Measured PC=135
IR=802  ZC=00  MEM,ABS,REL=001  MUX=BRA,  Expect PC=138,  Measured PC=138
IR=402  ZC=01  MEM,ABS,REL=001  MUX=+1 ,  Expect PC=139,  Measured PC=139
IR=484  ZC=10  MEM,ABS,REL=001  MUX=BRA,  Expect PC=1be,  Measured PC=1be
IR=100  ZC=11  MEM,ABS,REL=010  MUX=ABS,  Expect PC=100,  Measured PC=100
-- Simulation End
L59 "pc_test.v": $finish at simulation time 2020
```

4.3 Control Unit Simulation

```
0 simulation events (use +profile or +listcounts option to count)
CPU time: 0.1 secs to compile + 0.0 secs to link + 0.1 secs in simulation
End of VERILOG-XL 3.20.s004   Apr 28, 2002  15:08:40
Host command: /apps/cadence/LDV32/tools/verilog/bin/verilog.exe
Command arguments:
   +access+r
   invert_rst.v
   a_sel_reg.v
   clkby2.v
   cu_mod.v
   decode.v
   irh_reg.v
   irl_reg.v
   ir_reg.v
   next_state.v
   state_reg.v
   cu_intg.v
   cu_test.v

VERILOG-XL 3.20.s004 log file created Apr 28, 2002  15:50:42
VERILOG-XL 3.20.s004   Apr 28, 2002  15:50:42
Copyright (c) 1995 Cadence Design Systems, Inc.  All Rights Reserved.
Unpublished -- rights reserved under the copyright laws of the United States.
Copyright (c) 1995 UNIX Systems Laboratories, Inc.  Reproduced with Permission.
THIS SOFTWARE AND ON-LINE DOCUMENTATION CONTAIN CONFIDENTIAL INFORMATION
AND TRADE SECRETS OF CADENCE DESIGN SYSTEMS, INC.  USE, DISCLOSURE, OR
REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF
CADENCE DESIGN SYSTEMS, INC.
RESTRICTED RIGHTS LEGEND
Use, duplication, or disclosure by the Government is subject to
restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in
Technical Data and Computer Software clause at DFARS 252.227-7013 or
subparagraphs (c)(1) and (2) of Commercial Computer Software -- Restricted
Rights at 48 CFR 52.227-19, as applicable.
            Cadence Design Systems, Inc.
            555 River Oaks Parkway
            San Jose, California  95134
For technical assistance please contact the Cadence Response Center at
1-877-CDS-4911 or send email to support@cadence.com
For more information on Cadence's Verilog-XL product line send email to
talkv@cadence.com

Compiling source file "invert_rst.v"
Compiling source file "a_sel_reg.v"
Compiling source file "clkby2.v"
Compiling source file "cu_mod.v"
Compiling source file "decode.v"
Compiling source file "irh_reg.v"
Compiling source file "irl_reg.v"
Compiling source file "ir_reg.v"
Compiling source file "next_state.v"
Compiling source file "state_reg.v"
Compiling source file "cu_intg.v"
```

```
Compiling source file "cu_test.v"
Highest level modules:
cu_test

Cycle #         0
Cycle #         1
Cycle #         2
Cycle #         3
Cycle #         4
Cycle #         5
Cycle #         6
Cycle #         7
Cycle #         8
Cycle #         9
Cycle #        10
Cycle #        11
Cycle #        12
Cycle #        13
Cycle #        14
Cycle #        15
Simulation Done.
L51 "cu_test.v": $finish at simulation time 1600
0 simulation events (use +profile or +listcounts option to count)
CPU time: 0.1 secs to compile + 0.0 secs to link + 0.1 secs in simulation
End of VERILOG-XL 3.20.s004   Apr 28, 2002  15:50:43
```

4.4 Simulation of CPU – CPU_test

```
Host command: /apps/cadence/LDV32/tools/verilog/bin/verilog.exe
Command arguments:
   +access+r
   myalu.h
   a_reg.v
   b_reg.v
   c_reg.v
   my_alu.v
   alu_intg.v
   bra.h
   bra_gen.v
   next_pc_gen.v
   pc_reg.v
   pc_intg.v
   io_areg.v
   io_asel.v
   io_tri.v
   io_intg.v
   a_sel_reg.v
   clkby2.v
   cu_mod.v
   decode.v
   invert_rst.v
   irh_reg.v
   irl_reg.v
   ir_reg.v
   next_state.v
```

```
state_reg.v
cu_intg.v
my_cpu.v
clock_gen.v
memory.v
cpu_test.v

VERILOG-XL 3.20.s004 log file created Apr 28, 2002  16:02:48
VERILOG-XL 3.20.s004   Apr 28, 2002  16:02:48
Copyright (c) 1995 Cadence Design Systems, Inc.  All Rights Reserved.
Unpublished -- rights reserved under the copyright laws of the United States.
Copyright (c) 1995 UNIX Systems Laboratories, Inc.  Reproduced with Permission.
THIS SOFTWARE AND ON-LINE DOCUMENTATION CONTAIN CONFIDENTIAL INFORMATION
AND TRADE SECRETS OF CADENCE DESIGN SYSTEMS, INC.  USE, DISCLOSURE, OR
REPRODUCTION IS PROHIBITED WITHOUT THE PRIOR EXPRESS WRITTEN PERMISSION OF
CADENCE DESIGN SYSTEMS, INC.
RESTRICTED RIGHTS LEGEND
Use, duplication, or disclosure by the Government is subject to
restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in
Technical Data and Computer Software clause at DFARS 252.227-7013 or
subparagraphs (c)(1) and (2) of Commercial Computer Software -- Restricted
Rights at 48 CFR 52.227-19, as applicable.
            Cadence Design Systems, Inc.
            555 River Oaks Parkway
            San Jose, California  95134
For technical assistance please contact the Cadence Response Center at
1-877-CDS-4911 or send email to support@cadence.com
For more information on Cadence's Verilog-XL product line send email to
talkv@cadence.com

Compiling source file "myalu.h"
Compiling source file "a_reg.v"
Compiling source file "b_reg.v"
Compiling source file "c_reg.v"
Compiling source file "my_alu.v"
Compiling included source file "myalu.h"
Continuing compilation of source file "my_alu.v"
Compiling source file "alu_intg.v"
Compiling source file "bra.h"
Compiling source file "bra_gen.v"
Compiling included source file "bra.h"
Continuing compilation of source file "bra_gen.v"
Compiling source file "next_pc_gen.v"
Compiling source file "pc_reg.v"
Compiling source file "pc_intg.v"
Compiling source file "io_areg.v"
Compiling source file "io_asel.v"
Compiling source file "io_tri.v"
Compiling source file "io_intg.v"
Compiling source file "a_sel_reg.v"
Compiling source file "clkby2.v"
Compiling source file "cu_mod.v"
Compiling source file "decode.v"
Compiling source file "invert_rst.v"
Compiling source file "irh_reg.v"
Compiling source file "irl_reg.v"
```

```
Compiling source file "ir_reg.v"
Compiling source file "next_state.v"
Compiling source file "state_reg.v"
Compiling source file "cu_intg.v"
Compiling source file "my_cpu.v"
Compiling source file "clock_gen.v"
Compiling source file "memory.v"
Compiling source file "cpu_test.v"
Highest level modules:
cpu_test

MEMORY: LOADING FILE cpu_test.hex
Cycle:              1  Addr:000  Data:fa  RW:1
Cycle:              3  Addr:001  Data:fe  RW:1
Cycle:              5  Addr:002  Data:f8  RW:1
Cycle:              7  Addr:003  Data:80  RW:1
Cycle:              9  Addr:004  Data:02  RW:1
Cycle:             11  Addr:005  Data:00  RW:1
Cycle:             13  Addr:006  Data:0c  RW:1
Cycle:             15  Addr:00c  Data:03  RW:1
Cycle:             17  Addr:007  Data:e2  RW:1
Cycle:             19  Addr:008  Data:01  RW:1
L35 "cpu_test.v": $finish at simulation time 2000
0 simulation events (use +profile or +listcounts option to count)
CPU time: 0.1 secs to compile + 0.1 secs to link + 0.0 secs in simulation
End of VERILOG-XL 3.20.s004   Apr 28, 2002  16:02:49
```

5.0 Synthesized Area of Chip, Modules with constraint on optimize area

5.1 Integrated CPU Area

```
****************************************
Report : area
Design : my_cpu
Version: 2000.11
Date   : Sun Apr 28 16:05:14 2002
****************************************

Library(s) Used:

    lsi_10k (File: /apps/synopsys/2000.11/libraries/syn/lsi_10k.db)

Number of ports:              24
Number of nets:               72
Number of cells:               4
Number of references:          4

Combinational area:       755.000000
Noncombinational area:    690.000000
Net Interconnect area:    undefined  (No wire load specified)

Total cell area:         1445.000000
```

5.2 Integrated ALU module area

```
****************************************
Report : area
Design : alu_intg
Version: 2000.11
Date   : Sun Apr 28 15:06:05 2002
****************************************

Library(s) Used:

    lsi_10k (File: /apps/synopsys/2000.11/libraries/syn/lsi_10k.db)

Number of ports:              26
Number of nets:               43
Number of cells:               4
Number of references:          4

Combinational area:       422.000000
Noncombinational area:    198.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:          620.000000
```

5.3 Integrated Program Counter module area

```
****************************************
Report : area
Design : pc_intg
Version: 2000.11
Date   : Sun Apr 28 15:09:15 2002
****************************************

Library(s) Used:

    lsi_10k (File: /apps/synopsys/2000.11/libraries/syn/lsi_10k.db)

Number of ports:              31
Number of nets:               44
Number of cells:               3
Number of references:          3

