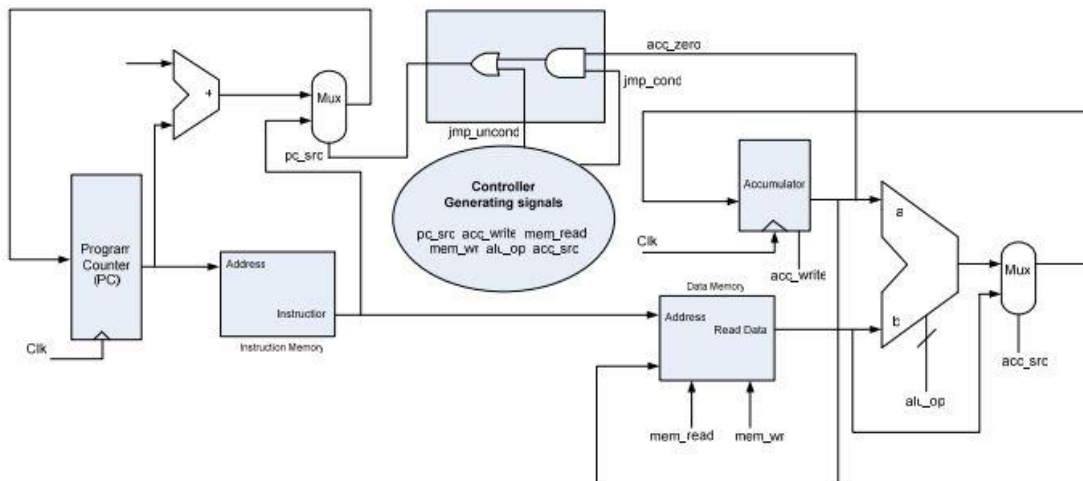


ASSIGNMENT-5

BY KANERIA DHAVAL

Verilog Code For Single Cycle Processor

DATAPATH WITH CONTROLLER



PC VERILOG CODE

```
module ProgramCounter
(
input [4:0]d_in,
input reset, clk,
output reg [4:0] d_out
);
always @(posedge clk)
if (reset)
d_out <= 5'b00000;
else
d_out <= d_in;
endmodule
```

AC VERILOG CODE

```

module Accumulator
(input [7:0] d_in,
input load, clk,
output reg [7:0] d_out
);
always @(posedge clk)
if (load)
d_out <= d_in;
initial
d_out=8'h00;
endmodule

```

ALU VERILOG CODE

```

module ALU
(
input [7:0]a, input [7:0]b,input [2:0]opcode,
output reg [7:0]alu_out
);
always @(opcode,a,b)
case(opcode)
3'b000:alu_out = a + b;
3'b001:alu_out = a - b;
3'b010:alu_out = a&b;
3'b011:alu_out = a|b;
3'b100:alu_out = ~b;
3'b101:alu_out = a^b;
3'b110:alu_out = a~^b;
default:alu_out = 0;
endcase
endmodule

```

ADDER VERILOG CODE

```

module CounterIncrement
(
input [4:0]a, input [4:0]b,
output[4:0] adder_out
);
assign adder_out = a + b;
endmodule

```

MUX-1 VERILOG CODE

```
module Mux2to1_6Bit
(
input [4:0] i0, i1, input sel,
output[4:0] mux_out
);
assign mux_out = sel ? i1 : i0;
endmodule
```

MUX-2 VERILOG CODE

```
module Mux2to1_8Bit
(
input [7:0]i0,i1, input sel,
output [7:0]mux_out
);
assign mux_out =sel?i1:i0;
endmodule
```

