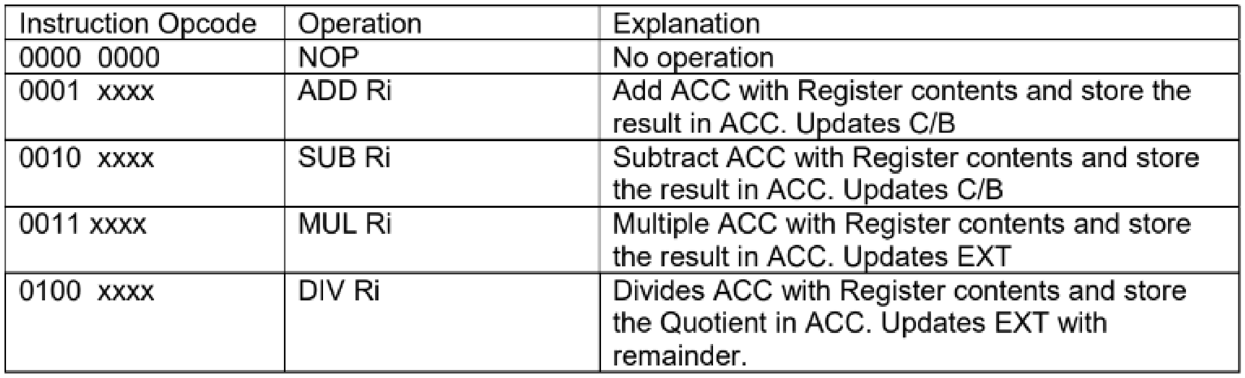
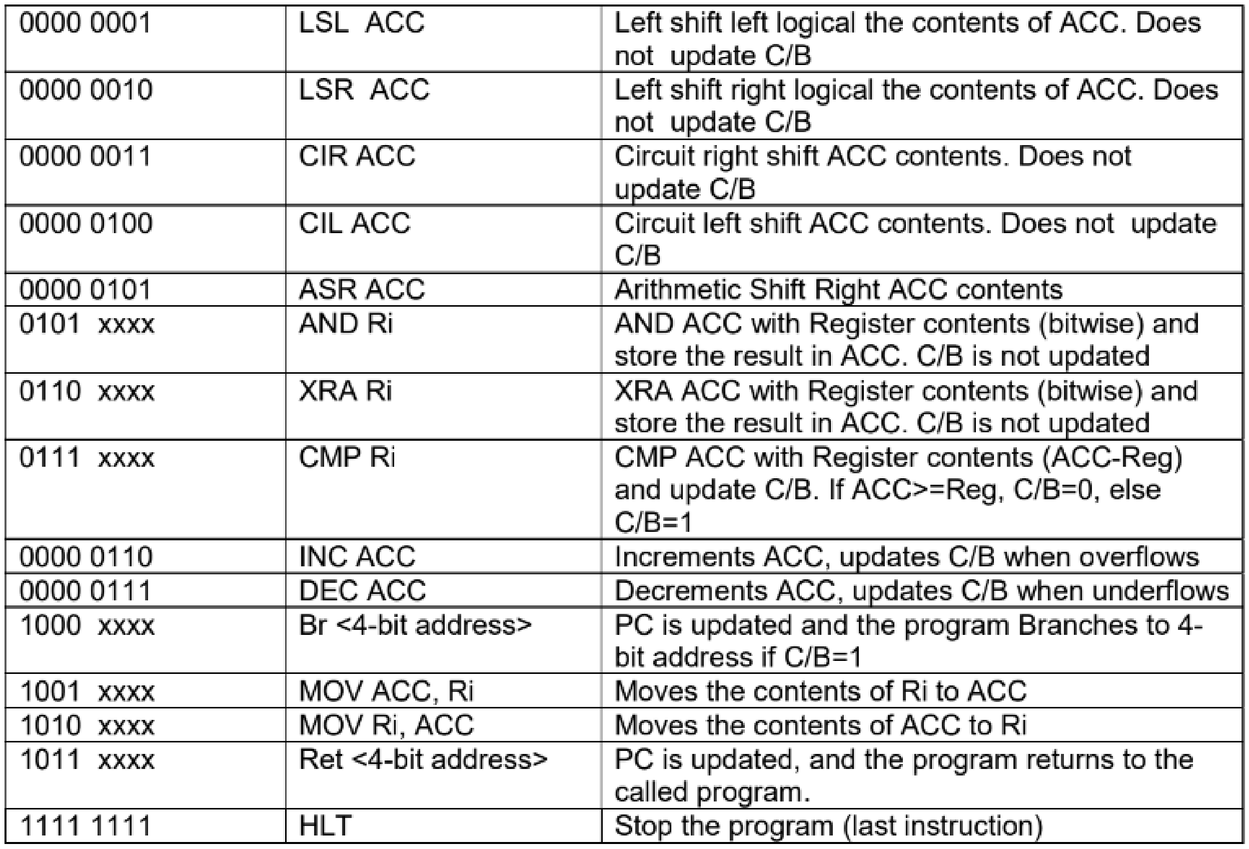
# ES 204 Digital Systems

