CPE166 Lab 2 Part 1
By: Prof. Pang

Lab 2

Part 1: 3 By 3 Binary Combinational Array Multiplier

The binary combinational multiplier diagram:

$$\begin{array}{cccccc}
 & & & X_2 & X_1 & X_0 \\
 & & \mathbf{x} & Y_2 & Y_1 & Y_0 \\
\cline{3-6}
 & & & X_2Y_0 & X_1Y_0 & X_0Y_0 \\
 & & X_2Y_1 & X_1Y_1 & X_0Y_1 & \\
 & X_2Y_2 & X_1Y_2 & X_0Y_2 & & \\
\hline
P_5 & P_4 & P_3 & P_2 & P_1 & P_0
\end{array}$$

Figure 1. 3 by 3 binary combinational multiplication diagram

This lab is to design the above multiplier by using the hardware structure shown below:

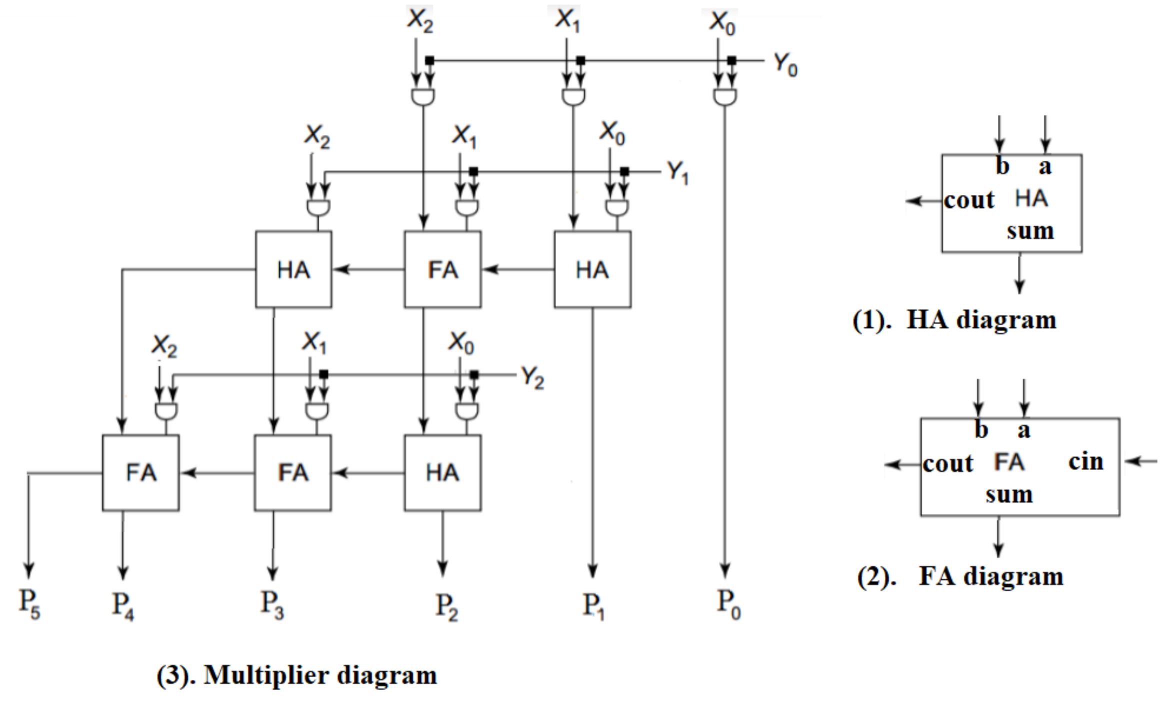


Figure 2. 3 by 3 binary combinational array multiplier hardware structure

CPE166 Lab 2 Part 1
By: Prof. Pang

Lab Procedure

Step 1. Half Adder Design

The half adder adds two 1-bit binary inputs a and b. It generates two outputs, sum signal and carry cout signal.

