**ELEC 4200 Lab 10 Report**

**Xiaodong Song**

**4/15/2020**

**Goal for the Lab**

For lab 10, students would try to use ASM charts to design sequential systems. There were two tasks for students to practice. ASM means Algorithmic State Machine. It’s better to do a flowchart as requirement. After doing this lab, students would be more familiar with the ASM method to do some complicate program.

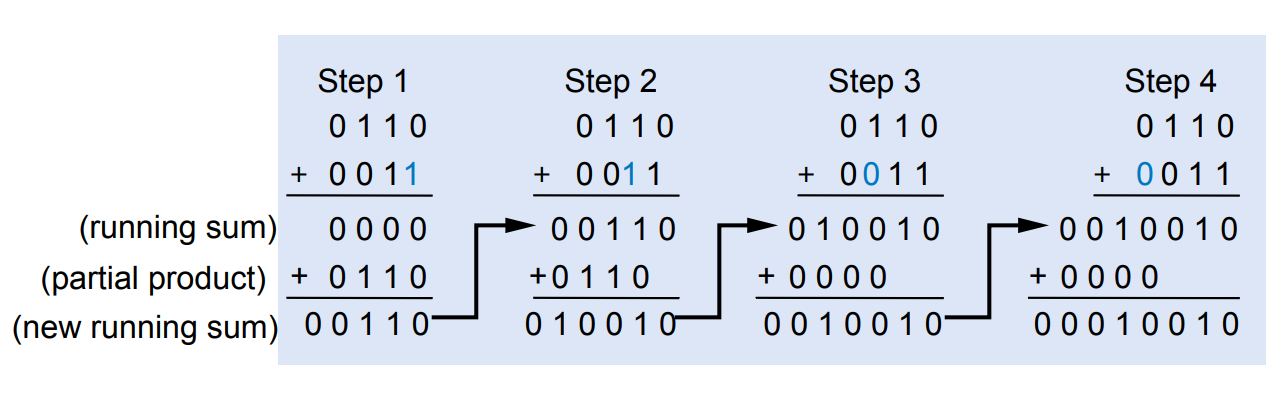
**Design for the Lab**

For lab 10-1-1, students would try to design a 3-bit x 3-bit binary multiplier. Before doing this task, students have to know the method to let two numbers multiply each other by adder and shifter.

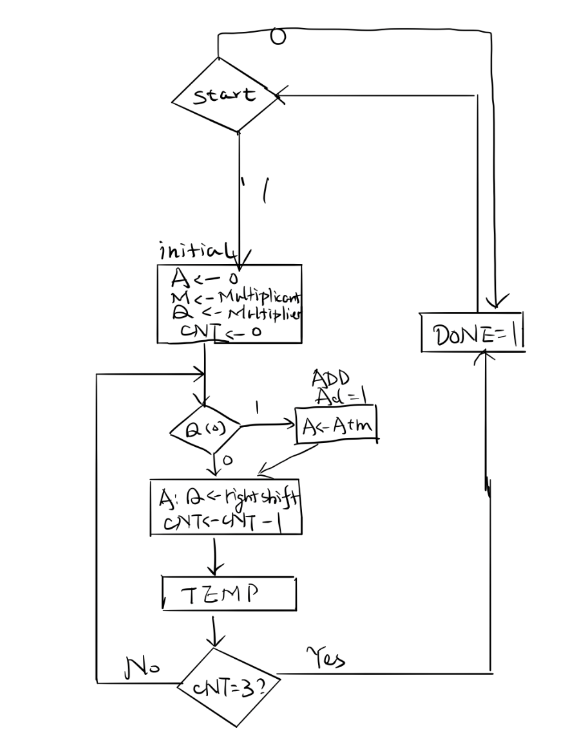
For lab 10-2-1, students would modify the 1-2’s program which should be same as 1-1 to perform a 4-bit x 4-bit unsigned multiplication. And students would store the multiplicand and multipliers in 32x4 ROM.

**Detailed Design**

For lab 10-1-1, this multiplier would output 6-bit produce. The data processor unit would consist of a 3-bit accumulator, a 3-bit multiplier register, a 3-bit adder, a counter, and a 3-bit shifter. The control unit will consist of a least-significant-bit (lsb) of the multiplier. A start signa, a done signal, and clock as an input. The program’s calculation process should be as following (**Figure 1**). And the ASM chart should be as **Figure 2** shown:



**Figure 1:** Calculation Process

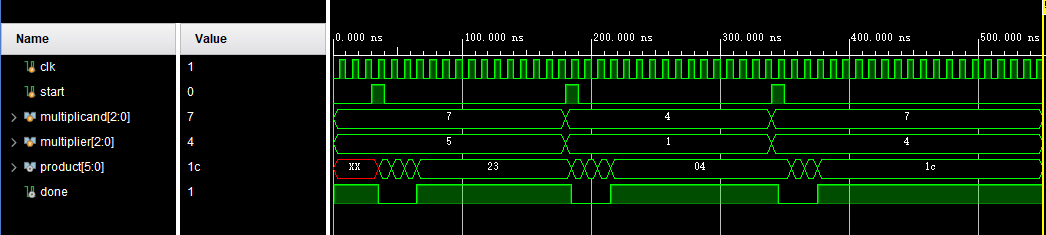


**Figure 2:** ASM Chart for Lab 10-1-1

For lab 10-2-1, students would modify the design of lab 10-1-1 to perform a 4-bit x 4-bit unsigned multiplication. Students would store the 4-bit multiplicand and multipliers in a 32x4 ROM. The first 16 locations holding multiplicands and the other 16 locations holding multipliers which means that a .txt file should contain 16 multiplicand numbers and 16 multiplier numbers which are both 0,1,2,3,…,F. The multiplicand and multiplier operand addresses should be input by testbench. After that, students still need to define 100 MHz clock in the testbench and use it for making an input clock in the Clocking Wizard. Use the 100 MHz clock source to generate a 5MHz clock and the appropriate clock divider circuit to generate further clock of 10 KHz. The ASM would be similar with the lab 10-1-1’s ASM chart. What students need to do were to create a IP Catalog’s Clocking Wizard to generate a 5MHz and input the ROM .txt file’s number to let them multiply them each other.

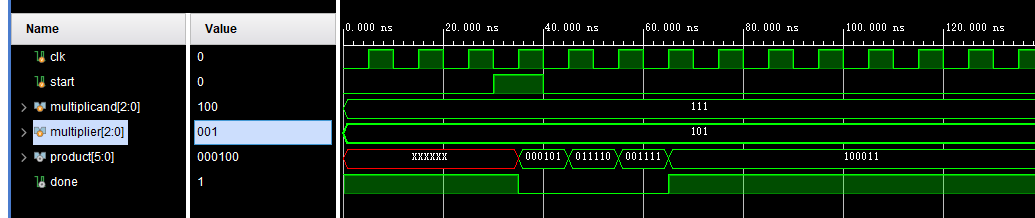
**Design Verification**

For lab 10-1-1, the simulation’s result was shown as below, as the **Figure 3** showing, the initial input 3-bit multiplicand number is 7 which’s binary number is 111 and 3-bit multiplier number is 5 which’s binary number is 101. For the multiplication between 111 and 101, the binary number result should be 010 011, which’s hexdecimal number is 2 3. And this is same with the next two input.



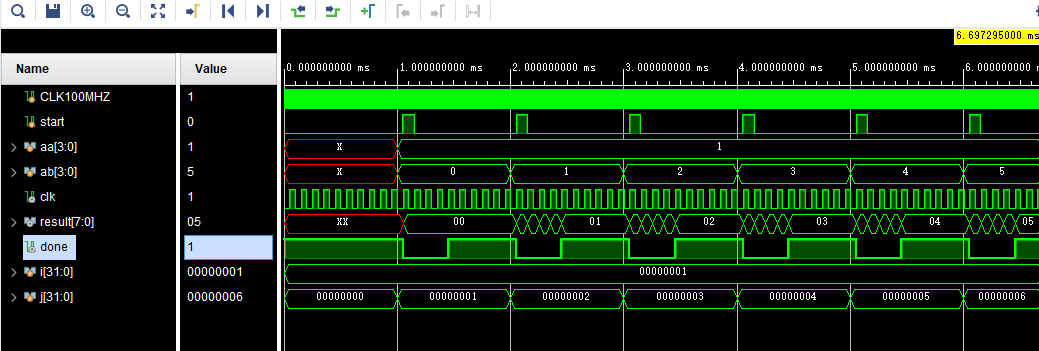
**Figure 3:** Lab 10\_1\_1’s simulation

And the following figure is the zoom-in simulation for the multiplication between 111 and 101, as we see, the first product we got is the 000101 which should be product by 1 x 101, and the next one is 011110 which should be product by “1 x 101” plus shifted “1 x 101”. And the 001111 also should be a added by the last one product and the 1 x 101. Then the final result 100011 should be got from last numbers by shifting and adding.