## Final Lab

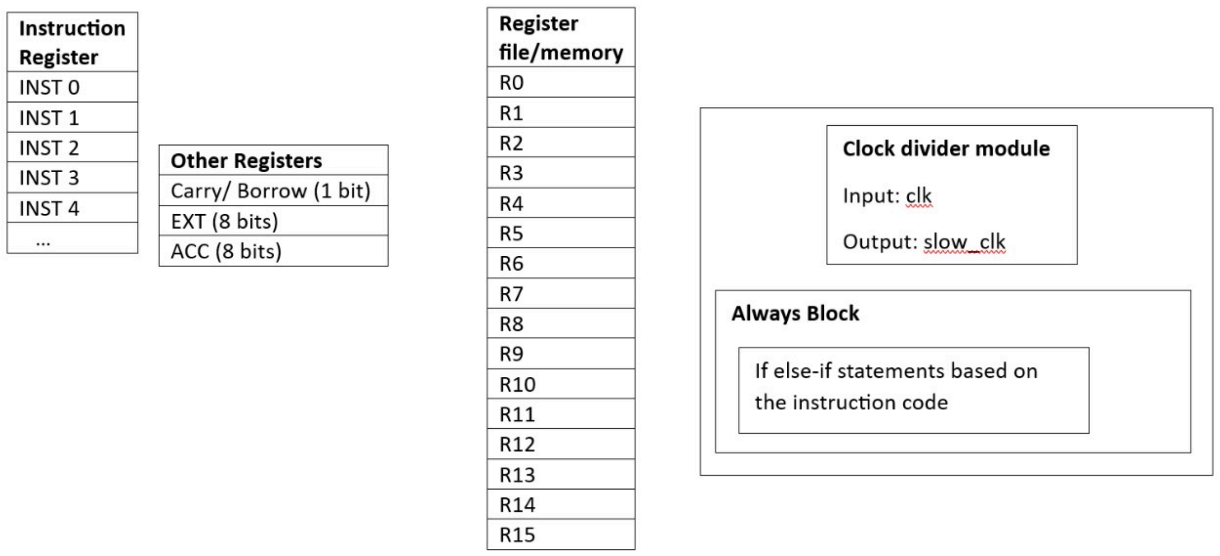
**Name:** Kishan Ved (22110122)
Lavanya (22110130)
**Date:** 22/04/2024



| Instruction Opcode | Operation | Explanation |
|---|---|---|
| 0000 0000 | NOP | No operation |
| 0001 xxxx | ADD Ri | Add ACC with Register contents and store the result in ACC. Updates C/B |
| 0010 xxxx | SUB Ri | Subtract ACC with Register contents and store the result in ACC. Updates C/B |
| 0011 xxxx | MUL Ri | Multiple ACC with Register contents and store the result in ACC. Updates EXT |
| 0100 xxxx | DIV Ri | Divides ACC with Register contents and store the Quotient in ACC. Updates EXT with remainder. |