Inputs		Outputs	
a	b	cout	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Logic equations:

$$\text{cout} = a\,b$$

$$\text{sum} = a \oplus b$$

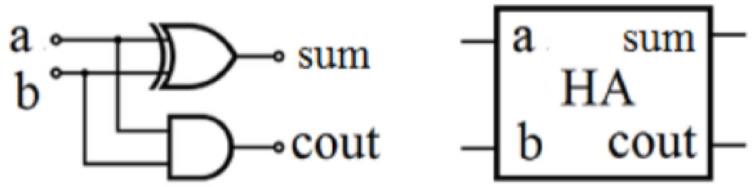


Figure 3. Half adder truth table, schematic and logic symbol

Design the above half adder circuit using Verilog, write testbench and run simulations. Verify that your simulation results are the same as the values listed in the truth table above.

CPE166 Lab 2 Part 1
By: Prof. Pang

Step 2. Full Adder Design

The full adder circuit adds three 1-bit binary inputs a, b and cin. It generates two outputs, sum signal and carry cout signal.

Inputs			Outputs	
a	b	cin	cout	sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1
Logic equations: $\text{sum} = a \oplus b \oplus \text{cin}$ $\text{cout} = (a \oplus b)\,\text{cin} + a\,b$				

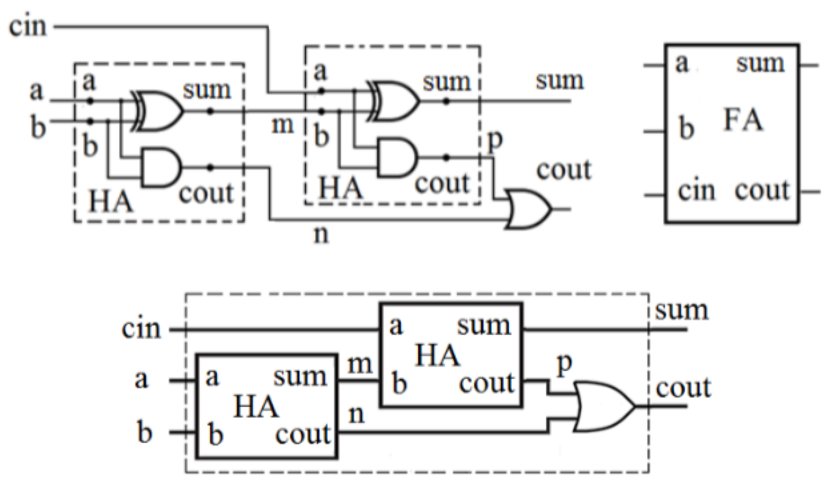
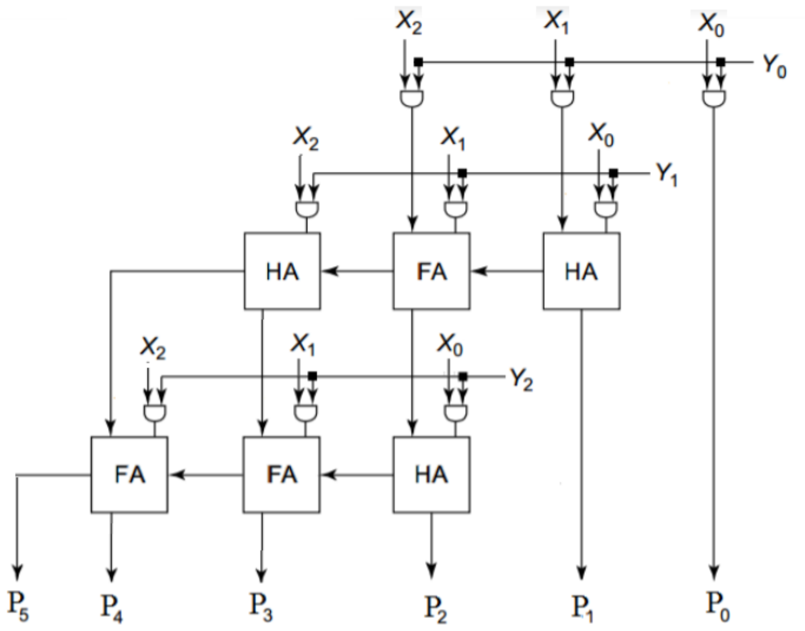


