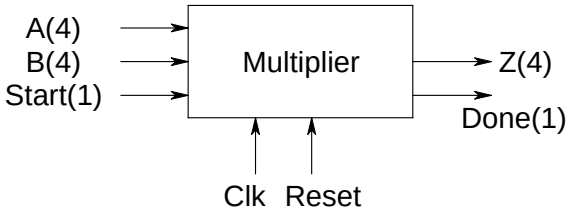
Optional Verilog Assignment

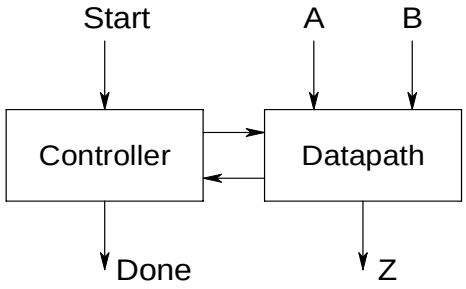
Objective : RTL (Register Transfer Level) Simulation of a Serial Multiplier

Multiplier: The multiplication algorithm is 'repeated addition' as described below:

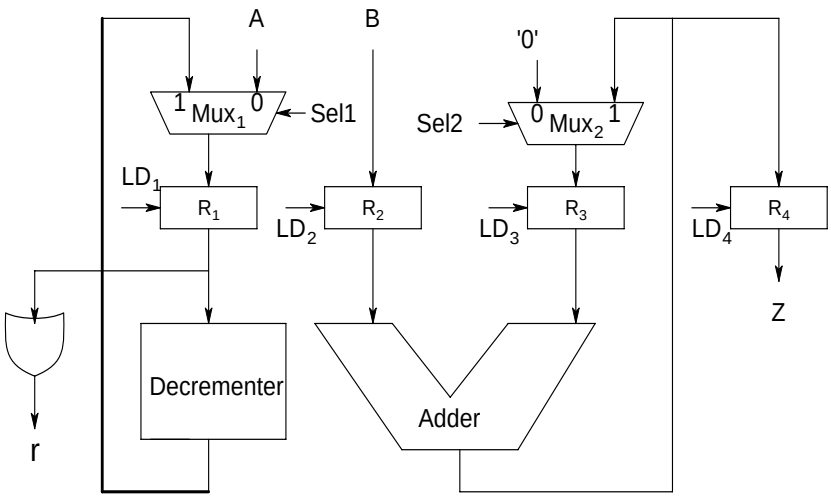
```
IF { Start = 1} Then
        Read A, B;
        Done←0;
        Temp←0;
        WHILE(A > 0) Loop
                A ←A-1;
                Temp←Temp+B;
        END WHILE
                Z←Temp;
                Done←1;
        ENDIF ;
```



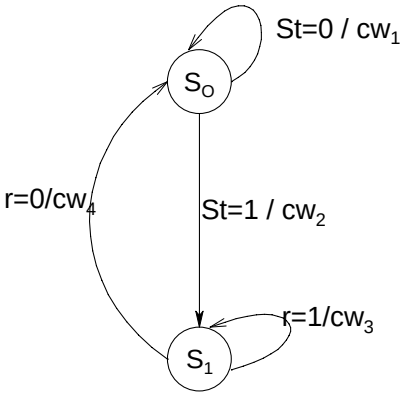
The RTL block diagrams are shown below:



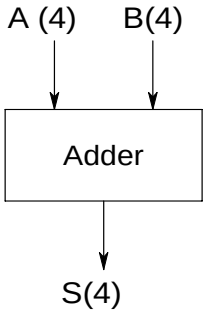
Multiplier



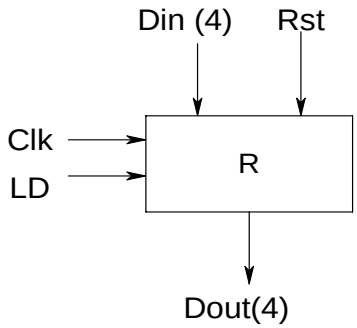
Datapath



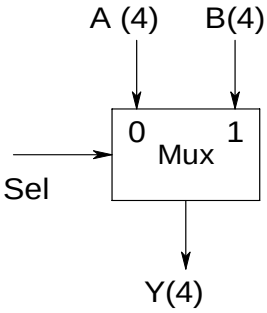
Controller



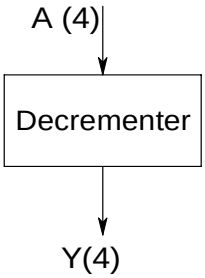

```
module adder4 (A,B,S);

/* Structural code given
using 1 bit Fulladder made
in verilog Lab */
```



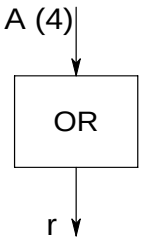

```
module Reg4 (Din,Rst,Clk,LD,Dout);

/* Write behavioral code */
```