| | | |
|---|---|---|
| 0000 0001 | LSL ACC | Left shift left logical the contents of ACC. Does not update C/B |
| 0000 0010 | LSR ACC | Left shift right logical the contents of ACC. Does not update C/B |
| 0000 0011 | CIR ACC | Circuit right shift ACC contents. Does not update C/B |
| 0000 0100 | CIL ACC | Circuit left shift ACC contents. Does not update C/B |
| 0000 0101 | ASR ACC | Arithmetic Shift Right ACC contents |
| 0101 xxxx | AND Ri | AND ACC with Register contents (bitwise) and store the result in ACC. C/B is not updated |
| 0110 xxxx | XRA Ri | XRA ACC with Register contents (bitwise) and store the result in ACC. C/B is not updated |
| 0111 xxxx | CMP Ri | CMP ACC with Register contents (ACC-Reg) and update C/B. If ACC>=Reg, C/B=0, else C/B=1 |
| 0000 0110 | INC ACC | Increments ACC, updates C/B when overflows |
| 0000 0111 | DEC ACC | Decrements ACC, updates C/B when underflows |
| 1000 xxxx | Br <4-bit address> | PC is updated and the program Branches to 4-bit address if C/B=1 |
| 1001 xxxx | MOV ACC, Ri | Moves the contents of Ri to ACC |
| 1010 xxxx | MOV Ri, ACC | Moves the contents of ACC to Ri |
| 1011 xxxx | Ret <4-bit address> | PC is updated, and the program returns to the called program. |
| 1111 1111 | HLT | Stop the program (last instruction) |



**<u>Main idea/ approach:</u>**

The Verilog module final_lab is designed with an input clock signal. We have defined parameters n and m to specify bit widths for the registers and memories in our Verilog module final_lab. Within this module, we declare registers such as inst, Cy, acc, ext, and cb, along with the memory array Q. To initialize memory, we utilize an initial block. Our module operates within an always block triggered by the positive clock edge. Here, we employ a program counter (pc) to keep track of the current instruction being executed. Depending on the opcode extracted from the instruction, various operations, including arithmetic, logical, and control tasks such as addition, subtraction, loading, bitwise XOR, multiplication, shifts, circular shifts, division, comparison, branching, storing, and no operation, are performed. The `inst` register serves as the instruction register, storing the set of instructions to be executed, while the program counter (`pc`) keeps track of the instruction's position. `Cy` is a register used for storing the carry flag. The `acc` register functions as the accumulator, holding the result of arithmetic and logical operations. `ext` is a register utilized for storing extended data, particularly relevant for multiplication and division operations. `cb` is the carry borrow register, used to indicate carry or borrow conditions in arithmetic operations. Finally, `Q` represents the memory array, where data is stored for use in computations by the processor. Each register serves a critical function in facilitating data manipulation and controlling the flow of the processor.

FPGA Implementation:

- To show the operations being performed on the FPGA, we are using 6 switches and all 16 LEDs.
- The first four switches correspond to the memory address (index of the register file), and the value or data stored on that address is displayed on the first eight LEDs.
- The next two switches are used to display the values stored in accumulator, carry/borrow(cb) and ext registers. When both the switches are off, the last eight LEDs

display the value stored in the accumulator. When the first switch is on and the second is off, the last LED shows the value in the cb register. When both the switches are on, the last eight LEDs show the value stored in the ext register.

**Implementation of our algorithm in the code:**

1. **`timescale 1ns / 1ps`**: This line specifies the timescale used in the simulation, indicating the unit of time for simulation.

2. **`module final_lab(clk)`**: This declares a Verilog module named `final_lab` with an input clock signal `clk`.

