# COE 405, Term 162

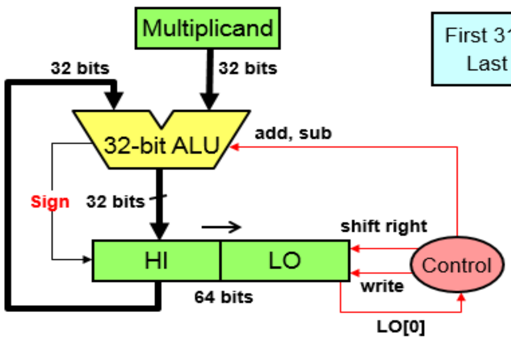
## Design & Modeling of Digital Systems

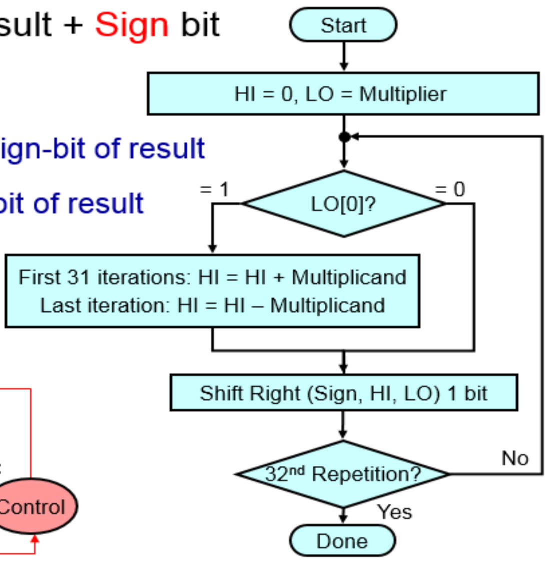
## Assignment# 4 Solution

**Due date: Saturday, April 23**

It is required to design an n-bit signed multiplier circuit. The architecture and the algorithm for performing signed multiplication are given below:

- ALU produces 32-bit result + Sign bit
- Sign bit set as follows:
  - No overflow ➔ Extend sign-bit of result
  - Overflow ➔ Invert sign-bit of result




Start
HI = 0, LO = Multiplier
LO[0]?
= 1
= 0
First 31 iterations: HI = HI + Multiplicand
Last iteration: HI = HI – Multiplicand
Shift Right (Sign, HI, LO) 1 bit
32nd Repetition?
No
Yes
Done

The example below illustrates the signed multiplication of two 4-bit numbers:

- Consider: $1010_2$ (-6)×$1111_2$ (-1), Product=$00000110_2$ (+6)
- Check for overflow: No overflow ➔ Extend sign bit
- Last iteration: add 2's complement of Multiplicand

| Iteration | | Multiplicand | Sign | Product = HI, LO |
|---|---|---|---|---|
| 0 | Initialize (HI = 0, LO = Multiplier) | 1 0 1 0 | | 0 0 0 0 1 1 1 1 |
| 1 | LO[0] = 1 => ADD | | 1 | 1 0 1 0 1 1 1 1 |
| | Shift (Sign, HI, LO) right 1 bit | 1 0 1 0 | | 1 1 0 1 0 1 1 1 |
| 2 | LO[0] = 1 => ADD | | 1 | 0 1 1 1 0 1 1 1 |
| | Shift (Sign, HI, LO) right 1 bit | 1 0 1 0 | | 1 0 1 1 1 0 1 1 |
| 3 | LO[0] = 1 => ADD | | 1 | 0 1 0 1 1 0 1 1 |
| | Shift (Sign, HI, LO) right 1 bit | 1 0 1 0 | | 1 0 1 0 1 1 0 1 |
| 4 | LO[0] = 1 => SUB (ADD 2's compl) | 0 1 1 0 | 0 | 0 0 0 0 1 1 0 1 |
| | Shift (Sign, HI, LO) right 1 bit | | | 0 0 0 0 0 1 1 0 |

**(i)** Write a parametrizable Verilog module to model the signed multiplication circuit as a combinational circuit using for loop.