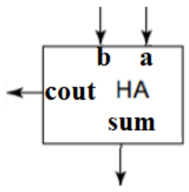
Figure 4. Full adder truth table, schematic and logic symbol

Design the above full adder circuit by using two half adder HA modules, and one OR gate in Verilog, write testbench and run simulations. Verify that your simulation results are the same as the values listed in the truth table above.

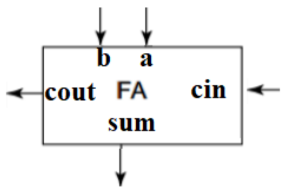
Step 3. Final Combinational Multiplier Design



(3). Multiplier diagram



(1). HA diagram



(2). FA diagram

Figure 5. 3 by 3 combinational array multiplier schematic

Design the above multiplier circuit by using nine AND gates, three half adder HA modules and three full adder FA modules in Verilog, write a testbench and run simulation.

Demo Requirement

You need to demonstrate the final simulation waveform of the multiplier design to your lab instructor.

CPE166 Lab 2 Part 1
By: Prof. Pang

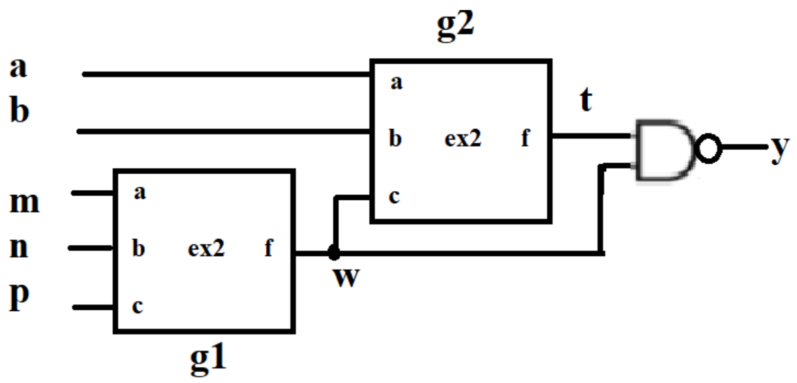
Note: Before starting this experiment, all the necessary knowledge required to complete this work has been introduced in the CPE166 lecture session. The following examples are for you to refresh your learning.

Sample Verilog Codes:

```
module ex1( a, b, f1, f2, f3, f4);

input a, b;

output f1, f2, f3, f4;

assign f1 = a ^ b;              // xor gate

assign f2 = a ~^ b;             // xnor gate

assign f3 = ~ ( a & b );        // nand gate

assign f4 = ~ (a | b);          // nor gate

endmodule


module ex2( a, b, c,  f );

input a, b, c;

wire    m, n;

output f;

assign  m  = ~a ;               // not gate

assign  n=   b & c ;            // and  gate

assign  f  =  m |  n;           // or gate

endmodule


`timescale  1ns/1ps

module ex2_tb;           //testbench for ex2 design

reg      a, b, c;

wire     f;               // Only a, b, c and f are used for testing.

integer k;

ex2    g1(a, b, c, f);
```

```
initial

begin

   $monitor($time, " ns, a=%b, b=%b, c=%b, f=%b", a, b, c, f);

    for (k=0; k<8; k=k+1)

    begin

        {a, b, c} = k;

         #5;

     end

     #5  $stop;

end

endmodule
```

Sample Circuit:



Sample Implementation in Verilog:

```
module sample (a, b, m, n, p, y);

input    a, b, m, n, p;

output    y;

wire    w,  t;

ex2    g1 ( .a (m),  .b(n), .c(p), .f(w) );

ex2    g2  (.a (a),   .b(b),  .c(w), .f (t) );

assign  y =  ~ ( w & t);

endmodule
```

CPE166 Lab 2 Part 2
By: Prof. Pang

Lab 2

Part 2: 8-bit Carry Select Adder

The purpose of this experiment is to design the 8-bit carry select adder circuit shown below.