CONTROLLER VERILOG CODE

```
module Controller(
input [2:0] opcode,
output reg rd_mem,wr_mem,ac_src,ld_ac,pc_src,jmp_uncond);

always @(opcode)
begin
rd_mem = 1'b0;
wr_mem = 1'b0;
ac_src = 1'b0;
pc_src = 1'b0;
ld_ac = 1'b0;
jmp_uncond=1'b0;

case (opcode)
3'b000: //load accumulator from memory
begin
rd_mem = 1'b1;
wr_mem = 1'b0;
ld_ac = 1'b1;
ac_src = 1'b0;
end
end
```

```
3'b001:
begin
rd_mem = 1'b1;
wr_mem = 1'b0;
ld_ac = 1'b1;
ac_src = 1'b0;//SUBTRACT
end
3'b010:
begin
rd_mem = 1'b1;
wr_mem = 1'b0;
ld_ac = 1'b1;
ac_src = 1'b0;//AND
end
3'b011:
begin
rd_mem = 1'b1;
wr_mem = 1'b0;
ld_ac = 1'b1;
ac_src = 1'b0;//OR
end
3'b100:
begin
rd_mem = 1'b1;
wr_mem = 1'b0;
ld_ac = 1'b1;
ac_src = 1'b0;//NOT
end
3'b101:
begin
rd_mem = 1'b1;
wr_mem = 1'b0;
ld_ac = 1'b1;
ac_src = 1'b0;//XOR
end
3'b110:
begin
rd_mem = 1'b1;
wr_mem = 1'b0;
ld_ac = 1'b1;
ac_src = 1'b0;//XNOR
end
3'b111:
begin
```

```

rd_mem = 1'b0;
wr_mem = 1'b0;
ld_ac = 1'b0;
ac_src = 1'b0;
pc_src=1'b1;
jmp_uncond=1'b1;//JUMP
end

```

```

default:
begin
rd_mem = 1'b0;
wr_mem = 1'b0;
ac_src = 1'b0;
pc_src = 1'b0;
ld_ac = 1'b0;
end
endcase //end case
end //end always
endmodule

```

DATA MEMORY VERILOG CODE

```

module DataMemory (
input rd, wr,
input [4:0] abus,
input [7:0] in_dbus,
output reg [7:0] out_dbus);
reg [7:0] dm_array [0:31];

always @(rd,abus)
begin
if (rd)
out_dbus = dm_array [abus];

end

always @(wr,in_dbus) //always @(wr or abus or in_dbus)
begin
if (wr)
dm_array [abus] = in_dbus;

end

```

```

initial
begin
dm_array[0] = 8'h01;
dm_array[1] = 8'h02;
dm_array[2] = 8'h03;
dm_array[3] = 8'h04;
dm_array[4] = 8'h05;
end

endmodule

```

INSTRUCTION MEMORY VERILOG CODE

```

module InstructionMemory (input [4:0] abus, output reg [7:0] dbus);

reg [7:0] im_array [0:12];

always @(abus)

dbus = im_array [abus];

initial

begin

im_array[0]= 8'h00; // Initialize Accumulator with 0 and do addition with content of DataMemory at address 0.

im_array[1]= 8'h21; // Subtract content of accumulator with content of DataMemory at address 1.

im_array[2]= 8'h42; // Logical AND of accumulator with content of DataMemory at address 2.

im_array[3]= 8'h63; // Logical OR of accumulator with content of DataMemory at address 3.

im_array[4]= 8'h84; // Logical NOT of accumulator with content of DataMemory at address 4.

im_array[5]= 8'hA4; // Logical XOR of accumulator with content of DataMemory at address 4.

im_array[6]= 8'hC4; // Logical XNOR of accumulator with content of DataMemory at address 4.

im_array[7]= 8'hEA; // Unconditional Jump to 01010 address of Instruction memory.

im_array[10]= 8'h00; // Addition with content of DataMemory at address 0.

im_array[11]= 8'hE0; // Unconditional Jump to 00000 address of Instruction memory.

end

endmodule