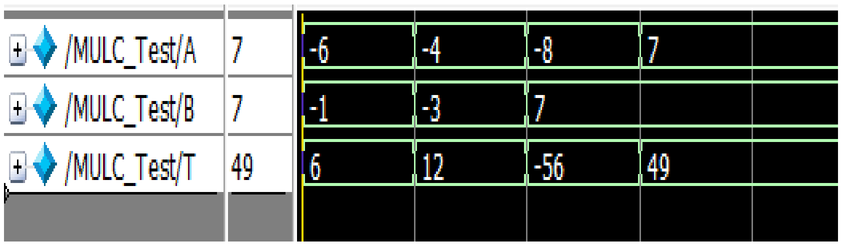
```
module MULC #(parameter N=4) (output reg [N-1:0] HI, LO, input [N-1:0] A, B);
integer i;
reg [N-1:0] C;
reg OVF, Sign;
always @(A, B) begin
   HI = 0;  LO = B;
   for (i=0; i<N; i=i+1) begin
       if( LO[0] ) begin
           if ( i==N-1 ) begin
                C = HI - A;
                OVF = (HI[N-1] ^ A[N-1]) & (HI[N-1]^C[N-1]);
           end
           else begin
               C = HI + A;
               OVF = (HI[N-1] ~^ A[N-1]) & (HI[N-1]^C[N-1]);
           end
           Sign = OVF ^ C[N-1];
           HI = C;
       end
       else Sign = HI[N-1];
     {HI, LO} = {Sign, HI,  LO[N-1:1]};
   end
end
endmodule
```

**(ii)** Write a test bench to test the correct functionality of your combinational signed multiplication circuit by instantiating a 4-bit multiplier and applying the following sets of inputs: -6*-1, -4*-3, -8*+7,+7*+7.

```
module MULC_Test();

reg [3:0] A, B;
wire [3:0] HI, LO;
wire [7:0] T;
assign T = {HI, LO};

MULC M1 (HI, LO, A, B);

initial begin
```

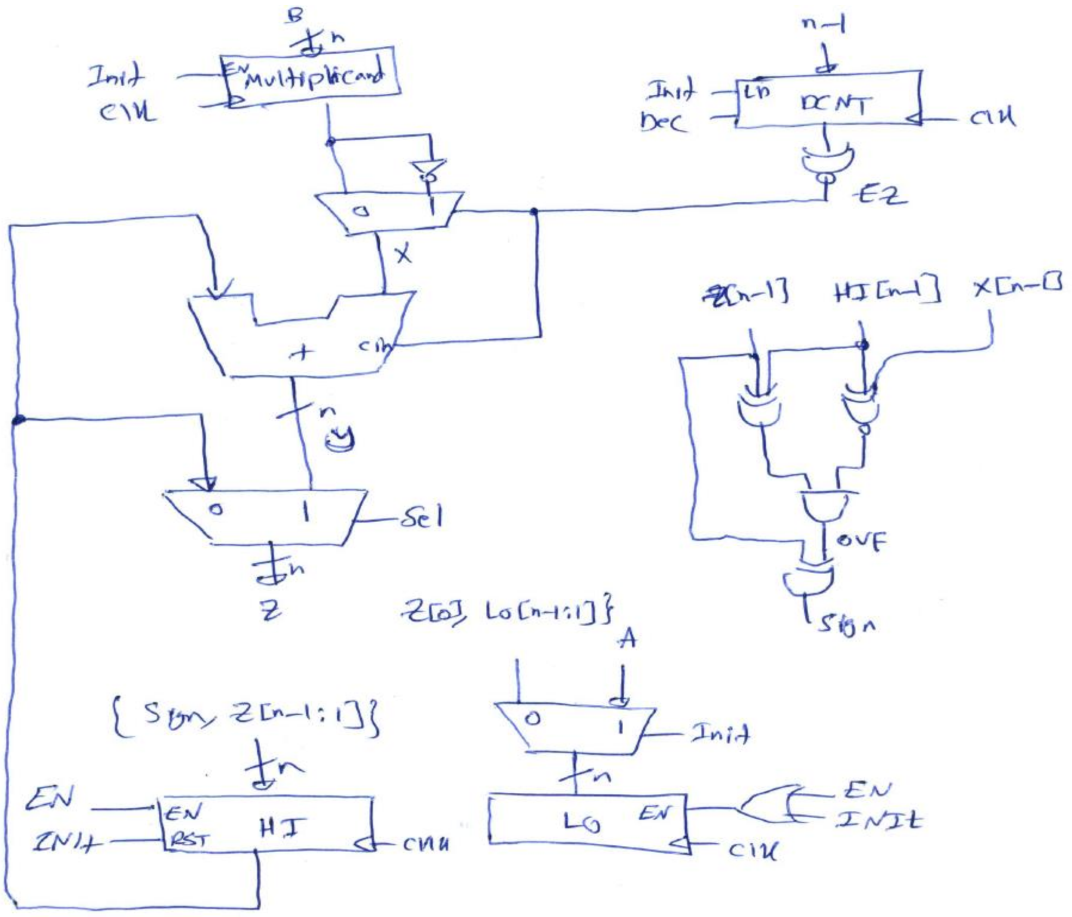
```
A=-6; B=-1;
#100 A=-4; B=-3;
#100 A=-8; B=+7;
#100 A=+7;
end
endmodule
```