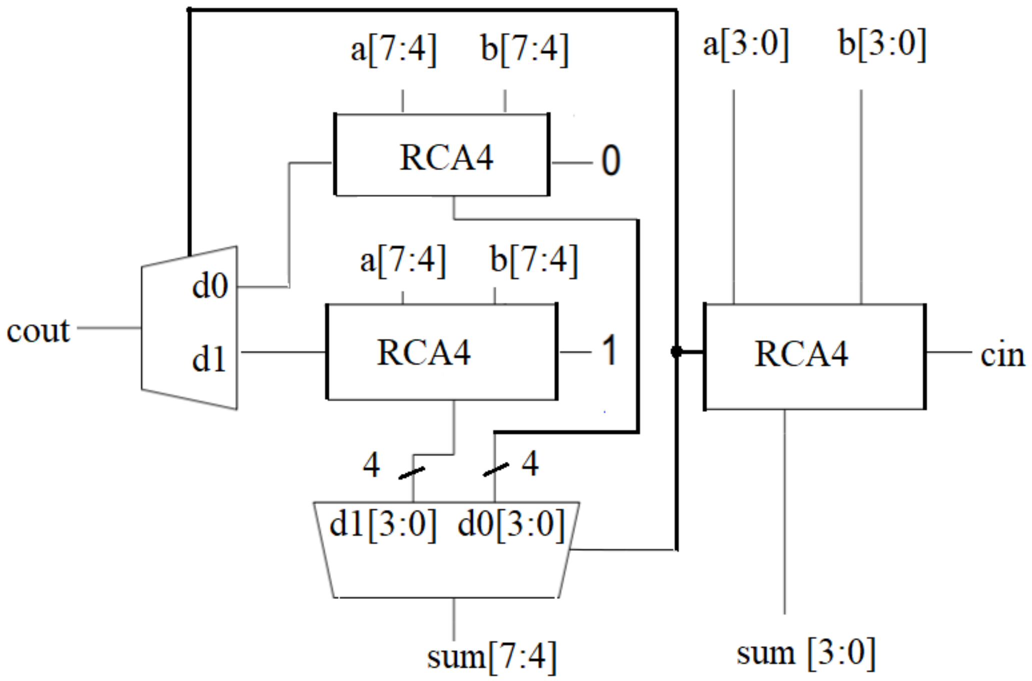


Figure 1. 8-bit carry-select adder circuit

The 8-bit carry-select adder circuit above consists of three 4-bit ripple carry adders (RCA4) and two multiplexers. One of 4-bit ripple carry adders assumes the carry-in to be zero, and the other assumes the carry-in to be one. The multiplexers select the correct sum and the carry-out based on the known carry-in value. This method is faster than the 8-bit ripple carry adder approach to obtain the 8-bit addition results.

Lab Procedure

<u>Step 1. 4-bit Ripple Carry Adder Design</u>

The 4-bit ripple carry adder circuit is shown below.