Combinational area:       187.000000
Noncombinational area:    108.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:          295.000000
```

5.4 Input-Output module area

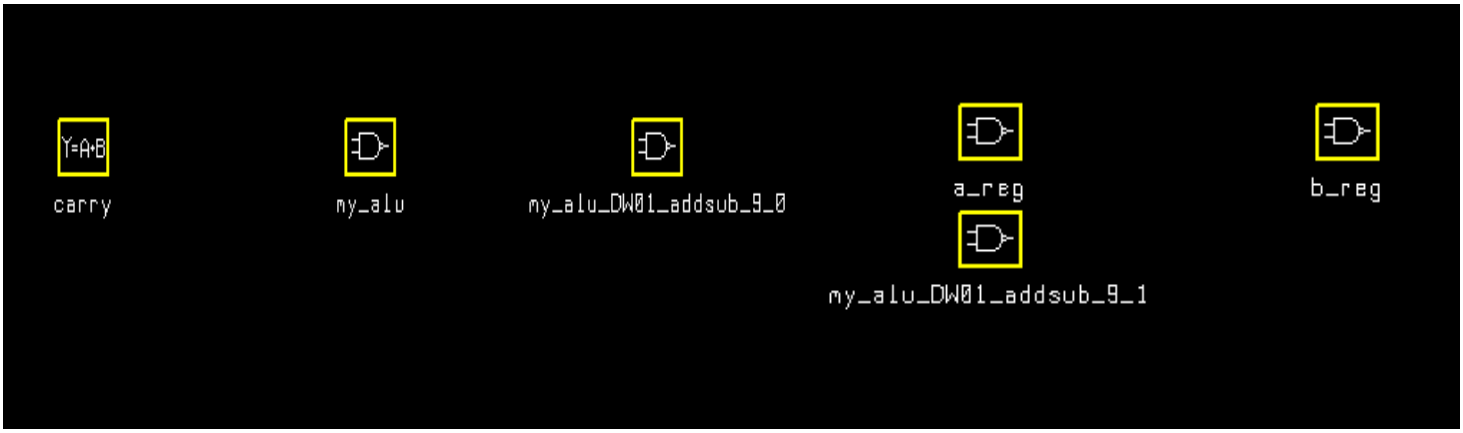
```
****************************************
Report : area
Design : io_intg
Version: 2000.11
Date   : Sun Apr 28 15:16:49 2002
****************************************

Library(s) Used:

    lsi_10k (File: /apps/synopsys/2000.11/libraries/syn/lsi_10k.db)

Number of ports:            57
Number of nets:             69
Number of cells:             3
Number of references:        3

Combinational area:       49.000000
Noncombinational area:    188.000000
Net Interconnect area:    undefined  (No wire load specified)

Total cell area:          237.000000
```

5.5 Control Unit Module Area

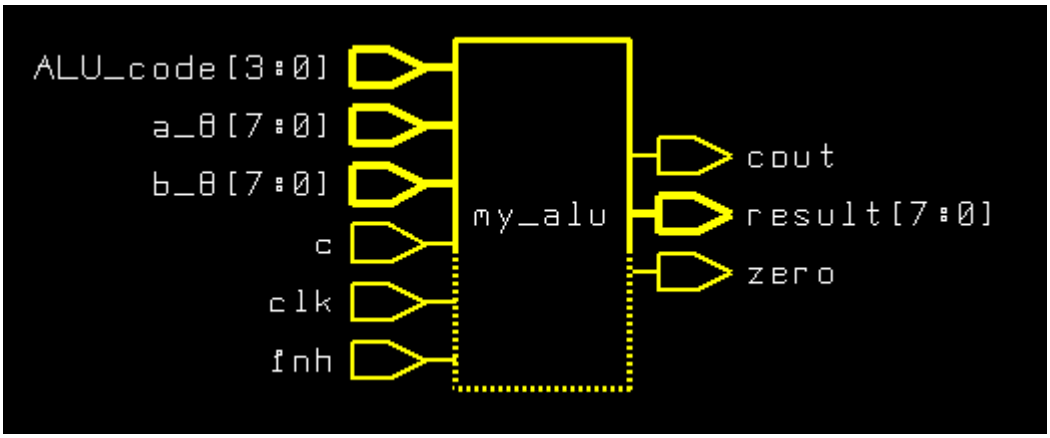
```
****************************************
Report : area
Design : cu_intg
Version: 2000.11
Date   : Sun Apr 28 15:54:02 2002
****************************************

Library(s) Used:

    lsi_10k (File: /apps/synopsys/2000.11/libraries/syn/lsi_10k.db)

Number of ports:            38
Number of nets:             54
Number of cells:             8
Number of references:        8

Combinational area:       88.000000
Noncombinational area:    252.000000
Net Interconnect area:    undefined  (No wire load specified)