As can be seen from the simulation results below, the multiplier module is producing the correct results:

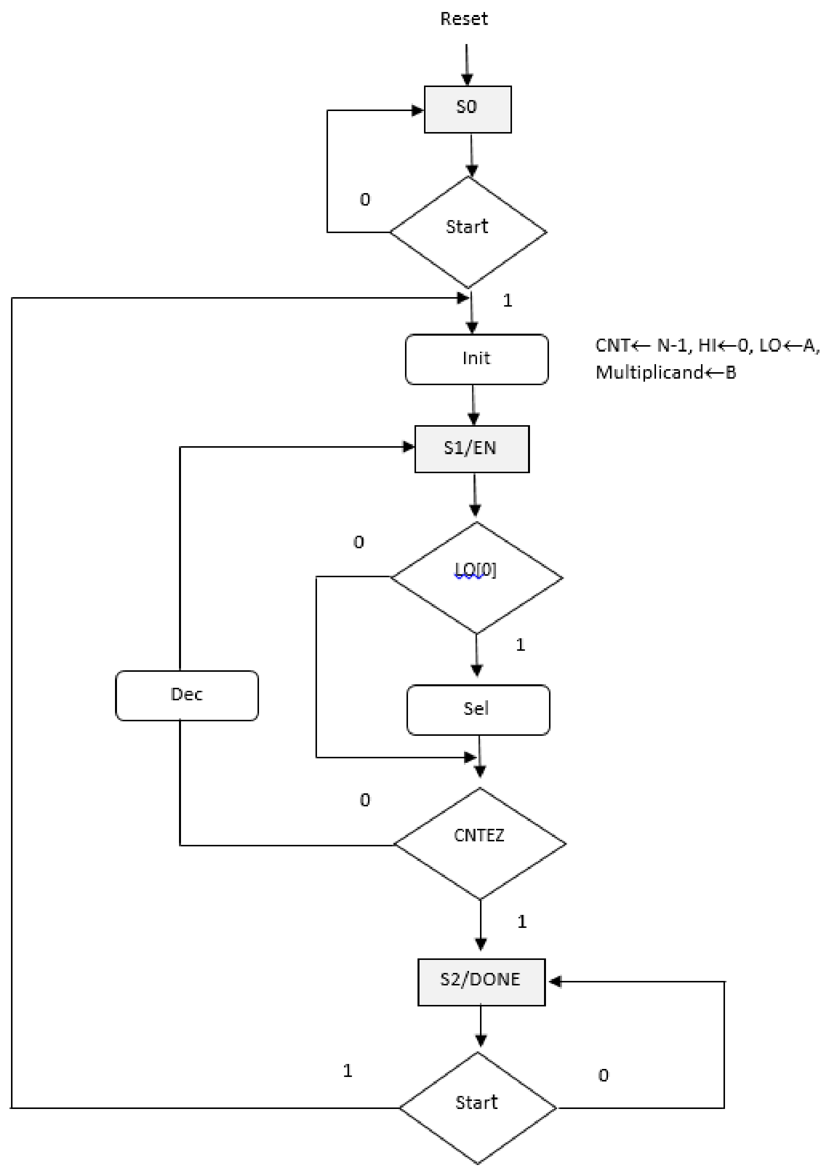


**(iii)** Implement your combinational signed multiplication circuit in FPGA and demonstrate its correct functionality. Include a video link that demonstrates the correct functionality of your implemented combinational signed multiplication circuit.

**(iv)** Next, consider implemting the signed multiplication algorithm as a sequential circuit. Show the design of the datapath components and their control signals.



**(v)** Derive the ASMD chart of the control unit that will control the operation of the signed multiplication circuit.



**(vi)** Model the data path of the sequential signed multiplication circuit in Verilog using a parametrizable module.

```
module SMUL_DP #(parameter n=4)(output reg [n-1:0] HI, LO,
output EZ, input [n-1:0] A, B, input Init, Dec, Sel, EN, CLK);

// Down Counter

reg [log2(n-1):0] DCNT;
```

```
assign EZ = ~|DCNT;