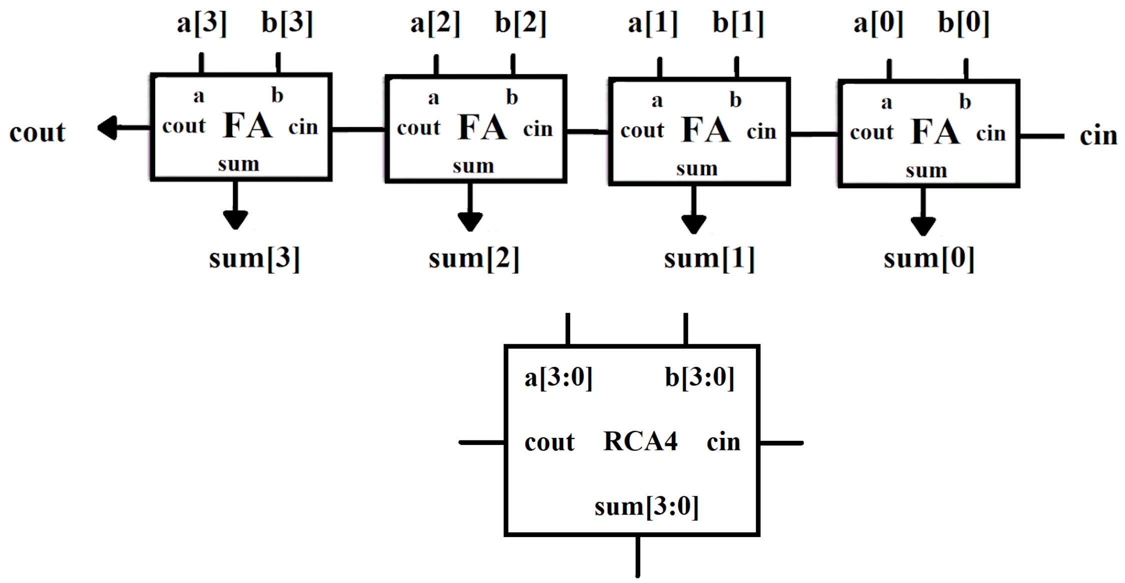


Figure 2. The 4-bit ripple carry adder circuit and block diagram

You can reuse the full adder design module FA you designed in the Lab2 part 1 for your RCA4 design in this part.

Therefore, you need to include the half adder ha.v file, the full adder fa.v file, and the 4-bit ripple carry adder rca.4 file in the design of step 1. In addition, you need to write a testbench for the rca4 design and run simulations.

Step 2. Multiplexer (MUX) Design

The 2-to-1 multiplexer consists of two inputs D0 and D1, one selection input S and one output Y. According to the logic value of the selection signal S, D0 or D1 will be passed to the output.

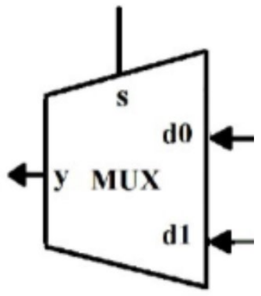


Figure 3. 2-to-1 multiplexer diagram

A **Verilog if statement** is used to choose which statement should be executed depending on the conditional expression.

Simplified Sample Syntax

```
reg   y;
always@(signal1 or signal2 or signal3)
begin
  if (conditional expression)
    y= statement1;
  else
     y= statement2;
end
```

You can design the 2-to-1 multiplexer circuit by using if…else statement in Verilog and also write a testbench to run simulations. Verify by yourself that your simulation results match with the values shown below.

Inputs			Output
s	d0	d1	y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

The above table can also be simplified into the following format.

s	y
0	d0
1	d1

Step 3. MUXB Design

The MUXB consists of two 4-bit inputs D0 and D1, one 1-bit selection input S and one 4-bit output Y. According to the logic value of the selection signal S, D0 or D1 will be passed to the output.

The Verilog design of this part will look very similar to the design in step 2. The major difference is that you must declare the inputs d1, d0 and the output y as 4-bit data.

Simplified Sample Syntax

```
reg [3:0]   y;
always@(signal1 or signal2 or signal3)
begin
  if (conditional expression)
    y= statement1;
  else
     y= statement2;
end
```

You can design the 2-to-1 multiplexer circuit by using if…else statement in Verilog and write a testbench to run simulations. Verify by yourself that your simulation results match with the truth table shown below.

s	y[3:0]
0	d0[3:0]
1	d1[3:0]

Step 4. Final 8-bit Carry Select Adder Design

Carry-Select Adder (CSA8) diagram:

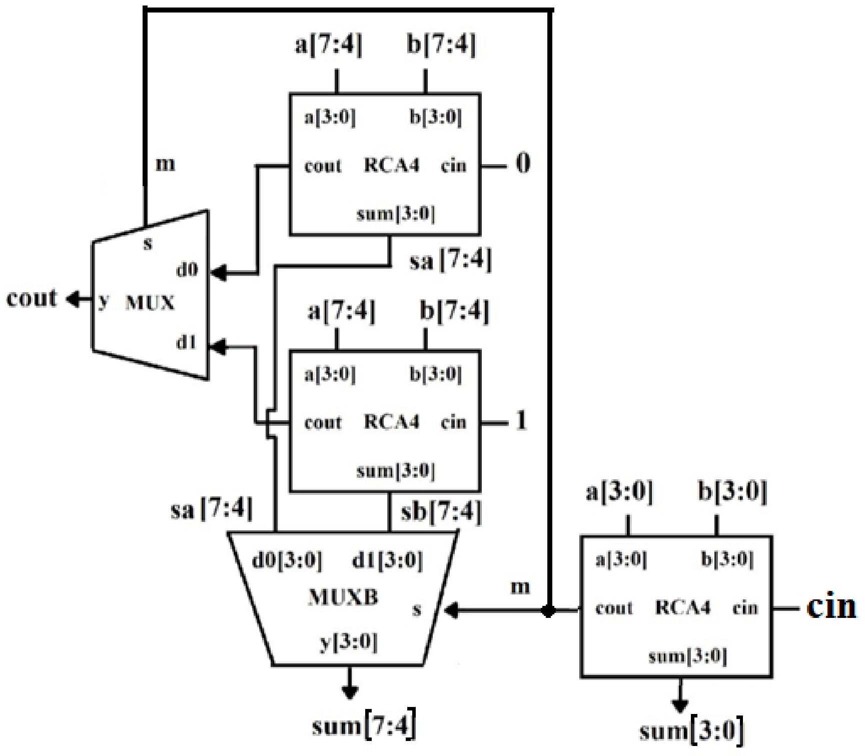


Figure 4. 8-bit carry select adder circuit

Design the 8-bit carry-select adder (CSA8) circuit above by using three 4-bit ripple carry adders (RCA4), one MUX and one MUXB. In addition, write testbench for CSA8 to run simulations.

Demo Requirement

You need to demonstrate the final simulation waveform of the CSA8 design to your lab instructor.

CPE166 Lab 2 Part 2
By: Prof. Pang

Note: Before starting this experiment, all the necessary knowledge required to complete this work has been introduced in the CPE166 lecture session. The following examples are for you to refresh your learning.

Sample Verilog Codes:

```
module mux4( d0, d1, d2, d3, s0, s1, y);

input   d0, d1, d2, d3, s0, s1;

output  y;

reg       y;

always@(d0 or d1 or d2 or d3 or s0 or s1)

begin

    if ( s1==0 && s0==0)

        y = d0;

   else if ( s1==0 && s0==1)

       y = d1;

   else if ( s1==1 && s0==0)

       y = d2;

   else

       y = d3;

end

endmodule
```

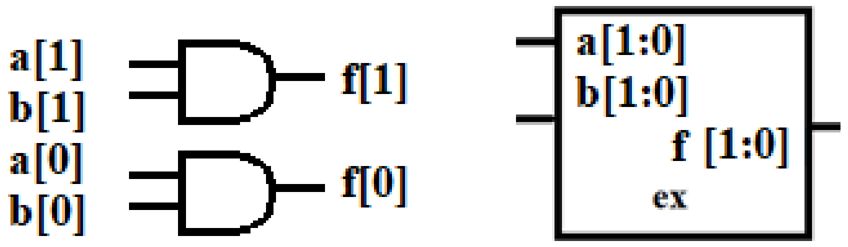
CPE166 Lab 2 Part 2
By: Prof. Pang

```
`timescale 1ns/1ps

module mux4_tb;

reg     d0, d1, d2, d3, s0, s1;

wire   y;

mux4    uut ( d0, d1, d2, d3, s0, s1, y);

initial

begin

   $monitor($time, " ns, d0=%b, d1=%b, d2=%b, d3=%b, s1=%b, s0=%b, y=%b", d0, d1, d2, d3, s1, s0, y);

     d0=0; d1=0; d2=0; d3=0;  s1=0; s0=0;

    #10    d0=1; d1=0; d2=0; d3=0;  s1=0; s0=0;

    #10    d0=0; d1=0; d2=0; d3=0;  s1=0; s0=1;

    #10    d0=0; d1=1; d2=0; d3=0;  s1=0; s0=1;

    #10    d0=0; d1=0; d2=0; d3=0;  s1=1; s0=0;

    #10    d0=0; d1=0; d2=1; d3=0;  s1=1; s0=0;

    #10    d0=0; d1=0; d2=0; d3=0;  s1=1; s0=1;

    #10    d0=0; d1=0; d2=0; d3=1;  s1=1; s0=1;

    #10    $stop;

end

endmodule
```