Total cell area:          340.000000
```

6.0 Synthesis Snapshots & Wave Form Snapshots

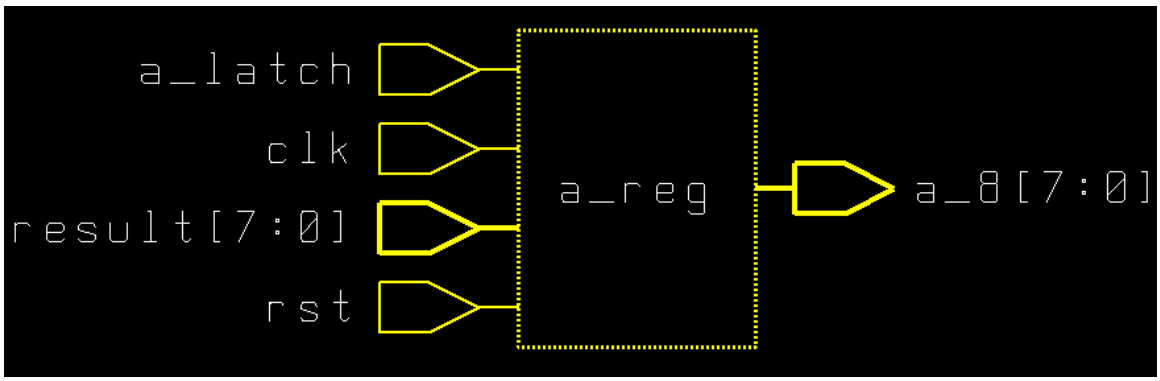
6.1.1 ALU Synthesized



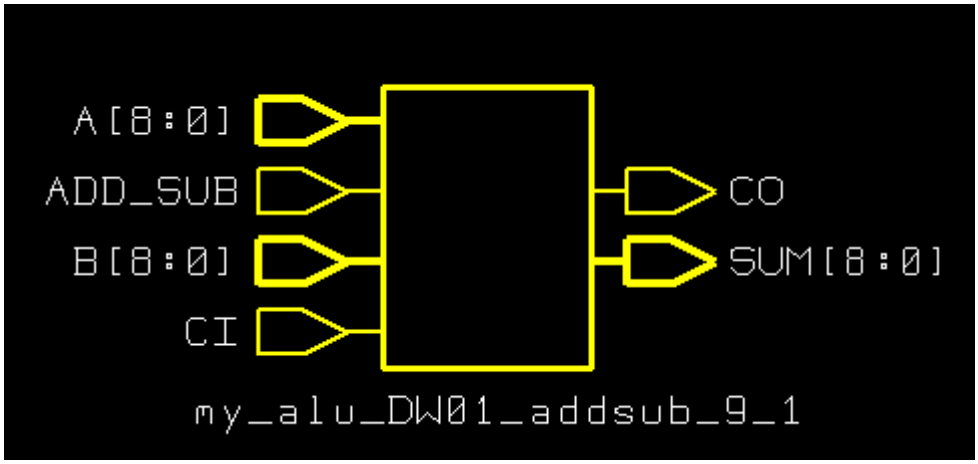
6.1.2 ALU Symbol



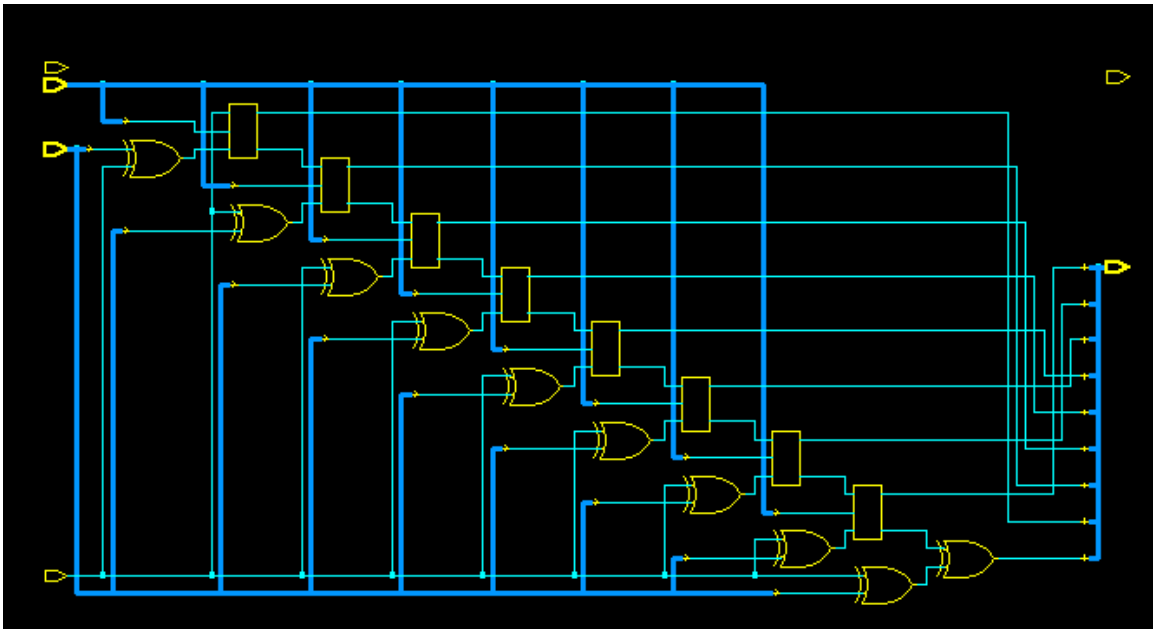
6.1.3 Register Symbol



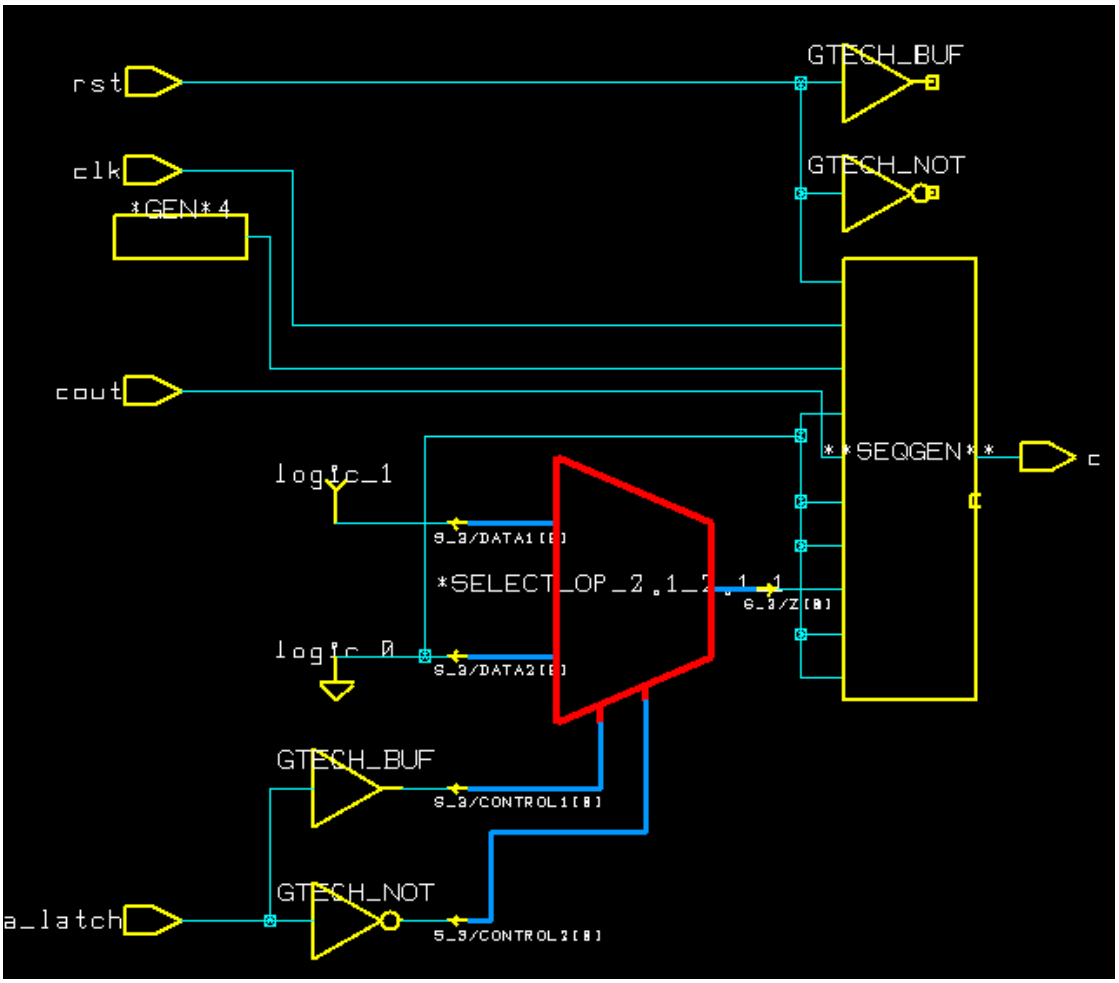
6.1.4 Adder-Subtractor Symbol