```

DATAPATH MEMORY VERILOG CODE

```
module DataPath (
input reset,ld_ac, ac_src, pc_src, clk,
output [2:0] opcode,
output [4:0] im_abus,
input [7:0] im_dbus,
output [4:0] dm_abus,
output [7:0] dm_in_dbus,
input [7:0] dm_out_dbus,
output [7:0] ac_out,alu_out);

//wire [7:0] ac_out,alu_out,mux2_out;
wire [7:0]mux2_out;
wire [4:0] pc_out, adder_out,mux1_out;

ProgramCounter    pc(.d_in(mux1_out),.reset(reset),.clk(clk),.d_out(pc_out)); //instantiation
of all module
CounterIncrement  adder(.a(pc_out),.b(5'b000001),.adder_out(adder_out));
Mux2to1_6Bit      mux1(.i0(adder_out),.i1(im_dbus[4:0]),.sel(pc_src),.mux_out(mux1_out));
Accumulator       ac(.d_in(mux2_out),.load(ld_ac),.clk(clk),.d_out(ac_out));
ALU               alu(.a(ac_out),.b(dm_out_dbus),.opcode(opcode),.alu_out(alu_out));
Mux2to1_8Bit      mux2(.i0(alu_out),.i1(dm_out_dbus),.sel(ac_src),.mux_out(mux2_out));

assign im_abus = pc_out;                //assign im_abus = 6'b0000000;
assign opcode = im_dbus [7:5];
assign dm_abus = im_dbus [4:0]; //abus for DataMemory.
assign dm_in_dbus=ac_out;

endmodule
```

SMPL. CPU MEMORY VERILOG CODE

```
module CPU( //The CPU
input clk,reset,
output rd_mem,wr_mem,
output [4:0] im_abus, input [7:0] im_dbus,
output [4:0] dm_abus, output [7:0] dm_in_dbus,
input [7:0] dm_out_dbus,
output [7:0] ac_out,alu_out,
output [2:0] opcode);

//wire [2:0] opcode;
```

```
wire ac_src,ld_ac, pc_src,jmp_uncond;
```

```
DataPath dpu
```

```
(.reset(reset),.ld_ac(ld_ac),.ac_src(ac_src),.pc_src(pc_src),.clk(clk),.opcode(opcode)  
,.im_abus(im_abus),.im_dbus(im_dbus),.dm_abus(dm_abus),.dm_in_dbus(dm_in_dbus),.dm  
_out_dbus(dm_out_dbus),.ac_out(ac_out),.alu_out(alu_out));//dj
```

```
Controller cu
```

```
(.opcode(opcode),.rd_mem(rd_mem),.wr_mem(wr_mem),.ac_src(ac_src),.ld_ac(ld_ac),  
.pc_src(pc_src),.jmp_uncond(jmp_uncond));
```

```
endmodule
```

TEST BENCH SMPL. CPU MEMORY VERILOG CODE

```
module testBench;
```

```
    reg clk;  
    reg reset;  
    wire [7:0] im_dbus;  
    wire [7:0] dm_out_dbus;
```

```
    wire rd_mem;  
    wire wr_mem;  
    wire [4:0] im_abus;  
    wire [4:0] dm_abus;  
    wire [7:0] dm_in_dbus;  
    wire [7:0] ac_out,alu_out;  
    wire [2:0] opcode;
```

```
CPU uut (
```

```
    .clk(clk),.reset(reset),.rd_mem(rd_mem),.wr_mem(wr_mem),  
    .im_abus(im_abus),.im_dbus(im_dbus),.dm_abus(dm_abus),
```

```
    .dm_in_dbus(dm_in_dbus),.dm_out_dbus(dm_out_dbus),.ac_out(ac_out),.alu_out(alu  
_out),.opcode(opcode));
```

```
InstructionMemory IM (.abus(im_abus),.dbus(im_dbus));
```


DataMemory DM

```
(.rd(rd_mem),.wr(wr_mem),.abus(dm_abus),.in_dbus(dm_in_dbus),.out_dbus(dm_out_dbus)
);
```

```
initial
```

```
begin
```

```
    clk = 0;          reset = 1;//im_dbus = 8'hxx;dm_out_dbus = 8'b00000000;
```

```
    #20 reset = 1'b0;
```

```
    #500 $finish;
```