3. **Parameters**:
   - `parameter n=8, m=16;`: These parameters define the bit widths for various registers and memories used in the design.

4. **Input/Output:**
   - `input clk;`: This declares an input port for the clock signal.

5. **Registers and Memory:**
   - `reg [n-1:0] inst[2:0];`: This declares a 2-dimensional array `inst` to hold instructions, with each instruction being `n` bits wide.
   - `reg Cy;`: This declares a register to hold the carry flag.
   - `reg [n-1:0] Q[m-1:0];`: This declares a memory array `Q` to store data, with each data element being `n` bits wide.
   - `integer pc=0;`: This declares an integer register `pc` to act as the program counter.
   - `reg [n-1:0] acc;`: This declares a register `acc` to hold the accumulator.
   - `reg [n-1:0] ext;`: This declares a register `ext` to hold extended data, particularly for multiplication and division.
   - `reg cb=0;`: This declares a register `cb` to hold the carry borrow flag.

6. **Initial Block:**
   - Initializes the memory `Q` with some predefined values.

7. **Always Block:**
   - This block executes on every positive edge of the clock signal.
   - It contains a series of conditional statements based on the value of `inst[pc]`, which represents the current instruction pointed to by the program counter.
   - Each conditional block performs specific operations based on the opcode extracted from the instruction.