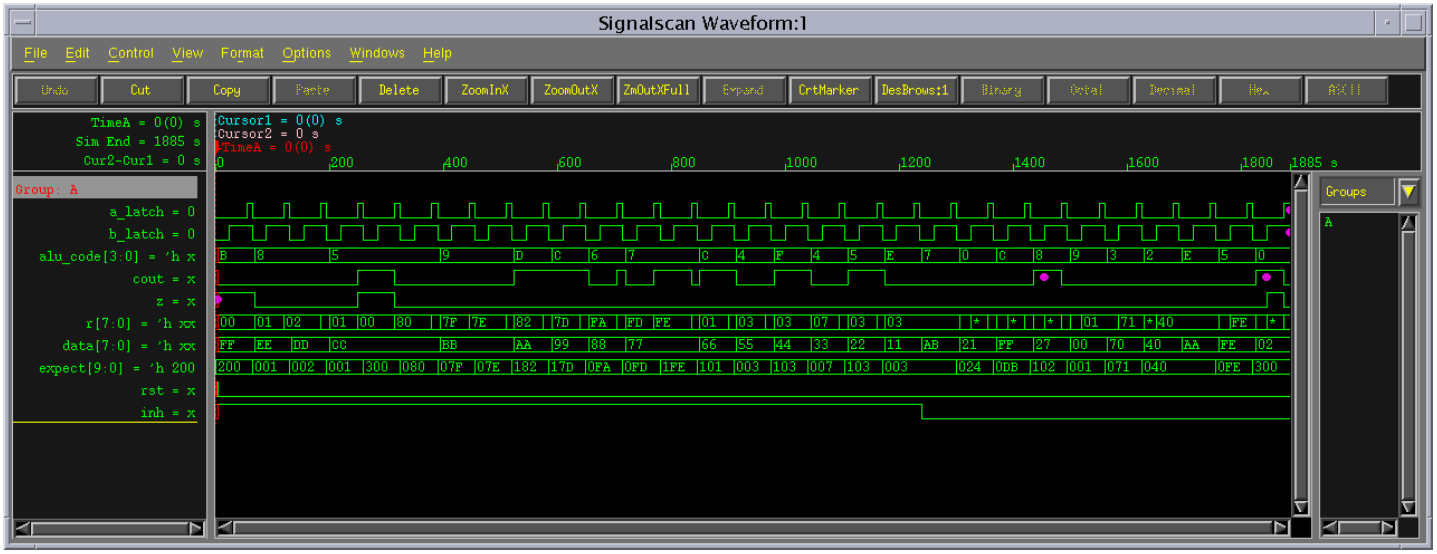
6.1.5 Adder-Subtractor Schematic



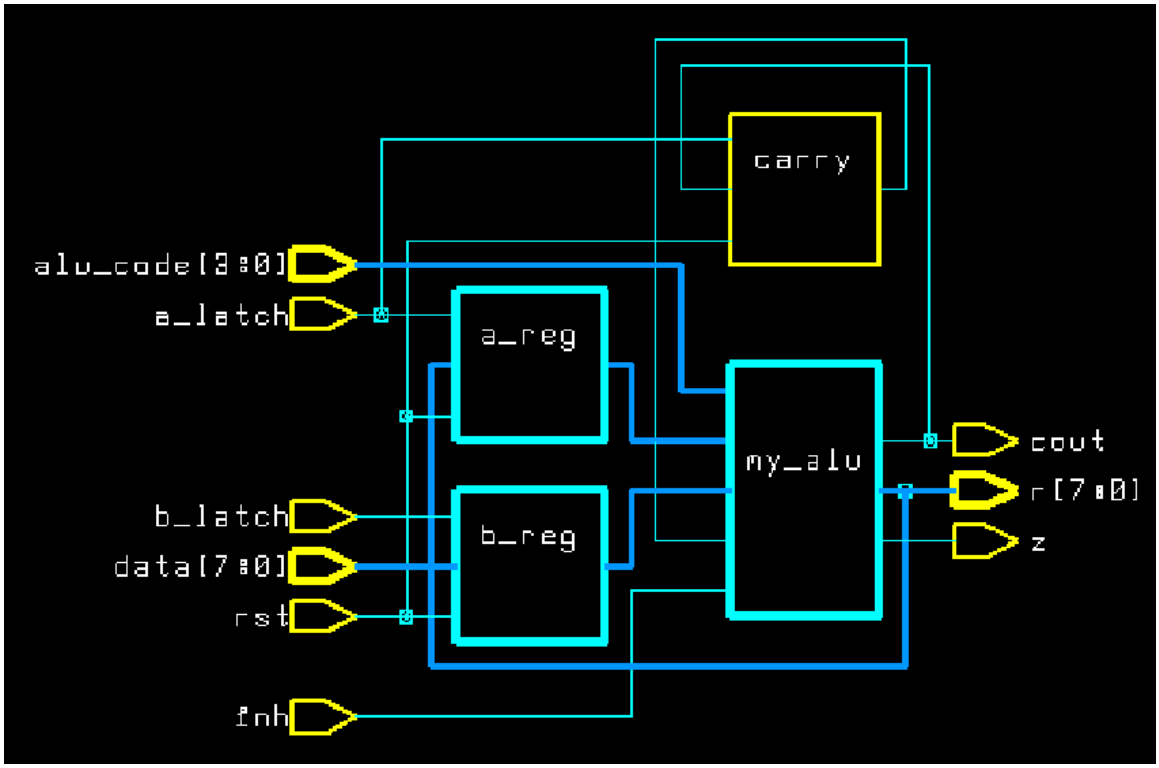
6.1.6 Carry Register Schematic



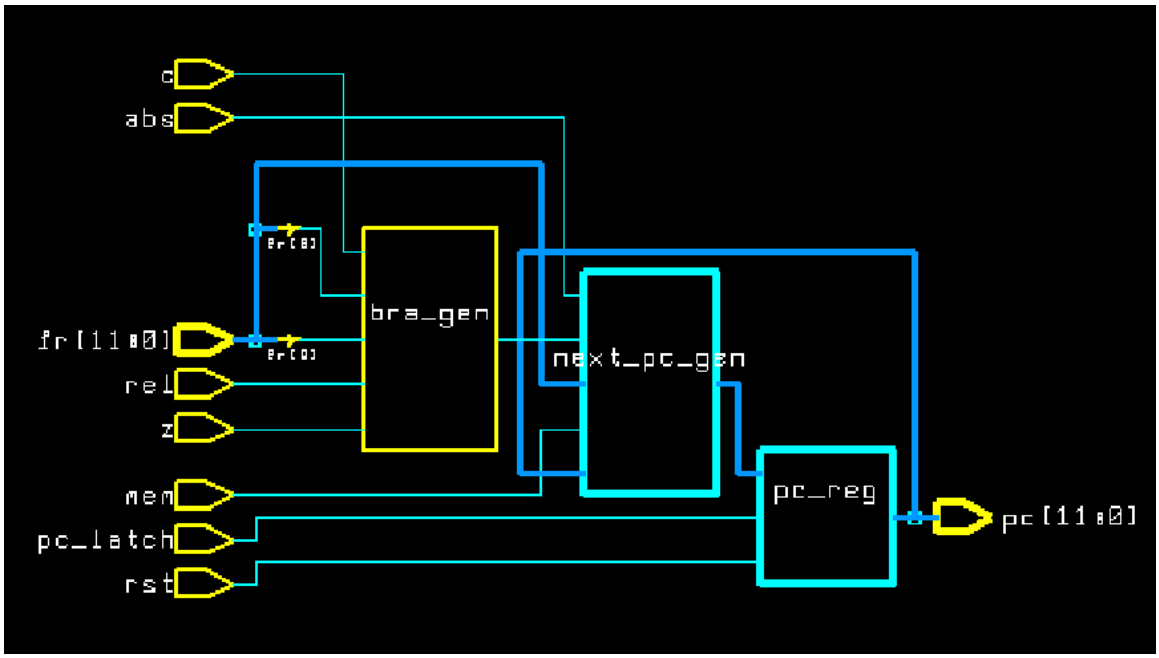
6.1.7 ALU Wave Form Output



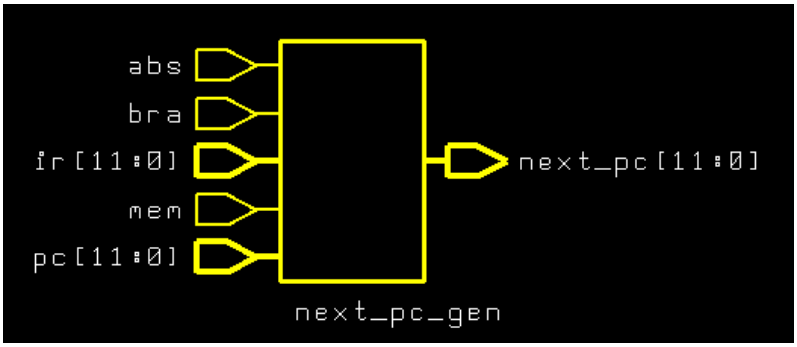
6.1.8 ALU Integrated



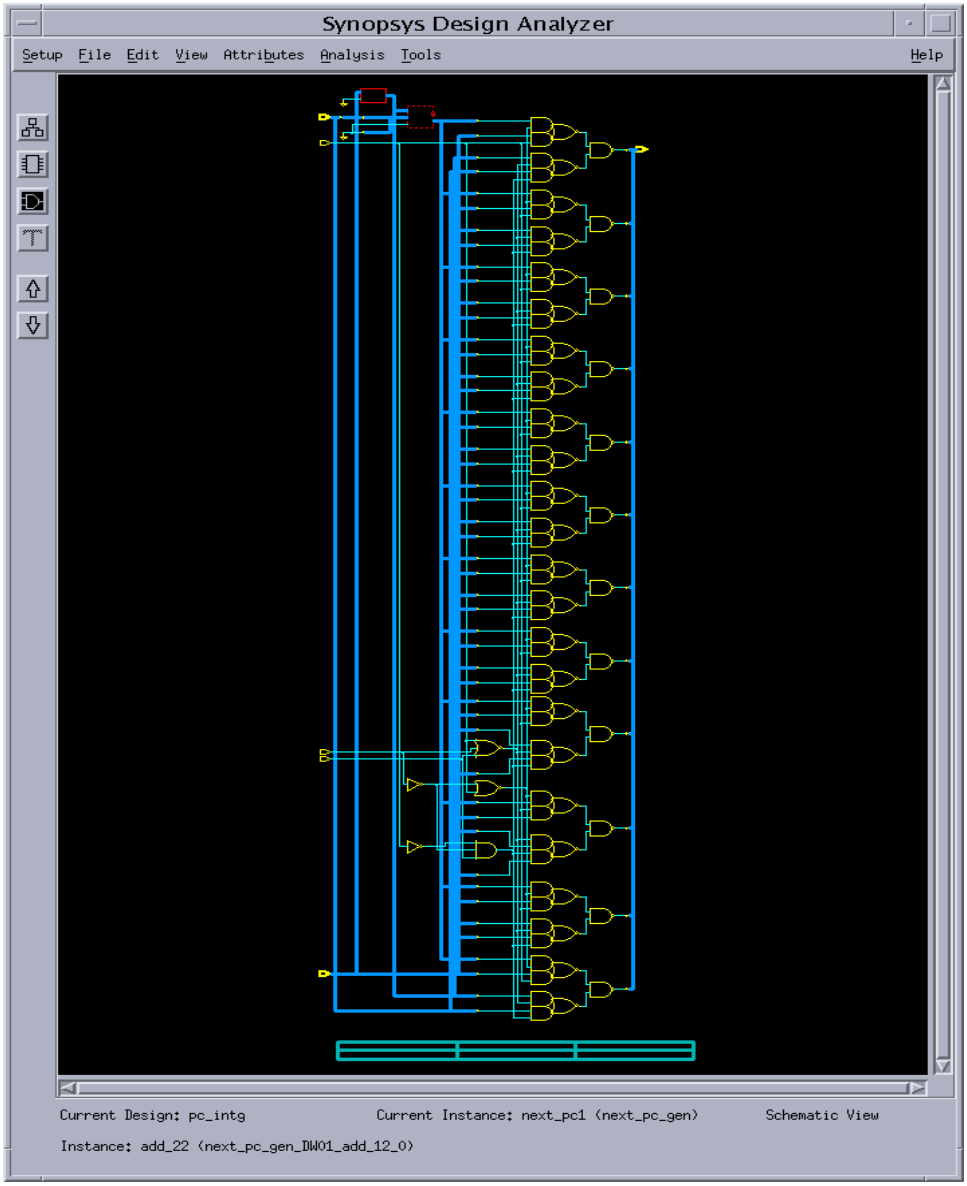
6.2.1 Program Control Integrated Symbol

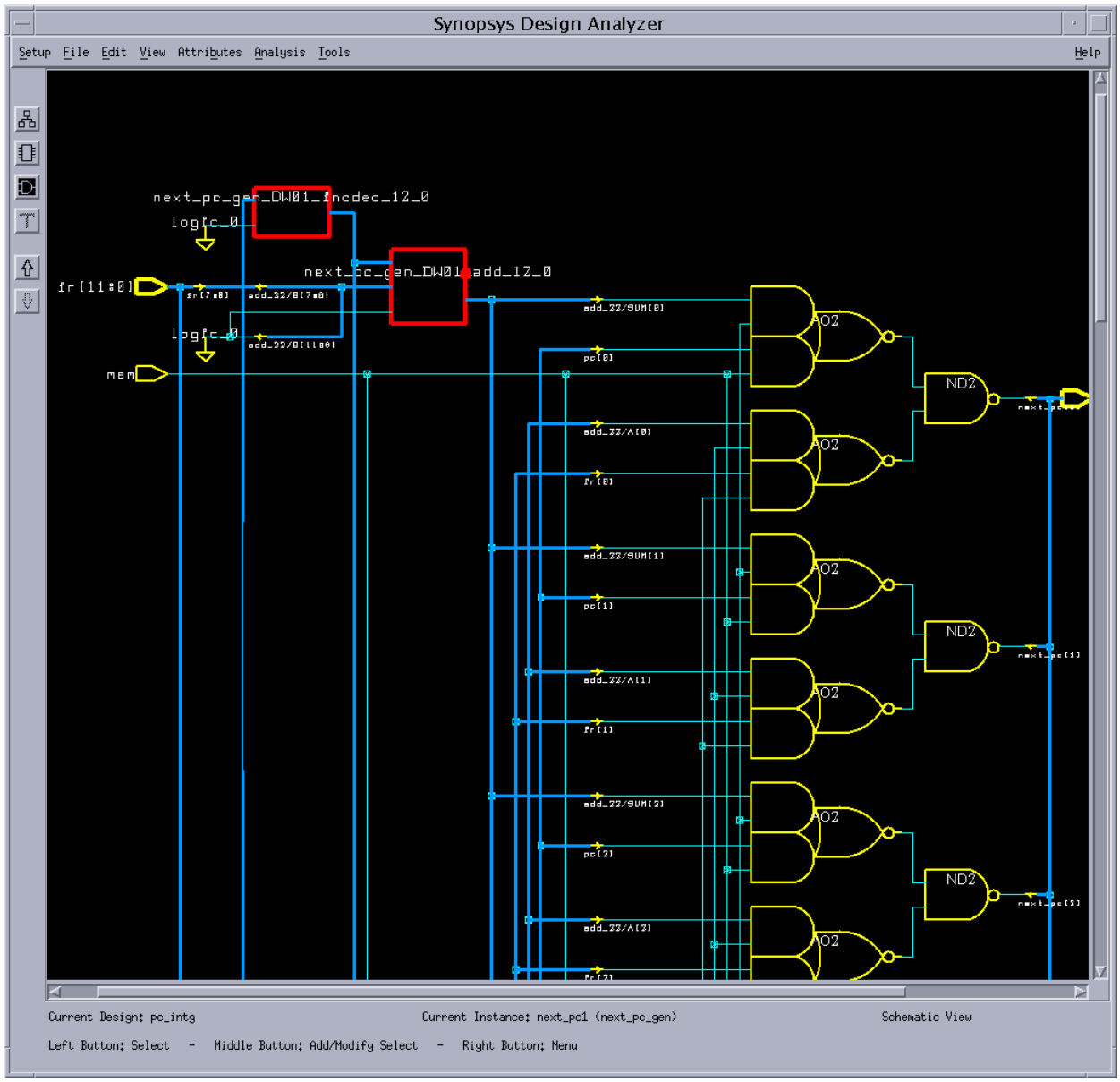