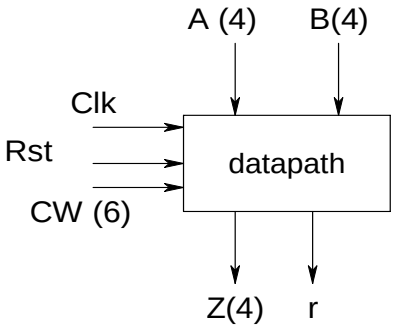

```
module mux4 (A,B,sel,Y);

/* Write behavioral code*/
```




```
module dec4 (A, Y);

/* Structural Code given*/
```

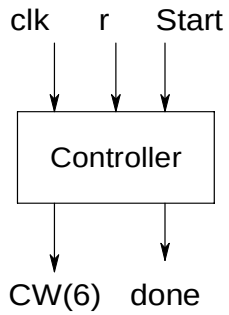



```
module or4 (A,r);

/*This essentially keeps
count of the number of
steps. Think carefully
when should r become zero,
and write the code */
```




```
module datapath(A,B,Clk,Rst,CW,r,Z);

/* Combine all these modules together*/
```

The Controller is a Mealy state machine.

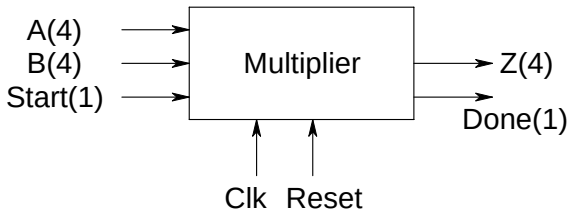



```
module controller(Clk,Start,r,CW,done);

/* Behavioral Code given */
```

CW[0] = LD_1 ; CW(1)=LD_2 ; CW(2)=LD_3 ; CW(3) = LD_4 ; CW(4) = Sel1 ; CW(5) = Sel2 .

CW1 : 6'b000000 ; CW2 : 6'b000111; CW3 : 6'b110101; CW4 : 6'b001000

Combine the datapath and controller to create a new entity mult.v




```
module multiplier
(A,B,Start,Clk,RSt,Z,done);

/* combine datapath and
controller together */
```

1. Simulate the multiplier and verify its operation using a testbench test_mult.v.
2. Put all the verilog files in a single folder, zip it and mail it to veeramani@gmail.com with the subject "[EE summer camp] Opt Verilog".
3. You may mail us as soon as you finish, but not later than 8 AM, 20th May.
4. This assgnt is compulsory for those who want to take a project in Computer Architecture.

Verilog Files

adder4.v

```
module adder4 (A,B,S);
input [3:0] A,B;
output [3:0] S;
wire [3:0] S;
wire [3:0] temp;

fulladder      adder1_1 (A[0],B[0],1'b0,S[0],temp[0]);
fulladder adder1_2 (A[1],B[1],temp[0],S[1],temp[1]);
fulladder adder1_3 (A[2],B[2],temp[1],S[2],temp[2]);
fulladder adder1_4 (A[3],B[3],temp[2],S[3],temp[3]);
/* Alternate style :
   fulladder   adder1_1
(.a(A[0]),.b(B[0]),.cin(1'b0),.sum(S[0]),.cout(temp[0]));
   Here the “.” refers to original arguments specified in the 1 bit
full adder file. They are not necessary to be used, however, this
style is considered better.
*/

endmodule
```

dec4.v

```
module decr4 (A, Y);
input [3:0] A;
output [3:0] Y;
wire [3:0] Y;
wire [3:0] C;
assign C[0]=1'b0;
assign C[1]=(A[0] | C[0]);
assign C[2]=(A[1] | C[1]);
assign C[3]=(A[2] | C[2]);
assign Y[0]=(~(A[0]) ^ C[0]);
assign Y[1]=(~(A[1]) ^ C[1]);
assign Y[2]=(~(A[2]) ^ C[2]);
assign Y[3]=(~(A[3]) ^ C[3]);

endmodule
```

controller.v

```
`define s0   1'b0
`define s1   1'b1

module controller(clk,r,start,done,CW);
input clk,r,start;
output done;
reg done;
output [5:0] CW;
reg [5:0] CW;

reg state;

always @(posedge clk) begin
   if (start == 0)
      state <= `s0;
   else begin
      if (state == `s0)
         state <= `s1;
      else if (r == 1'b1)
            state <= `s1;
      else
         state <= `s0;
   end
end

always @(state,r,start) begin
   if (state == `s0 ) begin
      if (start == 0) begin
          CW <= 6'b000000;
          done <= 0;
      end
      else begin
          CW <= 6'b000111;
          done <= 0;
      end
   end
   else if (state == `s1) begin
      if (r == 1) begin
          CW <= 6'b110101;
          done <= 0;
      end
      else begin
         CW <= 6'b001000;
         done <= 1;
      end
   end
end

endmodule
```