always @(posedge CLK)

if (Init) DCNT = n-1;
else if (Dec) DCNT = DCNT - 1;


// Multiplicand Register

reg [n-1:0] Multiplicand;


always @(posedge CLK)

if (Init) Multiplicand= B;


// MUX for ADDition/ Subtraction

wire [n-1:0] X;
assign X = EZ? ~Multiplicand: Multiplicand;

// Adder/Subtracter

wire [n-1:0] Y;
assign Y = HI + X + EZ;

// MUX for Selection between HI and result of ADDition/
Subtraction

wire [n-1:0] Z;
assign Z = Sel? Y : HI;

// Overflow and Sign Computation

wire OVF, Sign;
assign OVF = (HI[n-1] ~^ X[n-1]) & (HI[n-1] ^ Z[n-1]);
assign Sign = Z[n-1] ^ OVF;


// HI & LO Registres

always @(posedge CLK)

if (Init) begin
  HI = 0;
  LO = A;
end
else if (EN)  {HI, LO} = {Sign, Z, LO[n-1:1]};


function integer log2 (input integer n);
      integer i;
      begin
   log2 = 1;
   for (i=0; 2**i<n; i=i+1)
```

```
        log2 = i+1;
       end
endfunction

endmodule
```

**(vii)** Model the control unit of the sequential signed multiplication circuit in Verilog based on your derived ASM chart in (v).