6.2.2 Next PC Generation Symbol

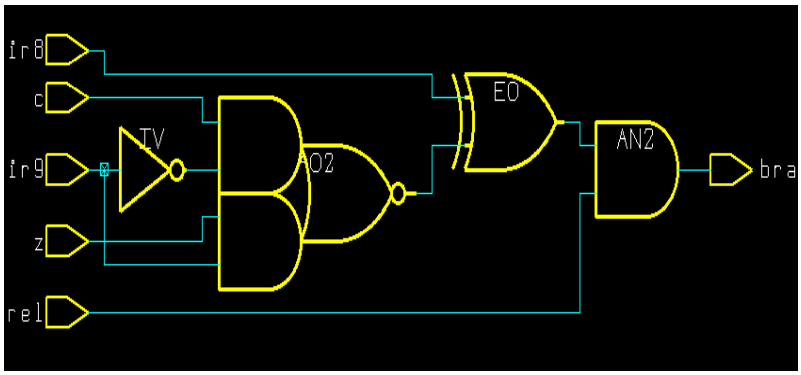


6.2.3 PC-Integrated Schematic

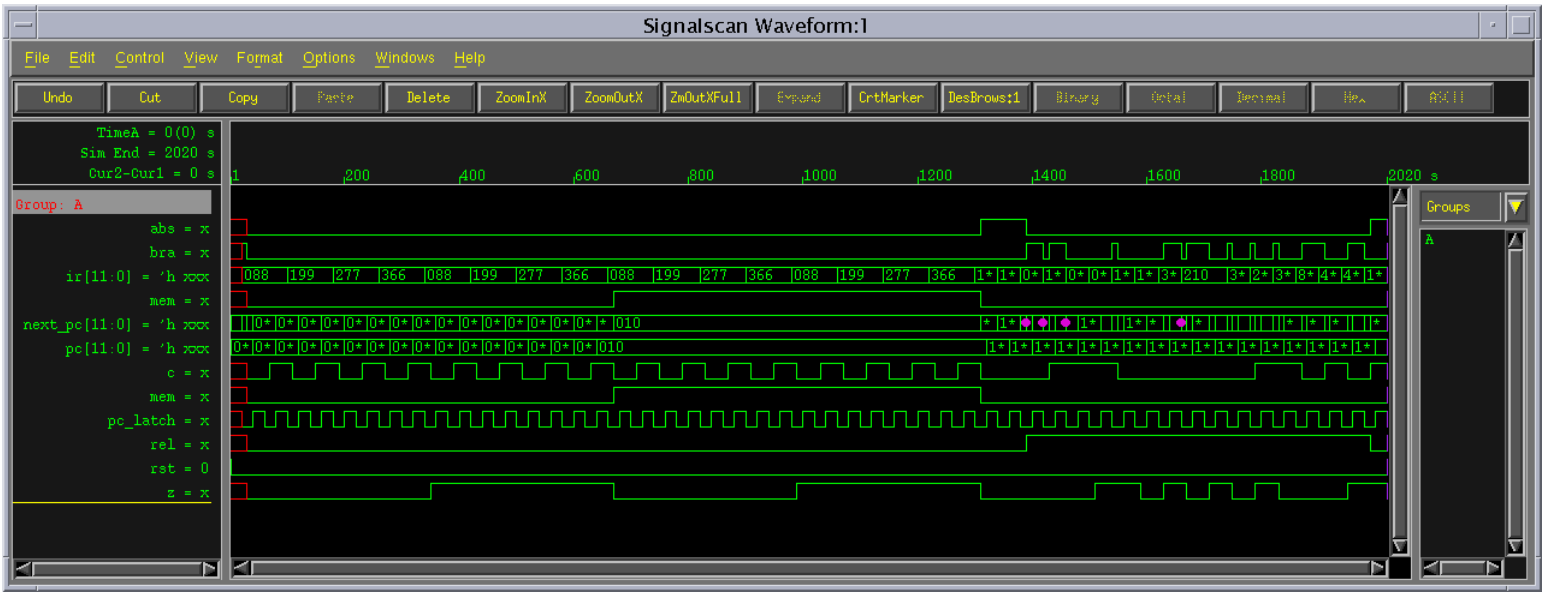




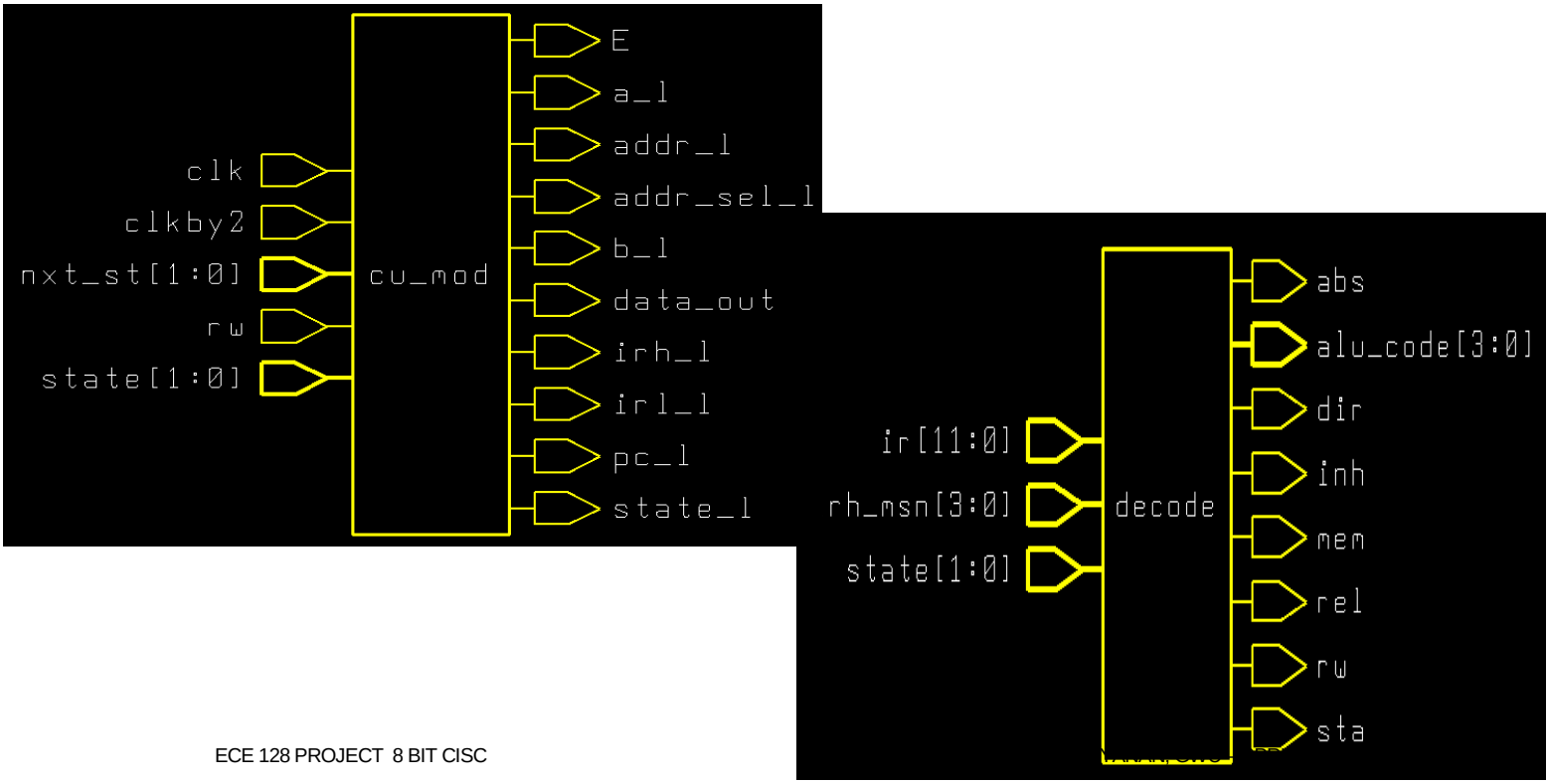
6.2.4 Program Control – Branch Genration Schematic



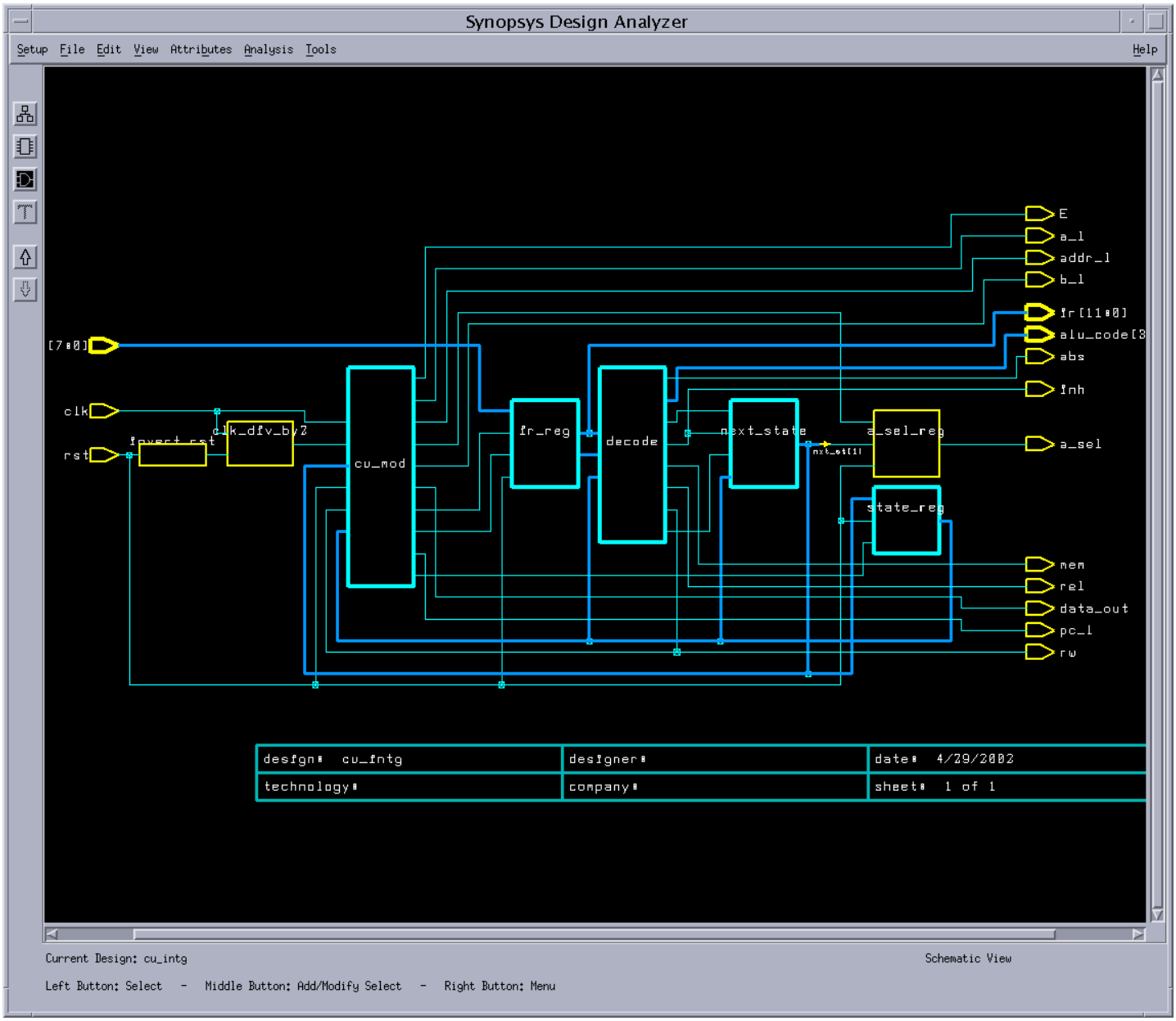
6.2.5 Program Control Waveform Output



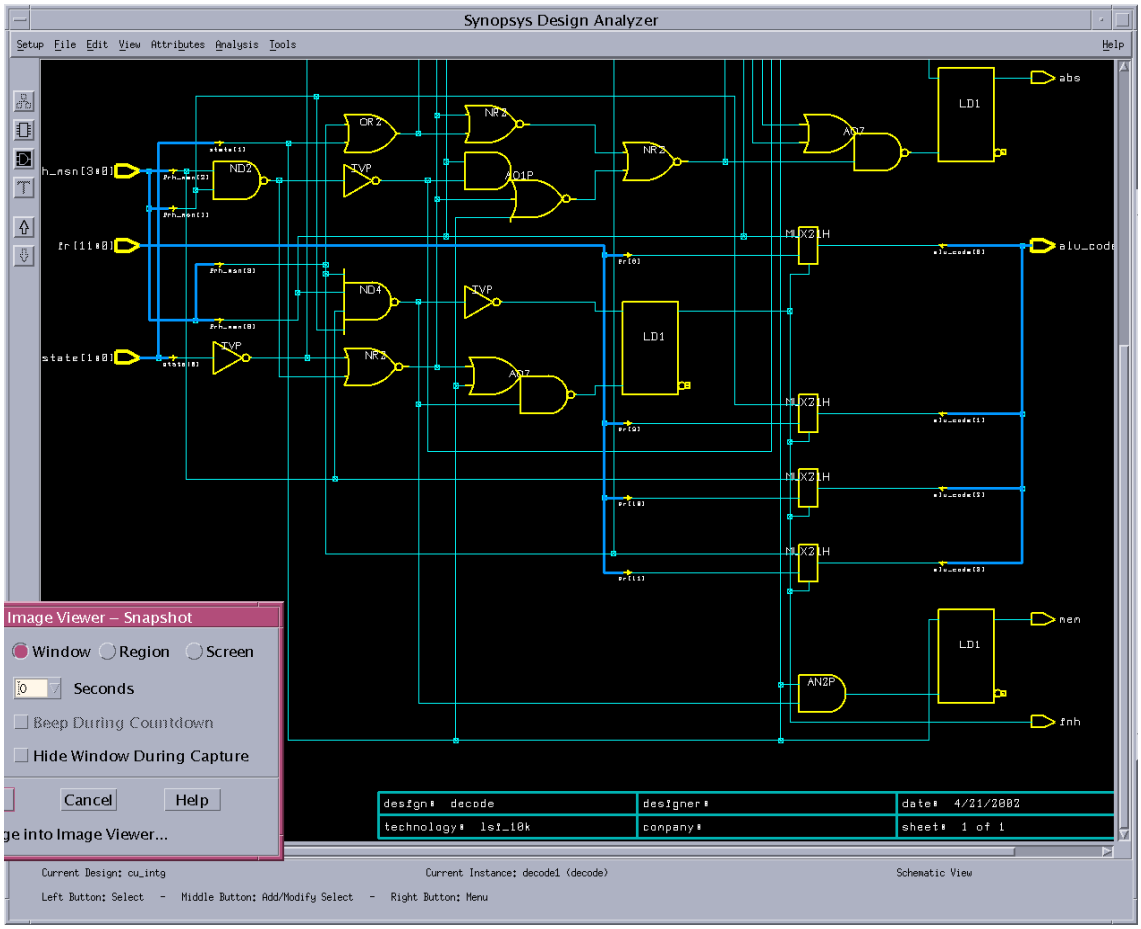