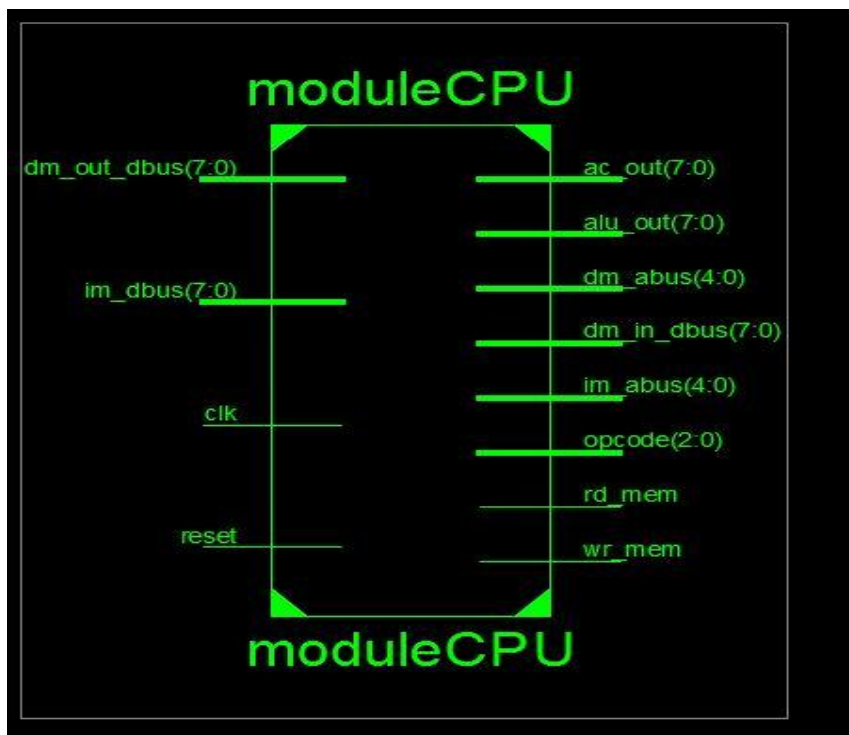
```
end
```

```
always
```

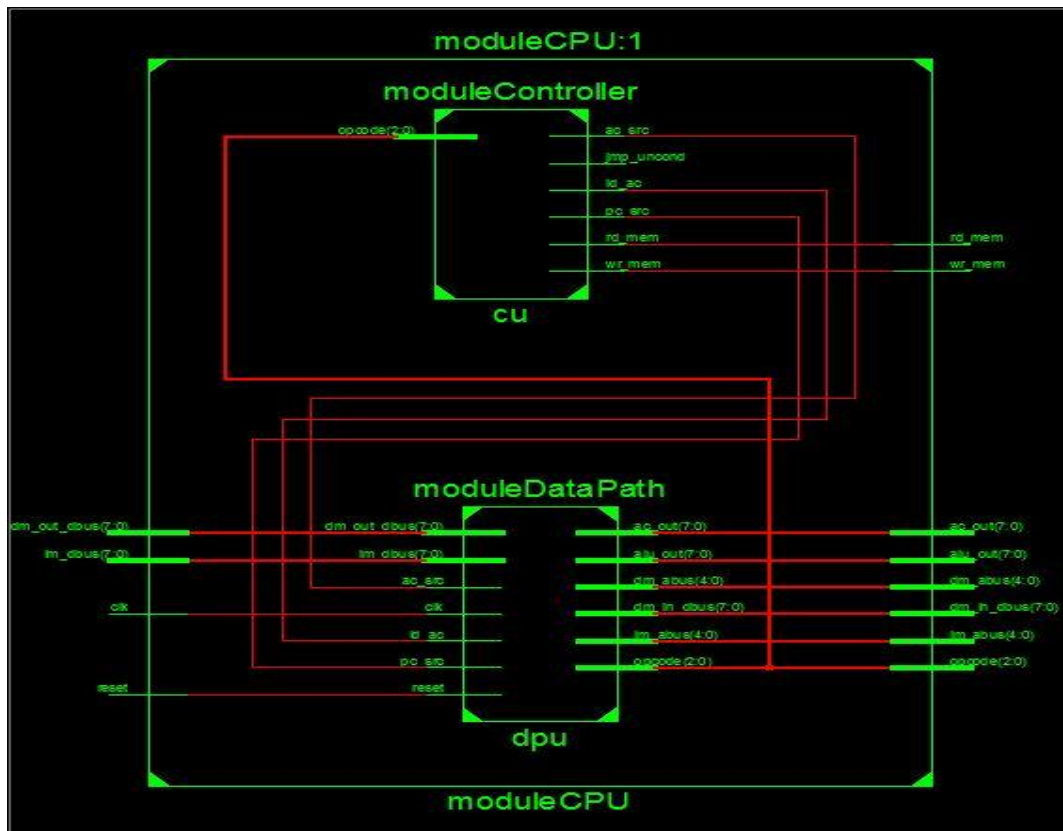
```
#10 clk = ~clk;
```

```
Endmodule
```