**Figure 4:** Lab 10\_1\_1’s Zoom-in Simulation

For lab 10-2-1, the simulation result was shown below. The input and output result are 4-bit and 8-bit, respectively. As the figures showing, the multiplicands(aa) start from 1 to multiply the multipliers. And for the multipliers(ab), it starts from 0 to F to be multiplied one by one by multiplicands. And the way this multiplier is working is same with the lab 10-1-1. The input signal “start” determines when the calculation for multiplication starts and the done represents the moment when the calculation for multiplication ends



**Figure 5:** Lab 10\_2\_1’s Simulation

**Conclusion**

After doing this lab, students should be more familiar with how to use ASM for figuring out how let a complication program work. And for the lab 10-2-1, this is a combinational practice of ROM, IP Catalog and ASM. This kind of combinational practice can be very helpful to students understand the real usefulness of them.

**Appendix**

module lab10\_1\_1(

input clk, start,

input [2:0] a, b,

output done,

output [5:0] r);

wire c\_start, c\_add, c\_shift, c\_done;

multiplier\_control controller (clk, start, c\_start, c\_add, c\_shift, c\_done);

multiplier data\_processor (clk, a, b, c\_start, c\_add, c\_shift, c\_done, r);

assign done = c\_done;

endmodule

module multiplier\_control(

input clk, start,

output reg s\_start, s\_add, s\_shift, s\_done);

parameter DONE = 0, START = 15,

A1 = 1, S1 = 2, A2 = 3, S2 = 4, A3 = 5, S3 = 6;

reg [3:0] state, nextstate;

initial begin

state = DONE;

nextstate = DONE;

end

always @(posedge clk) begin

state <= nextstate;

end

always @(state) begin

case(state)

DONE: {s\_start, s\_add, s\_shift, s\_done} <= 4'b0001;

START: {s\_start, s\_add, s\_shift, s\_done} <= 4'b1000;

A1: {s\_start, s\_add, s\_shift, s\_done} <= 4'b0100;

A2: {s\_start, s\_add, s\_shift, s\_done} <= 4'b0100;

A3: {s\_start, s\_add, s\_shift, s\_done} <= 4'b0100;

S1: {s\_start, s\_add, s\_shift, s\_done} <= 4'b0010;

S2: {s\_start, s\_add, s\_shift, s\_done} <= 4'b0010;

S3: {s\_start, s\_add, s\_shift, s\_done} <= 4'b0010;

endcase

end

always @(start or state) begin

if (state == DONE) begin

if (start)

nextstate <= START;

else

nextstate <= DONE;

end

else begin

// ignore [start] signal

case(state)

START: nextstate <= A1;

A1: nextstate <= S1;

S1: nextstate <= A2;

A2: nextstate <= S2;

S2: nextstate <= A3;

A3: nextstate <= S3;

S3: nextstate <= DONE;

endcase

end

end

endmodule

module multiplier(

input clk,

input [2:0] a, b,

input start, add, shift, done,

output [5:0] result);

reg [6:0] r; // result register, 1 bit longer

reg [3:0] shl; // shift left

assign result = r[5:0];

wire [2:0] add\_a, add\_b, add\_s;

wire add\_ci, add\_co;

adder #3 adder\_core (add\_a, add\_b, add\_s, add\_ci, add\_co);

assign add\_a = a & {3{b[shl]}}; // a or zero

assign add\_b = (r >> shl) & 3'h7;

assign add\_ci = 0;

always @(posedge clk) begin

if (start) begin

shl <= 0;

r <= 7'b0;

end

else if (add) begin

r <= (r & ~(7'b1111 << shl)) | ({3'h0, add\_co, add\_s} << shl);

end

else if (shift) begin

shl <= shl + 1;

end

else if (done) begin

// boo

end

end

endmodule

module adder #(parameter WIDTH = 1)(

input [WIDTH-1:0] a, b,

output [WIDTH-1:0] s,

input ci,

output co);

assign {co, s} = {1'b0, a} + {1'b0, b} + ci;

endmodule

module lab10\_1\_1\_tb;

reg clk, start;

reg [2:0] a, b;

wire done;

wire [5:0] s;

lab10\_1\_1 DUT (clk, start, a, b, done, s);

initial begin

clk = 0; start = 0; a = 3'b111; b = 3'b101;

#30 start = 1;

#10 start = 0;

#100 start = 1; a = 3'b100; b = 3'b001;

#10 start = 0;

#100 start = 1; a = 3'b111; b = 3'b100;

#10 start = 0;