6.3.1 Control Unit CU – MOD & Decode symbol



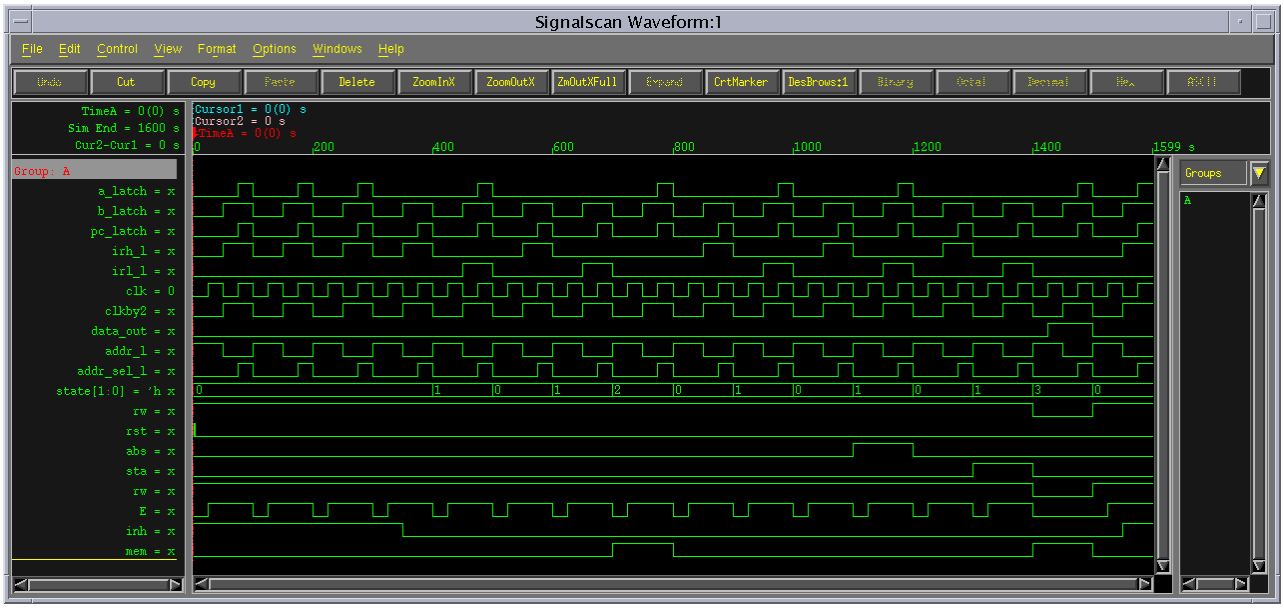
6.3.2 Control unit Integrated Symbol



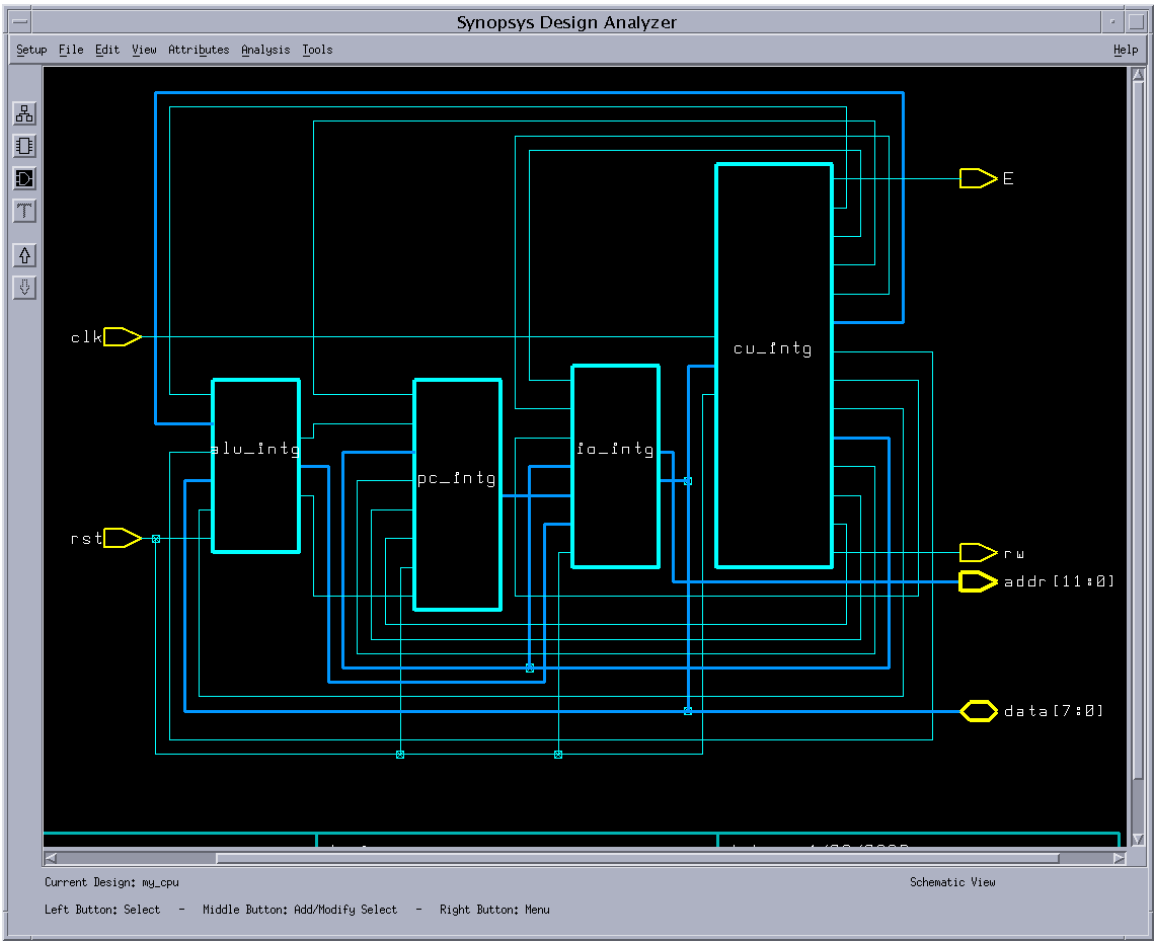
6.3.3 Decoder Schematic



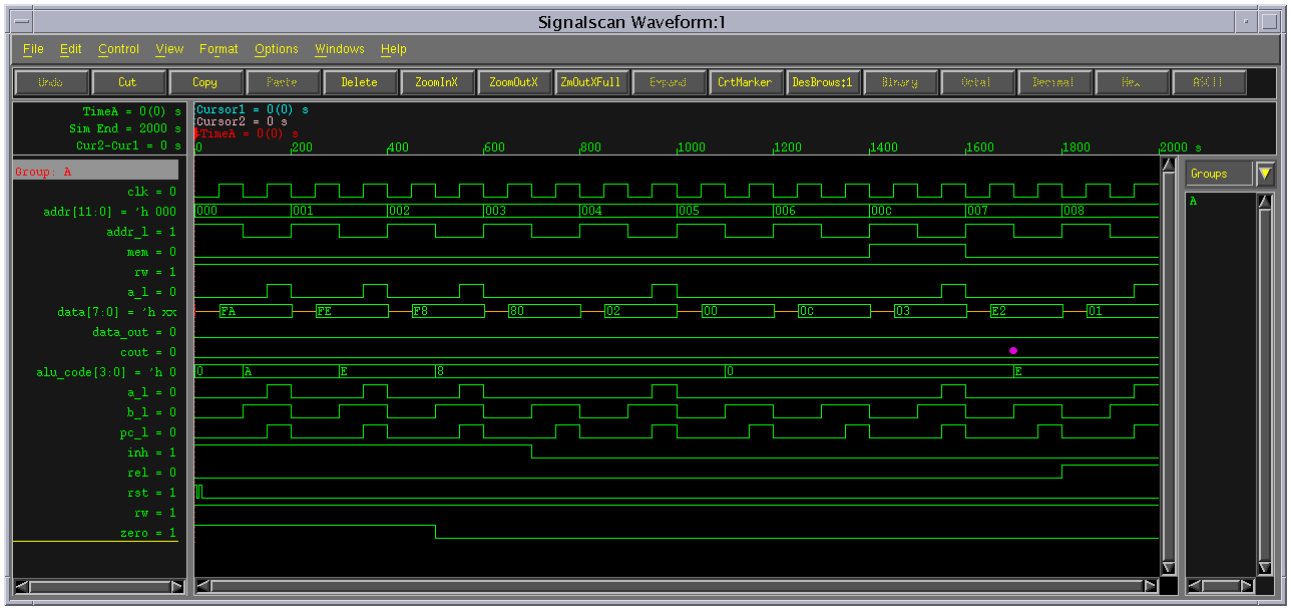
6.3.4 CU Waveform Output



6.4.0 CPU Integrated Symbol



6.4.1 CPU_test Waveform



6.4.2 Script File for CPU synthesis