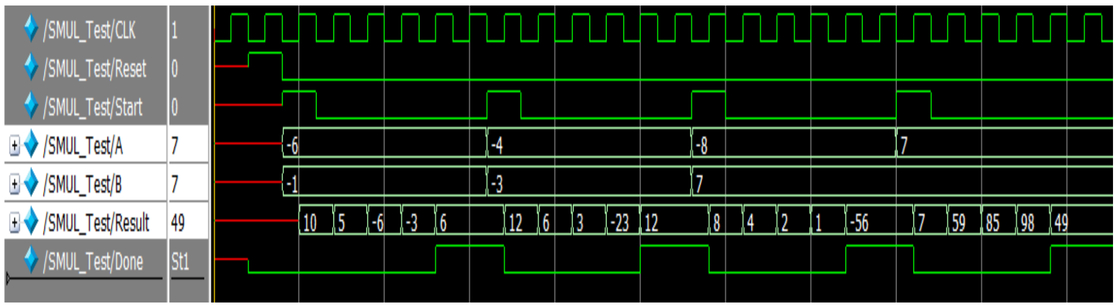
```
module SMUL_CU (output reg Init, Dec, Sel, EN, Done, input
Start, EZ, LOZ, CLK, Reset);

parameter S0 = 2'b00, S1=2'b01, S2=2'b10;

reg [1:0] state, next_state;

always @(posedge CLK, posedge Reset)
   if (Reset) state <= S0;
   else state <= next_state;

always @(state, Start, EZ, LOZ) begin
   Init=0; Dec=0; Sel=0; Dec=0; EN=0; Done=0;
   case (state)
      S0:
      if (Start)begin
         Init = 1;
         next_state=S1;
      end
      else next_state=S0;
   S1: begin EN=1;
            if (LOZ) Sel=1;
            if (EZ) next_state=S2;
            else begin Dec=1; next_state=S1; end
          end
   S2: begin Done=1;
           if (Start)begin
              Init = 1;
              next_state=S1;
           end
           else next_state=S2;
          end
   default: begin
              Init=1'bx; Dec=1'bx; Sel=1'bx; Dec=1'bx;
               EN=1'bx; Done=1'bx;
              next_state=2'bx;
            end

endcase

end

endmodule
```

**(viii)** Write a test bench to test the correct functionality of your sequential signed multiplication circuit by instantiating a 4-bit multiplier and applying the following sets of inputs: -6*-1, -4*-3, -8*+7,+7*+7.

```
module SMUL #(parameter n=4) (output [2*n-1:0] Result, output
Done, input [n-1:0] A, B, input Start, CLK, Reset);


SMUL_DP #(4) M1 (Result[2*n-1:n], Result[n-1:0], EZ, A, B,
Init, Dec, Sel, EN, CLK);


SMUL_CU M2 (Init, Dec, Sel, EN, Done, Start, EZ, Result[0],
CLK, Reset);


endmodule




module SMUL_Test();

wire [7:0] Result;
wire Done;
reg [3:0] A, B;
reg Start, CLK, Reset;

SMUL #(4) M1 (Result, Done, A, B, Start, CLK, Reset);

initial begin
CLK=0;
forever
#100 CLK = ~ CLK;
end

initial begin

// Resetting the machine

@(negedge CLK) Reset=1;

// A=-6 B=-1
@(negedge CLK) Reset=0; Start=1; A=-6; B=-1;
@(negedge CLK) Start=0;
@(negedge CLK) ;
@(negedge CLK) ;
@(negedge CLK) ;
@(negedge CLK) ;

// A=-4 B=-3
@(negedge CLK) Start=1; A=-4; B=-3;
```

```
@(negedge CLK) Start=0;
@(negedge CLK) ;
@(negedge CLK) ;
@(negedge CLK) ;
@(negedge CLK) ;

// A=-8 B=+7
@(negedge CLK) Start=1; A=-8; B=+7;
@(negedge CLK) Start=0;
@(negedge CLK) ;
@(negedge CLK) ;
@(negedge CLK) ;
@(negedge CLK) ;

// A=+7 B=+7
@(negedge CLK) Start=1; A=+7;
@(negedge CLK) Start=0;
@(negedge CLK) ;
@(negedge CLK) ;
@(negedge CLK) ;
@(negedge CLK) ;

end

endmodule
```

As can be seen from the simulation results below, the sequential multiplier is working correctly and is producing the correct results as expected.



**(ix)** Implement your sequential signed multiplication circuit in FPGA and demonstrate its correct functionality. Include a video link that demonstrates the correct functionality of your implemented sequential signed multiplication circuit.