#240 $finish();

end

always #5 clk <= ~clk;

endmodule

module lab10\_2\_1(

input CLK100MHZ, start,

input [3:0] aa, ab,

output done,

output [7:0] result,

output CLK10kHZ

);

wire CLK5MHZ;

clk\_wiz\_0 clock\_converter (CLK5MHZ, CLK100MHZ);

clk\_div div (CLK5MHZ,CLK10kHZ);

reg [3:0] D\_ROM [31:0];

wire [3:0] a, b;

wire c\_start, c\_add, c\_shift, c\_done;

assign done = c\_done;

assign a = D\_ROM[aa];

assign b = D\_ROM[5'h10 | ab];

control controller (CLK10kHZ, start, c\_start, c\_add, c\_shift, c\_done);

multiplier data\_processor (CLK10kHZ, a, b, c\_start, c\_add, c\_shift, c\_done, result);

initial

$readmemh("ROM\_data.txt", D\_ROM, 0, 31);

endmodule

module clk\_div(

input clkIn,

output Clk);

wire clock\_in;

reg[27:0] counter=28'd0;

parameter DIVISOR = 28'd500;

always @(posedge clkIn)

begin

counter <= counter + 28'd1;

if(counter>=(DIVISOR-1))

counter <= 28'd0;

end

assign Clk = (counter<DIVISOR/2)?1'b0:1'b1;

endmodule

module control(

input clk, start,

output reg s\_start, s\_add, s\_shift, s\_done

);

parameter DONE = 0, START = 15,

A1 = 1, S1 = 2, A2 = 3, S2 = 4, A3 = 5, S3 = 6, A4 = 7, S4 = 8;

reg [3:0] state, nextstate;

initial begin

state = DONE;

nextstate = DONE;

end

always @(posedge clk) begin

state <= nextstate;

end

always @(state) begin

case(state)

DONE: {s\_start, s\_add, s\_shift, s\_done} <= 4'b0001;

START:{s\_start, s\_add, s\_shift, s\_done} <= 4'b1000;

A1: {s\_start, s\_add, s\_shift, s\_done} <= 4'b0100;

A2: {s\_start, s\_add, s\_shift, s\_done} <= 4'b0100;

A3: {s\_start, s\_add, s\_shift, s\_done} <= 4'b0100;

A4: {s\_start, s\_add, s\_shift, s\_done} <= 4'b0100;

S1: {s\_start, s\_add, s\_shift, s\_done} <= 4'b0010;

S2: {s\_start, s\_add, s\_shift, s\_done} <= 4'b0010;

S3: {s\_start, s\_add, s\_shift, s\_done} <= 4'b0010;

S4: {s\_start, s\_add, s\_shift, s\_done} <= 4'b0010;

endcase

end

always @(start or state) begin

if (start)

nextstate <= START;

else begin

case(state)

DONE: nextstate <= DONE;

START: nextstate <= A1;

A1: nextstate <= S1;

S1: nextstate <= A2;

A2: nextstate <= S2;

S2: nextstate <= A3;

A3: nextstate <= S3;

S3: nextstate <= A4;

A4: nextstate <= S4;

S4: nextstate <= DONE;

endcase

end

end

endmodule

module multiplier(

input clk,

input [3:0] a, b,

input start, add, shift, done,

output [7:0] result);

reg [7:0] r; // result register, 1 bit longer

assign result = r;

reg [3:0] shl; // shift left

assign result = r[5:0];

wire [3:0] add\_a, add\_b, add\_s;

wire add\_ci, add\_co;

adder #4 adder\_core (add\_a, add\_b, add\_s, add\_ci, add\_co);

assign add\_a = a & {4{b[shl]}}; // a or zero

assign add\_b = (r >> shl) & 4'hF;

assign add\_ci = 0;

always @(posedge clk) begin

if (start) begin

shl <= 0;

r <= 5'b0;

end

else if (add) begin

r <= (r & ~(8'b11111 << shl)) | ({3'h0, add\_co, add\_s} << shl);

end

else if (shift) begin

shl <= shl + 1;

end

else if (done) begin

// boo

end

end

endmodule

module adder #(parameter WIDTH = 1)(

input [WIDTH-1:0] a, b,

output [WIDTH-1:0] s,

input ci,

output co);

assign {co, s} = {1'b0, a} + {1'b0, b} + ci;

endmodule

module lab10\_2\_1\_tb;

reg CLK100MHZ,start;

reg [3:0] aa,ab;

wire done,clk;

wire [7:0] result;

integer i=0,j=0;

lab10\_2\_1 DUT (CLK100MHZ,start,aa,ab,done,result,clk);

initial begin

CLK100MHZ = 0;

forever #5 CLK100MHZ = ~CLK100MHZ;

end

initial begin

start = 0;

for(i=1;i<= 10;i=i+1)begin

for(j=0;j<=10;j=j+1)begin

#1000000 aa=i; ab = j;

end

end

end

initial begin

#150000;

forever begin

#900000 start = 1;

#100000 start = 0;

end end

endmodule

0 1 2 3 4 5 6 7 8 9 A B C D E F

0 1 2 3 4 5 6 7 8 9 A B C D E F