CS 207 Digital Logic - Spring 2020
Lab 12

James Yu

May 6, 2020

Objective

1. Implement other arithmetic circuits.

Lab Exercise Submission

1. You should name all source code as instructed. Mis-named files will not be recognized.
2. You should submit all source code files with an extension "`.v`".
3. You should submit all source code directly into the sakai system below. **Do not compress them into one folder.**

   `https://sakai.sustech.edu.cn/portal/site/ebf68254-68c5-4cfe-9d42-b26758f854ee`
4. Lab exercises should be submitted before the deadline, typically one week after the lab session. No late submission policy applies to lab exercises.

1 Binary Multiplier

During the lecture, we introduced the digital implementation of a binary multiplier. In this lab, we provide the Verilog implementation of binary multipliers in three different modeling ways.

`mul_gate.v`

```
module mul_gate(A, B, C);
output[3:0] C;
input[1:0] A, B;
wire w1, w2, w3, w4;

```

```
and and1(C[0], A[0], B[0]);
and and2(w1, A[0], B[1]);
and and3(w2, A[1], B[0]);
half_adder myadd1(w1, w2, C[1], w3);
and and4(w4, A[1], B[1]);
half_adder myadd2(w4, w3, C[2], C[3]);
endmodule

module half_adder(A, B, S, C);
input A, B;
output S, C;
and and1(C, A, B);
xor xor1(S, A, B);
endmodule
```

mul_behav.v

```
module mul_behav(A, B, C);
output[3:0] C;
input[1:0] A, B;
reg[3:0] C;
always @(A or B)
begin
  case({A,B})
    4'h0 : C = 4'b0000;
    4'h1 : C = 4'b0000;
    4'h2 : C = 4'b0000;
    4'h3 : C = 4'b0000;
    4'h4 : C = 4'b0000;
    4'h5 : C = 4'b0001;
    4'h6 : C = 4'b0010;
    4'h7 : C = 4'b0011;
    4'h8 : C = 4'b0000;
    4'h9 : C = 4'b0010;
    4'ha : C = 4'b0100;
    4'hb : C = 4'b0110;
    4'hc : C = 4'b0000;
    4'hd : C = 4'b0011;
    4'he : C = 4'b0110;
    4'hf : C = 4'b1001;
    default : C='bx;
  endcase
end
endmodule
```

mul_data.v

```
module mul_data(A, B, C);
output[3:0] C;
input[1:0] A, B;
assign C = A * B;
endmodule
```

All three implementations can share the same testbench as follows:

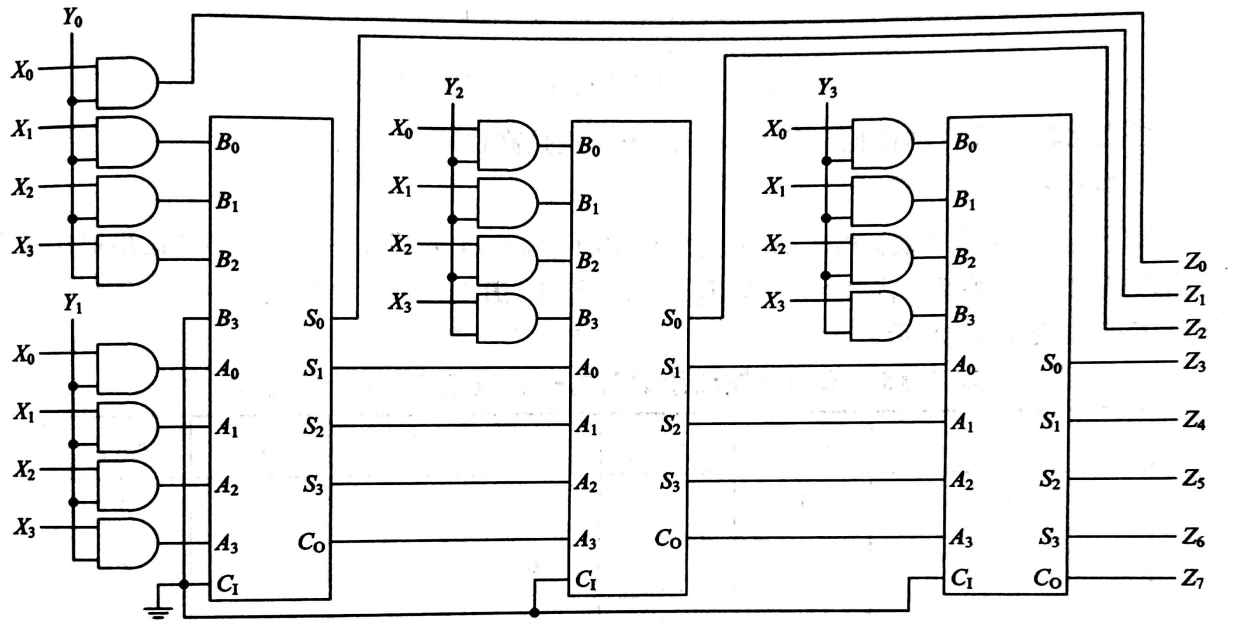
mul_tb.v

```
module mul_tb;
reg [1:0] A, B;
wire [3:0] Z;
mul_gate mult1(A, B, Z);
# mul_behav mult1(A, B, Z);
# mul_data mult1(A, B, Z);

initial begin
  $monitor("%3t: A is %d, B is %d, Z is %d",$time,A,B,Z);
  # 5 A=0;B=0;
  # 5 A=1;B=0;
  # 5 A=2;B=0;
  # 5 A=3;B=0;

  # 5 A=0;B=1;
  # 5 A=1;B=1;
  # 5 A=2;B=1;
  # 5 A=3;B=1;

  # 5 A=0;B=2;
  # 5 A=1;B=2;
  # 5 A=2;B=2;
  # 5 A=3;B=2;

  # 5 A=0;B=3;
  # 5 A=1;B=3;
  # 5 A=2;B=3;
  # 5 A=3;B=3;
  # 10 $finish;
end
endmodule
```

2 Exercise

2.1 Exercise 1

Implement a 4-bit binary multiplier with 4-bit adders and logic gates. The inputs are X and Y, and the circuit outputs Z. You can use any modeling way for the 4-bit adder. However, only gate level design is allowed outside the adders. A sample circuit design is shown as follows:



The module should follow the design below:

mul_4.v

```
module mul_4(X, Y, Z);
// Module Code
endmodule
```

and save in file `mul_4.v`.

> **Assignment**
> Save the source code in **`mul_4.v`**. Upload this file to Sakai under **Assignments → Lab Exercise 12**.

2.2 Exercise 2

Implement an 8-bit binary multiplier. The inputs are 8-bit X and Y, and the circuit outputs Z. You cannot use keyword `assign` in the code except for inside adders. The module should follow the design

below:

mul_8.v

```
module mul_8(X, Y, Z);
// Module Code
endmodule
```

and save in file `mul_8.v`.

Assignment

Save the source code in **`mul_4.v`**. Upload this file to Sakai under **Assignments → Lab Exercise 12**.

Assignment

When you finished Lab Exercise 11, there should be two `.v` files in the Sakai system: **`mul_4.v`** and **`mul_8.v`**.