Code for simulation:

```
`timescale 1ns / 1ps

module final_lab(clk);
parameter n=8, m=16;
input clk;
reg [n-1:0]inst[3:0]; // This is the instruction register

reg Cy;

reg  [n-1:0] Q[m-1:0]; // This is the memory
integer pc=0; // This is the program counter
reg [n-1:0]acc; // This is the accumulator
reg [n-1:0]ext; // This is for mult and div
reg cb=0; // This is the carry borrow register
reg [7:0] temp_a, count;
initial
begin
    Q[0] = 8'b00000000;
    Q[1] = 8'b00101010;
    Q[2] = 178;
    Q[3] = 546;
    Q[4] = 657;
    Q[5] = 345;
    Q[6] = 15;
    Q[7] = 63;
    Q[8] = 711;
    Q[9] = 914;
    Q[10] = 923;
    Q[11] = 782;
    Q[12] = 822;
    Q[13] = 178;
    Q[14] = 711;
    Q[15] = 926;
// Addition
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b00010001;
//    inst[2] = 8'b11111111;
//// Multiplication
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b00110001;
//    inst[2] = 8'b11111111;
    // Circular right shift
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b00000011;
//    inst[2] = 8'b00000011;
//    inst[3] = 8'b11111111;
    // Circular left shift
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b00000100;
//    inst[2] = 8'b00000100;
//    inst[3] = 8'b11111111;
    // Arithematic right shift
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b00000101;
//    inst[2] = 8'b00000101;
//    inst[3] = 8'b11111111;
// And
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b01010001;
//    inst[2] = 8'b11111111;
// Increment ACC
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b00000110;
//    inst[2] = 8'b00000110;
//    inst[3] = 8'b11111111;

//testcase for division
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b01000001;
//    inst[2] = 8'b11111111;
//testcase for comparison
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b01110001;
//    inst[2] = 8'b11111111;
//testcase for moving contents of acc to Ri and no operation
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b00000000;
//    inst[2] = 8'b10100001;
//    inst[3] = 8'b11111111;
//testcase for branching
//    cb = 1;
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b10000000;
//    inst[2] = 8'b10100001;
//    inst[3] = 8'b11111111;
//    inst[4] = 8'b00000010;
//    inst[5] = 8'b00000010;
//    inst[6] = 8'b11111111;
```

```
//testcase for subtraction
   inst[0] = 8'b10010000;
   inst[1] = 8'b00000111;
   inst[2] = 8'b11111111;
end

always@(posedge clk) begin
   if(inst[pc]==8'b11111111)
      $finish();
   else if(inst[pc][7:4]==4'b0001) begin
      {cb,acc} = acc + Q[inst[pc][3:0]];
      pc = pc+1;
   end
   else if(inst[pc][7:4]==4'b1001) begin
      acc = Q[inst[pc][3:0]]; // Load the acc
      pc = pc+1;
   end
   else if(inst[pc][7:4]==4'b0110) begin
      acc = acc^Q[inst[pc][3:0]]; // xor acc's contents with register's contents
      pc = pc+1;
   end
   else if(inst[pc][7:4]==4'b0011) begin
      {ext,acc} = acc * Q[inst[pc][3:0]];
      pc = pc+1;
   end
   else if(inst[pc]==8'b00000001) begin
      acc <= acc << 1;
      pc = pc+1;
   end
   else if(inst[pc]==8'b00000010) begin
      acc <= acc >> 1;
      pc = pc+1;
   end
   else if(inst[pc]==8'b00000011) begin
      acc <= {acc[0], acc[7:1]}; // Circular right shift
      pc = pc + 1;
   end
   else if(inst[pc]==8'b00000100) begin
      acc <= {acc[6:0],acc[7]}; // Circular left shift
      pc = pc + 1;
   end
   else if(inst[pc]==8'b00000101) begin
      acc <= {acc[7], acc[7:1]};
      pc = pc+1;
```

```
end
else if(inst[pc][7:4]==4'b0101) begin
    acc = acc & Q[inst[pc][3:0]]; // xor acc's contents with register's contents
    pc = pc+1;
end
else if(inst[pc]==8'b00000110) begin
    if(acc==8'b11111111) begin
        cb <= 1;
        acc <= 0;
    end
    else begin
        acc <= acc + 1;
    end
    pc = pc+1;
end

else if(inst[pc][7:4]==4'b0100)begin
     temp_a = acc;
     count = 8'b00000000;
     while(temp_a >=Q[inst[pc][3:0]] )begin
     temp_a = temp_a - Q[inst[pc][3:0]];
     count = count + 1;
     end

 acc = count;
 ext = temp_a;
 pc = pc + 1;

end
else if(inst[pc][7:4]==4'b0111)begin
    if(acc>=Q[inst[pc][3:0]])
     cb = 0;
    else cb = 1;
    pc = pc + 1;
end
else if(inst[pc]==8'b00000000)
    pc = pc + 1;
else if(inst[pc][7:4]==4'b1010)begin
    Q[inst[pc][3:0]]= acc;
    pc = pc + 1;
end
else if(inst[pc][7:4] == 4'b1000)begin
    if(cb==1)
    pc <= inst[pc][3:0];
```

```
        else
        pc <= pc + 1;

    end
    else if(inst[pc][7:4] == 4'b1000)begin
        pc <= inst[pc][3:0];
    end
    else if(inst[pc][7:4] == 4'b0010)begin
        {cb,acc} = acc - Q[inst[pc][3:0]];
        pc <= pc + 1;
    end
    else if(inst[pc]==8'b00000111)begin
        {cb,acc} = acc - 1;
        pc <= pc + 1;
    end
end

endmodule
```

Testbench:

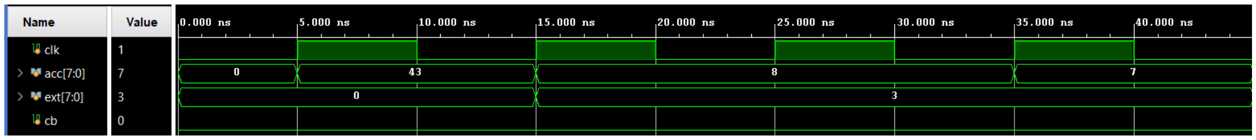
```
`timescale 1ns / 1ps
module tb();
reg clk;

final_lab uut(clk);


initial begin
clk=0;
forever #5 clk=~clk;
end

initial begin
#100;
$finish();
end

endmodule
```

Simulation:

Instructions:

```
inst[0] = 8'b10010001; // Load ACC
inst[1] = 8'b01000010; // Divide operation
inst[2] = 8'b10000101; // cb (carry borrow) is 0 here, hence, no branching takes place
inst[3] = 8'b00000111; // Decrement ACC
inst[4] = 8'b11111111; // HLT
inst[5] = 8'b00000101;
inst[6] = 8'b10110011;
```

So, 43/5, acc stores 8 and ext stores 3. Carry borrow is 0. Acc decrements to 7.