```
/*************************************************************************/
/*      Module name: script file to synthesise                           */
/*      Arguments  :                                                     */
/*      Description:                                                     */
/*      Returns    :                                                     */
/*      Author     : K.Narayanan                                         */
/*      Bugs       :                                                     */
/*      Date       : April 29 2002                                       */
/*************************************************************************/
remove_design -all;

read -format verilog "myalu.h a_reg.v  b_reg.v c_reg.v my_alu.v  alu_intg.v bra.h bra_gen.v
next_pc_gen.v pc_reg.v pc_intg.v  io_areg.v io_asel.v io_tri.v io_intg.v a_sel_reg.v clkby2.v  cu_mod.v
decode.v invert_rst.v  irh_reg.v irl_reg.v ir_reg.v  next_state.v state_reg.v cu_intg.v my_cpu.v"

active_design=my_cpu
current_design active_design

uniquify

set_max_area 0

create_clock -name "clk" -period 10 -waveform {0.0 5.0}
set_clock_skew -ideal -uncertainty 0.33 clk
set_dont_touch_network find(clock, "clk")
set_input_delay -clock clk 5 all_inputs()
set_output_delay -clock clk 5 all_outputs()
/*set_load 5 * load_of("atl60_3_wccom/INV2/I") all_outputs()*/
/*set_drive drive_of("atl60_3_wccom/DFFC/Q") all_inputs()*/
/*set_drive 0 clk*/
compile -map_effort high
/*ungroup -all
compile -map_effort high*/
write -format db -hierarchy -output CPU_OUT_syn.db
write -format verilog -hierarchy -output CPU_OUT_syn.v
check_design > CPU_OUt.CHK
report_area > CPU_OUT.area
report_timing -path full -delay max -max_paths 5 -nworst 1 > CPU_OUT.timing
report_timing -delay min >> CPU_OUT.timing
report_constraint -all_violators -verbose > CPU_OUT.const
```

7.0 Timing Report

7.1 CPU Timing Worst &Best

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : my_cpu
Version: 2000.11
Date   : Thu May  2 15:33:19 2002
****************************************
Operating Conditions:
Wire Load Model Mode: top
  Startpoint: cu1/cu_mod1/data_out_reg
              (negative level-sensitive latch clocked by clk')
  Endpoint: cu1/cu_mod1/data_out_reg
            (negative level-sensitive latch clocked by clk')
  Path Group: clk
  Path Type: max
  Point                                       Incr       Path
  -----------------------------------------------------------
  clock clk' (fall edge)                      0.00       0.00
  clock network delay (ideal)                 0.00       0.00
  time given to startpoint                    0.00       0.00
  cu1/cu_mod1/data_out_reg/D (LD2)            0.00       0.00 r
  cu1/cu_mod1/data_out_reg/QN (LD2)           1.67       1.67 f
  cu1/cu_mod1/U194/Z (AO6P)                   0.93       2.61 r
  cu1/cu_mod1/data_out_reg/D (LD2)            0.00       2.61 r
  data arrival time                                      2.61

  clock clk' (fall edge)                      0.00       0.00
  clock network delay (ideal)                 0.00       0.00
  cu1/cu_mod1/data_out_reg/GN (LD2)           0.00       0.00 f
  time borrowed from endpoint                 0.00       0.00
  data required time                                     0.00
  -----------------------------------------------------------
  data required time                                     0.00
  data arrival time                                     -2.61
  -----------------------------------------------------------
  slack (VIOLATED)                                      -2.61

  Time Borrowing Information
  -------------------------------------------
  clk' pulse width                            5.00
  library setup time                         -0.40
  clock uncertainty                          -0.33
  -------------------------------------------
  max time borrow                             4.27
  actual time borrow                          0.00
  -------------------------------------------
```

```
****************************************
Report : timing
        -path full
        -delay min
        -max_paths 1
Design : my_cpu
Version: 2000.11
Date   : Thu May  2 15:33:19 2002
****************************************
Operating Conditions:
Wire Load Model Mode: top
  Startpoint: clk (clock source 'clk')
  Endpoint: cu1/cu_mod1/addr_sel_l_reg
            (positive level-sensitive latch clocked by clk)
  Path Group: clk
  Path Type: min
  Point                                    Incr       Path
  -----------------------------------------------------------
  clock clk (fall edge)                    5.00       5.00
  input external delay                     1.00       6.00 f
  clk (in)                                 0.00       6.00 f
  cu1/clk (cu_intg)                        0.00       6.00 f
  cu1/cu_mod1/clk (cu_mod)                 0.00       6.00 f
  cu1/cu_mod1/addr_sel_l_reg/D (LD1)       0.00       6.00 f
  data arrival time                                   6.00

  clock clk (fall edge)                    5.00       5.00
  clock network delay (ideal)              0.00       5.00
  clock uncertainty                        0.33       5.33
  cu1/cu_mod1/addr_sel_l_reg/G (LD1)       0.00       5.33 f
  library hold time                        0.40       5.73
  data required time                                  5.73
  -----------------------------------------------------------
  data required time                                  5.73
  data arrival time                                  -6.00
  -----------------------------------------------------------
  slack (MET)                                         0.27
```

7.2 Control Unit Timing Worst &Best

```
 ****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : cu_intg
Version: 2000.11
Date   : Thu May  2 15:53:55 2002
****************************************
Operating Conditions:
Wire Load Model Mode: top
  Startpoint: clk (clock source 'clk')
  Endpoint: cu_mod1/E_reg
            (positive level-sensitive latch clocked by clk)
  Path Group: clk
  Path Type: max
```

```
  Point                                    Incr       Path
  -----------------------------------------------------------
  clock clk (rise edge)                    0.00       0.00
  input external delay                     5.00       5.00 r
  clk (in)                                 0.00       5.00 r
  cu_mod1/clk (cu_mod)                     0.00       5.00 r
  cu_mod1/U193/Z (IVAP)                    0.52       5.52 f
  cu_mod1/U189/Z (ND2P)                    1.06       6.58 r
  cu_mod1/U195/Z (IVAP)                    0.21       6.79 f
  cu_mod1/U192/Z (AO6P)                    0.93       7.72 r
  cu_mod1/E_reg/D (LD1)                    0.00       7.72 r
  data arrival time                                   7.72

  clock clk (rise edge)                    0.00       0.00
  clock network delay (ideal)              0.00       0.00
  cu_mod1/E_reg/G (LD1)                    0.00       0.00 r
  time borrowed from endpoint              4.27       4.27
  data required time                                  4.27
  -----------------------------------------------------------
  data required time                                  4.27
  data arrival time                                  -7.72
  -----------------------------------------------------------
  slack (VIOLATED)                                   -3.45

  Time Borrowing Information
  -------------------------------------------
  clk pulse width                          5.00
  library setup time                      -0.40
  clock uncertainty                       -0.33
  -------------------------------------------
  max time borrow                          4.27
  actual time borrow                       4.27
  -------------------------------------------


****************************************
Report : timing
        -path full
        -delay min
        -max_paths 1
Design : cu_intg
Version: 2000.11
Date   : Thu May  2 15:53:55 2002
****************************************
Operating Conditions:
Wire Load Model Mode: top
  Startpoint: ck2/clkby2_reg
              (rising edge-triggered flip-flop clocked by clk')
  Endpoint: ck2/clkby2_reg
            (rising edge-triggered flip-flop clocked by clk')
  Path Group: clk
  Path Type: min

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock clk' (rise edge)                   5.00       5.00
```

```
  clock network delay (ideal)              0.00       5.00
  ck2/clkby2_reg/CP (FD2)                  0.00       5.00 r
  ck2/clkby2_reg/QN (FD2)                  1.62       6.62 r
  ck2/clkby2_reg/D (FD2)                   0.00       6.62 r
  data arrival time                                   6.62

  clock clk' (rise edge)                   5.00       5.00
  clock network delay (ideal)              0.00       5.00
  clock uncertainty                        0.33       5.33
  ck2/clkby2_reg/CP (FD2)                  0.00       5.33 r
  library hold time                        0.40       5.73
  data required time                                  5.73
  ---------------------------------------------------------
  data required time                                  5.73
  data arrival time                                  -6.62
  ---------------------------------------------------------
  slack (MET)                                         0.89
```