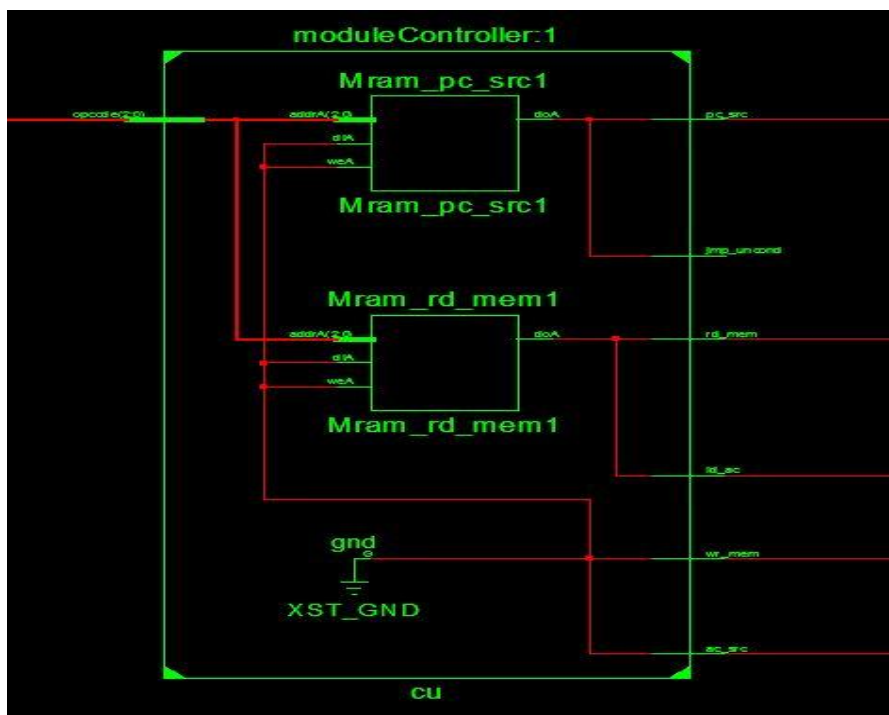
RTL DIAGRAM



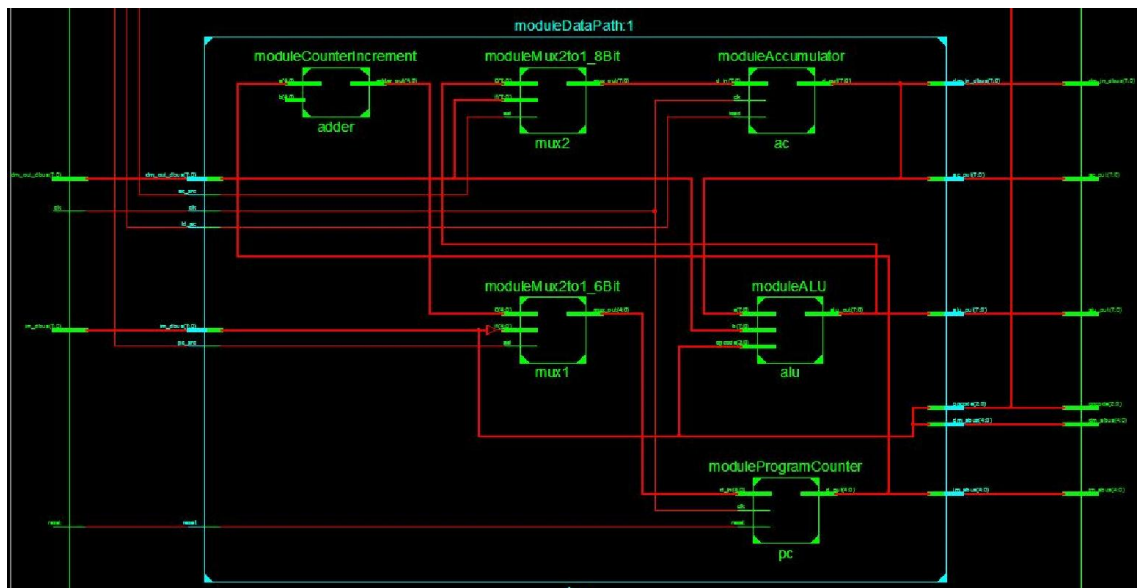
Insideview of RTL



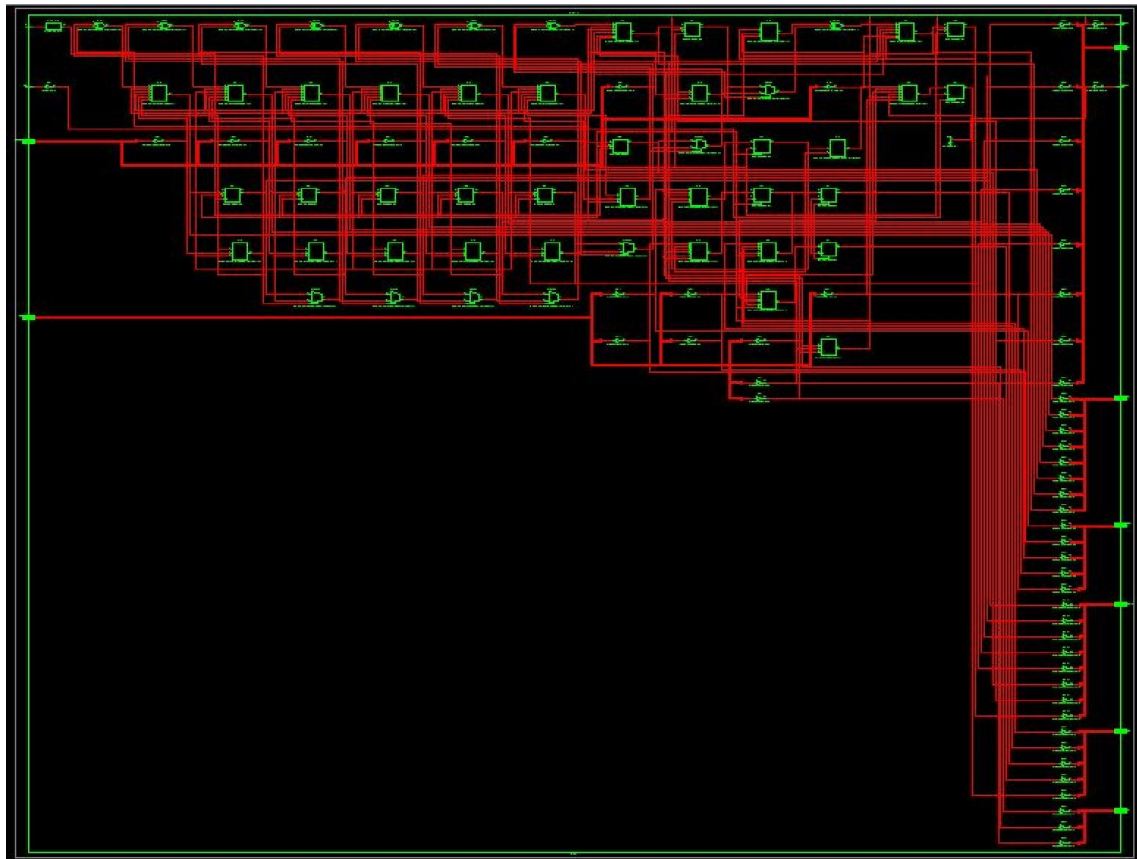
RTL OF CPU



RTL OF DATAPATH



Technology map



TEST BENCH WAVE FORM :

Data memory :

Location 00 01 02 03 04

Data 01 02 03 04 05

Instruction memory :

Location 0 1 2 3 4 5 6 7 8 9

Instruction 00 21 42 63 84 A4 C4 EA 00 E0

(* operation with accumulator with location in last 5 bit)

Ans (in acc in hex) 00 01 FF 03 07 FA FF 05 06 07