Code for FPGA:

```
`timescale 1ns / 1ps

module final_lab(clk,i1,i2,i3,i4,i5,i6,out1,out2);
parameter n=8, m=16;
input clk,i1,i2,i3,i4,i5,i6;
reg [n-1:0]inst[6:0]; // This is the instruction register

reg Cy=0;

reg  [n-1:0] Q[m-1:0]; // This is the memory
integer pc=0; // This is the program counter
reg [n-1:0]acc=0; // This is the accumulator
reg [n-1:0]ext=0; // This is for mult and div
reg cb=0; // This is the carry borrow register
reg [7:0] temp_a, count;
reg [3:0]idx;

wire slow_clk;
integer j;
output reg [7:0]out1,out2;
initial
begin
   Q[0] = 2;
   Q[1] = 40;
   Q[2] = 5;
```

```
    Q[3] = 3;
    Q[4] = 4;
    Q[5] = 8'b10000000;
    Q[6] = 8'b10001111;
    Q[7] = 7;
    Q[8] = 8;
    Q[9] = 9;
    Q[10] = 10;
    Q[11] = 11;
    Q[12] = 12;
    Q[13] = 13;
    Q[14] = 14;
    Q[15] = 15;
// Addition
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b00010001;
//    inst[2] = 8'b11111111;
//// Multiplication
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b00110001;
//    inst[2] = 8'b11111111;
    // Circular right shift
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b00000011;
//    inst[2] = 8'b00000011;
//    inst[3] = 8'b11111111;
    // Circular left shift
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b00000100;
//    inst[2] = 8'b00000100;
//    inst[3] = 8'b11111111;
    // Arithematic right shift
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b00000101;
//    inst[2] = 8'b00000101;
//    inst[3] = 8'b11111111;
// And
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b01010001;
//    inst[2] = 8'b11111111;
// Increment ACC
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b00000110;
//    inst[2] = 8'b00000110;
//    inst[3] = 8'b11111111;

//testcase for division
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b01000001;
//    inst[2] = 8'b11111111;
//testcase for comparison
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b01110001;
//    inst[2] = 8'b11111111;
//testcase for moving contents of acc to Ri and no operation
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b00000000;
//    inst[2] = 8'b10100001;
//    inst[3] = 8'b11111111;
//testcase for branching
//    cb = 1;
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b10000000;
//    inst[2] = 8'b10100001;
//    inst[3] = 8'b11111111;
//    inst[4] = 8'b00000010;
//    inst[5] = 8'b00000010;
//    inst[6] = 8'b11111111;
//testcase for subtraction
//    inst[0] = 8'b10010000;
//    inst[1] = 8'b00000111;
//    inst[2] = 8'b11111111;
// Testcase given by Ma'am
//      inst[0] = 8'b10010001;
//      inst[1] = 8'b01100001;
//      inst[2] = 8'b00010101;
//      inst[3] = 8'b00010110;
//      inst[4] = 8'b10100111;
//      inst[5] = 8'b11111111;
      inst[0] = 8'b10010001;
      inst[1] = 8'b01000010;
      inst[2] = 8'b10000101; // make sure carry borrow is 1
      inst[3] = 8'b00000111;
      inst[4] = 8'b11111111;
      inst[5] = 8'b00000101;
      inst[6] = 8'b10110011;
end
```

```
clock_divider obj(clk, slow_clk);

always@(posedge slow_clk) begin
   if(inst[pc]==8'b11111111)
      $finish();
   else if(inst[pc][7:4]==4'b0001) begin
      {cb,acc} = acc + Q[inst[pc][3:0]];
      pc = pc+1;
   end
   else if(inst[pc][7:4]==4'b1001) begin