7.3 ALU timing

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : alu_intg
Version: 2000.11
Date   : Thu May  2 15:45:40 2002
****************************************

Operating Conditions:
Wire Load Model Mode: top

  Startpoint: alu1/result_reg[7]
              (positive level-sensitive latch)
  Endpoint: z (output port clocked by a_latch)
  Path Group: (none)
  Path Type: max

  Point                                    Incr       Path
  ---------------------------------------------------------
  alu1/result_reg[7]/G (LD1)               0.00       0.00 r
  alu1/result_reg[7]/Q (LD1)               1.12       1.12 f
  alu1/U490/Z (NR8P)                       2.11       3.23 r
  alu1/zero (my_alu)                       0.00       3.23 r
  z (out)                                  0.00       3.23 r
  data arrival time                                   3.23
  ---------------------------------------------------------
  (Path is unconstrained)
```

7.4 Program Control timing Worst & Best

```
****************************************
Report : timing
         -path full
         -delay max
         -max_paths 1
Design : pc_intg
Version: 2000.11
Date   : Thu May  2 15:49:48 2002
****************************************
Operating Conditions:
Wire Load Model Mode: top
  Startpoint: ir[4] (input port clocked by pc_latch)
  Endpoint: pc_reg1/pc_reg[9]
            (rising edge-triggered flip-flop clocked by pc_latch)
  Path Group: pc_latch
  Path Type: max

  Point                                                   Incr       Path
  ---------------------------------------------------------------------------
  clock pc_latch (rise edge)                              0.00       0.00
  clock network delay (ideal)                             0.00       0.00
  input external delay                                    5.00       5.00 f
  ir[4] (in)                                              0.00       5.00 f
  next_pc1/ir[4] (next_pc_gen)                            0.00       5.00 f
  next_pc1/add_22/B[4] (next_pc_gen_DW01_add_12_0)        0.00       5.00 f
  next_pc1/add_22/U93/Z (NR2P)                            0.82       5.82 r
  next_pc1/add_22/U91/Z (NR2P)                            0.29       6.11 f
  next_pc1/add_22/U75/Z (AN2)                             0.87       6.98 f
  next_pc1/add_22/U70/Z (ND4P)                            0.77       7.75 r
  next_pc1/add_22/U72/Z (ND3P)                            0.57       8.32 f
  next_pc1/add_22/U80/Z (ND2P)                            0.56       8.89 r
  next_pc1/add_22/U36/Z (ENP)                             1.13      10.02 f
  next_pc1/add_22/SUM[9] (next_pc_gen_DW01_add_12_0)      0.00      10.02 f
  next_pc1/U87/Z (ND2P)                                   0.62      10.64 r
  next_pc1/U94/Z (ND4P)                                   0.49      11.14 f
  next_pc1/next_pc[9] (next_pc_gen)                       0.00      11.14 f
  pc_reg1/next_pc[9] (pc_reg)                             0.00      11.14 f
  pc_reg1/pc_reg[9]/D (FD2)                               0.00      11.14 f
  data arrival time                                                 11.14

  clock pc_latch (rise edge)                             10.00      10.00
  clock network delay (ideal)                             0.00      10.00
  clock uncertainty                                      -0.33       9.67
  pc_reg1/pc_reg[9]/CP (FD2)                              0.00       9.67 r
  library setup time                                     -0.85       8.82
  data required time                                                 8.82
  ---------------------------------------------------------------------------
  data required time                                                 8.82
  data arrival time                                                -11.14
  ---------------------------------------------------------------------------
  slack (VIOLATED)                                                  -2.32
```

```
****************************************
Report : timing
         -path full
         -delay min
         -max_paths 1
Design : pc_intg
Version: 2000.11
Date   : Thu May  2 15:49:48 2002
****************************************

Operating Conditions:
Wire Load Model Mode: top

  Startpoint: pc_reg1/pc_reg[11]
              (rising edge-triggered flip-flop clocked by pc_latch)
  Endpoint: pc_reg1/pc_reg[11]
            (rising edge-triggered flip-flop clocked by pc_latch)
  Path Group: pc_latch
  Path Type: min

  Point                                       Incr       Path
  -----------------------------------------------------------
  clock pc_latch (rise edge)                  0.00       0.00
  clock network delay (ideal)                 0.00       0.00
  pc_reg1/pc_reg[11]/CP (FD2)                 0.00       0.00 r
  pc_reg1/pc_reg[11]/Q (FD2)                  1.58       1.58 f
  pc_reg1/pc[11] (pc_reg)                     0.00       1.58 f
  next_pc1/pc[11] (next_pc_gen)               0.00       1.58 f
  next_pc1/U62/Z (ND2)                        0.64       2.22 r
  next_pc1/U43/Z (AO3)                        0.50       2.72 f
  next_pc1/next_pc[11] (next_pc_gen)          0.00       2.72 f
  pc_reg1/next_pc[11] (pc_reg)                0.00       2.72 f
  pc_reg1/pc_reg[11]/D (FD2)                  0.00       2.72 f
  data arrival time                                      2.72

  clock pc_latch (rise edge)                  0.00       0.00
  clock network delay (ideal)                 0.00       0.00
  clock uncertainty                           0.33       0.33
  pc_reg1/pc_reg[11]/CP (FD2)                 0.00       0.33 r
  library hold time                           0.40       0.73
  data required time                                     0.73
  -----------------------------------------------------------
  data required time                                     0.73
  data arrival time                                     -2.72
  -----------------------------------------------------------
  slack (MET)                                            1.99
```

7.5 Input Output Timing Worst & Best

```
****************************************
Report : timing
         -path full
         -delay max
         -max_paths 1
Design : io_intg
Version: 2000.11
Date   : Thu May  2 15:52:15 2002
****************************************
Operating Conditions:
Wire Load Model Mode: top
  Startpoint: result[0] (input port clocked by clk)
  Endpoint: data[0] (output port clocked by clk)
  Path Group: clk
  Path Type: max
  Point                                     Incr       Path
  -----------------------------------------------------------
  clock clk (rise edge)                     0.00       0.00
  clock network delay (ideal)               0.00       0.00
  input external delay                      5.00       5.00 f
  result[0] (in)                            0.00       5.00 f
  tri_1/result[0] (io_tri)                  0.00       5.00 f
  tri_1/data_tri[0]/Z (BTS4P)               0.74       5.74 f
  tri_1/data[0] (io_tri)                    0.00       5.74 f
  data[0] (out)                             0.00       5.74 f
  data arrival time                                    5.74

  clock clk (rise edge)                    10.00      10.00
  clock network delay (ideal)               0.00      10.00
  clock uncertainty                        -0.33       9.67
  output external delay                    -5.00       4.67
  data required time                                   4.67
  -----------------------------------------------------------
  data required time                                   4.67
  data arrival time                                   -5.74
  -----------------------------------------------------------
  slack (VIOLATED)                                    -1.07

****************************************
Report : timing
         -path full
         -delay min
         -max_paths 1
Design : io_intg
Version: 2000.11
Date   : Thu May  2 15:52:15 2002
****************************************
Operating Conditions:
Wire Load Model Mode: top
  Startpoint: addr_l (input port clocked by clk)
  Endpoint: addr1/addr_reg[0]
            (rising edge-triggered flip-flop clocked by clk)
  Path Group: clk
  Path Type: min
```

```
Point                                   Incr       Path
-----------------------------------------------------------
clock clk (rise edge)                   0.00       0.00
clock network delay (ideal)             0.00       0.00
input external delay                    5.00       5.00 r
addr_l (in)                             0.00       5.00 r
addr1/addr_l (addr_reg)                 0.00       5.00 r
addr1/addr_reg[0]/TE (FJK2S)            0.00       5.00 r
data arrival time                                  5.00

clock clk (rise edge)                   0.00       0.00
clock network delay (ideal)             0.00       0.00
clock uncertainty                       0.33       0.33
addr1/addr_reg[0]/CP (FJK2S)            0.00       0.33 r
library hold time                       0.00       0.33
data required time                                 0.33
-----------------------------------------------------------
data required time                                 0.33
data arrival time                                 -5.00
-----------------------------------------------------------
slack (MET)                                        4.67
```

8.0 Scope , Limitations of the project and Future work

Limitations:

- ✓ The specifications and timing references for the generation of latches was given and more emphasis was laid on the use of Verilog in description.
- ✓ Alu_test, PC_test, CU_test and CPU_test passed while CPU_test2 differed in one cycle, CPU_test3 went awry after the 25th cycle.
- ✓ Synthesis and design for timing or timing constraints was not fully attempted.

Achievements:

- ✓ The signal scan usage has been thorough. It has enabled me to track timing problems, and to understand the Control units operation in terms of generation of latches in a computer system .

Future Work:

- ✓ A complete design of a more complicated architecture viz. Implementing Pipelining, FIFO queue, Cache Controller or RAM refresher could be done starting from description at behavioral level, through designing state-machines to the ultimate simulation and synthesis.