      acc = Q[inst[pc][3:0]]; // Load the acc
      pc = pc+1;
   end
   else if(inst[pc][7:4]==4'b0110) begin
      acc = acc^Q[inst[pc][3:0]]; // xor acc's contents with register's contents
      pc = pc+1;
   end
   else if(inst[pc][7:4]==4'b0011) begin
      {ext,acc} = acc * Q[inst[pc][3:0]];
      pc = pc+1;
   end
   else if(inst[pc]==8'b00000001) begin
      acc <= acc << 1;
      pc = pc+1;
   end
   else if(inst[pc]==8'b00000010) begin
      acc <= acc >> 1;
      pc = pc+1;
   end
   else if(inst[pc]==8'b00000011) begin
      acc <= {acc[0], acc[7:1]}; // Circular right shift
      pc = pc + 1;
   end
   else if(inst[pc]==8'b00000100) begin
      acc <= {acc[6:0],acc[7]}; // Circular left shift
      pc = pc + 1;
   end
   else if(inst[pc]==8'b00000101) begin
      acc <= {acc[7], acc[7:1]};
      pc = pc+1;
   end
   else if(inst[pc][7:4]==4'b0101) begin
      acc = acc & Q[inst[pc][3:0]]; // xor acc's contents with register's contents
      pc = pc+1;
```

```
end
else if(inst[pc]==8'b00000110) begin
    if(acc==8'b11111111) begin
        cb <= 1;
        acc <= 0;
    end
    else begin
        acc <= acc + 1;
    end
    pc = pc+1;
end
else if(inst[pc][7:4]==4'b0100)begin
     temp_a = acc;
     count = 8'b00000000;
     for(j=0; j<256; j=j+1) begin
        if(temp_a >= Q[inst[pc][3:0]]) begin
             temp_a = temp_a - Q[inst[pc][3:0]];
             count = count + 1;
        end
     end

 acc = count;
 ext = temp_a;
 pc = pc + 1;
end
else if(inst[pc][7:4]==4'b0111)begin
    if(acc>=Q[inst[pc][3:0]])
     cb = 0;
    else cb = 1;
    pc = pc + 1;
end
else if(inst[pc]==8'b00000000)
    pc = pc + 1;
else if(inst[pc][7:4]==4'b1010)begin
    Q[inst[pc][3:0]]= acc;
    pc = pc + 1;
end
else if(inst[pc][7:4] == 4'b1000)begin
    if(cb==1)
    pc <= inst[pc][3:0];
    else
    pc <= pc + 1;

end
```

```
    else if(inst[pc][7:4] == 4'b1011)begin
        pc <= inst[pc][3:0];
    end
    else if(inst[pc][7:4] == 4'b0010)begin
        {cb,acc} = acc - Q[inst[pc][3:0]];
        pc <= pc + 1;
    end
    else if(inst[pc]==8'b00000111)begin
        {cb,acc} = acc - 1;
        pc <= pc + 1;
    end
    else begin
        pc <= pc+1;
    end

    if(i1==0 && i2==0 && i3==0 && i4==0) out1 = Q[0];
    else if(i1==0 && i2==0 && i3==0 && i4==1) out1 = Q[1];
    else if(i1==0 && i2==0 && i3==1 && i4==0) out1 = Q[2];
    else if(i1==0 && i2==0 && i3==1 && i4==1) out1 = Q[3];
    else if(i1==0 && i2==1 && i3==0 && i4==0) out1 = Q[4];
    else if(i1==0 && i2==1 && i3==0 && i4==1) out1 = Q[5];
    else if(i1==0 && i2==1 && i3==1 && i4==0) out1 = Q[6];
    else if(i1==0 && i2==1 && i3==1 && i4==1) out1 = Q[7];
    else if(i1==1 && i2==0 && i3==0 && i4==0) out1 = Q[8];
    else if(i1==1 && i2==0 && i3==0 && i4==1) out1 = Q[9];
    else if(i1==1 && i2==0 && i3==1 && i4==0) out1 = Q[10];
    else if(i1==1 && i2==0 && i3==1 && i4==1) out1 = Q[11];
    else if(i1==1 && i2==1 && i3==0 && i4==0) out1 = Q[12];
    else if(i1==1 && i2==1 && i3==0 && i4==1) out1 = Q[13];
    else if(i1==1 && i2==1 && i3==1 && i4==0) out1 = Q[14];
    else out1 = Q[15];

    if(i5==0 && i6==0) out2 <= acc;
    else if(i5==1 && i6==0) out2 <= cb;
    else if(i5==1 && i6==1) out2 <= ext;
end

endmodule
```

## Clock Divider:

```
`timescale 1ns / 1ps
```

```
module clock_divider(main_clk,slow_clk);
input main_clk;
output slow_clk;
reg[31:0]counter;
always@(posedge main_clk)begin
counter = counter + 1;
end
assign slow_clk = counter[27];
endmodule
```

Constraint file:

```
set_property PACKAGE_PIN W5 [get_ports clk]
set_property PACKAGE_PIN R2 [get_ports i1]
set_property PACKAGE_PIN T1 [get_ports i2]
set_property PACKAGE_PIN U1 [get_ports i3]
set_property PACKAGE_PIN W2 [get_ports i4]
set_property PACKAGE_PIN R3 [get_ports i5]
set_property PACKAGE_PIN T2 [get_ports i6]
set_property PACKAGE_PIN L1 [get_ports {out1[7]}]
set_property PACKAGE_PIN P1 [get_ports {out1[6]}]
set_property PACKAGE_PIN N3 [get_ports {out1[5]}]
set_property PACKAGE_PIN P3 [get_ports {out1[4]}]
set_property PACKAGE_PIN U3 [get_ports {out1[3]}]
set_property PACKAGE_PIN W3 [get_ports {out1[2]}]
set_property PACKAGE_PIN V3 [get_ports {out1[1]}]
set_property PACKAGE_PIN V13 [get_ports {out1[0]}]
set_property PACKAGE_PIN V14 [get_ports {out2[7]}]
set_property PACKAGE_PIN U14 [get_ports {out2[6]}]
set_property PACKAGE_PIN U15 [get_ports {out2[5]}]
set_property PACKAGE_PIN W18 [get_ports {out2[4]}]
set_property PACKAGE_PIN V19 [get_ports {out2[3]}]
set_property PACKAGE_PIN U19 [get_ports {out2[2]}]
set_property PACKAGE_PIN E19 [get_ports {out2[1]}]
set_property PACKAGE_PIN U16 [get_ports {out2[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports clk]
set_property IOSTANDARD LVCMOS33 [get_ports i1]
set_property IOSTANDARD LVCMOS33 [get_ports i2]
set_property IOSTANDARD LVCMOS33 [get_ports i3]
set_property IOSTANDARD LVCMOS33 [get_ports i4]
set_property IOSTANDARD LVCMOS33 [get_ports i5]
set_property IOSTANDARD LVCMOS33 [get_ports i6]
set_property IOSTANDARD LVCMOS33 [get_ports {out1[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out1[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out1[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out1[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out1[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out1[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out1[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out1[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out2[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out2[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out2[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out2[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out2[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out2[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out2[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {out2[0]}]
```

Videos:

https://drive.google.com/file/d/1X0bTufJRHkhaIYNaGbHit4fKMBfZsjAH/view?usp=drive_link

https://drive.google.com/file/d/1WzrrrOwmod-TqNyy4HPSHvWjR7qwGzSK/view?usp=drive_link