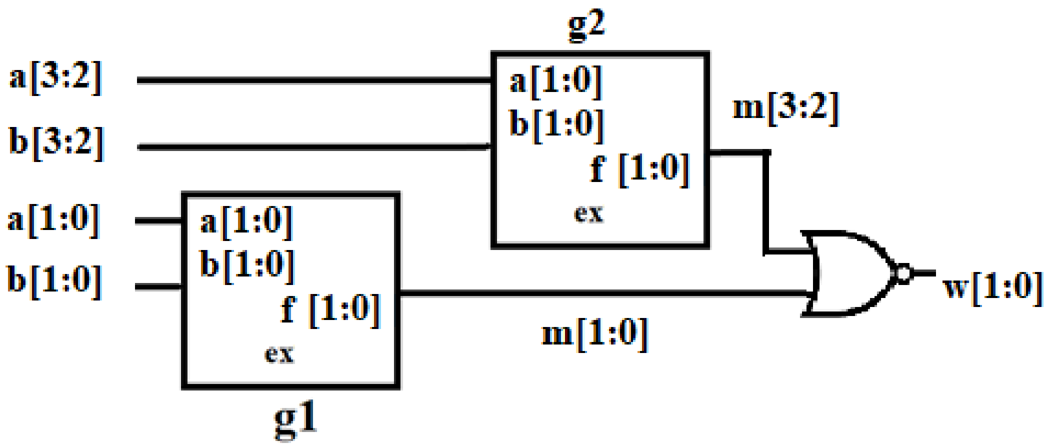
CPE166 Lab 2 Part 2
By: Prof. Pang

Sample Circuits:

(1).



(2).



Sample Implementation in Verilog:

```
module ex (a, b, f);

input    [1:0]  a, b;

output  [1:0]  f;

assign   f = a & b;

endmodule
```

CPE166 Lab 2 Part 2
By: Prof. Pang

```
`timescale 1ns/1ps

module  ex_tb;

reg   [1:0]    a, b;

wire  [1:0]   y;

ex    uut ( a, b, y);

integer   k;

initial   begin

    $monitor($time, " ns, a=%b, b=%b,  y=%b", a, b,  y);

     for (k=0; k<16; k=k+1)

     begin

         {a, b} = k;

          #5;

      end

      #5  $stop;

end

endmodule
```

```
module ex2 (a, b, w);

input    [3:0]  a, b;

output  [1:0]  w;

wire      [3:0]  m;

ex     g1 ( .a ( a[1:0] ), .b( b[1:0] ),  .f (  m[1:0] ) );

ex     g2 ( .a ( a[3:2] ), .b( b[3:2] ),  .f (  m[3:2] ) );

assign    w =   ~ ( m[3:2] |  m{1:0] );

endmodule
```

CPE166 Lab 2 Part 3
By: Prof. Pang

Lab 2

Part 3: Two-Speed BCD Counter

The binary coded decimal (BCD) counter is a serial digital counter that counts from 0 to 9 and then repeats. The purpose of this experiment is to design the BCD counter circuit shown below, which can choose two different speeds.

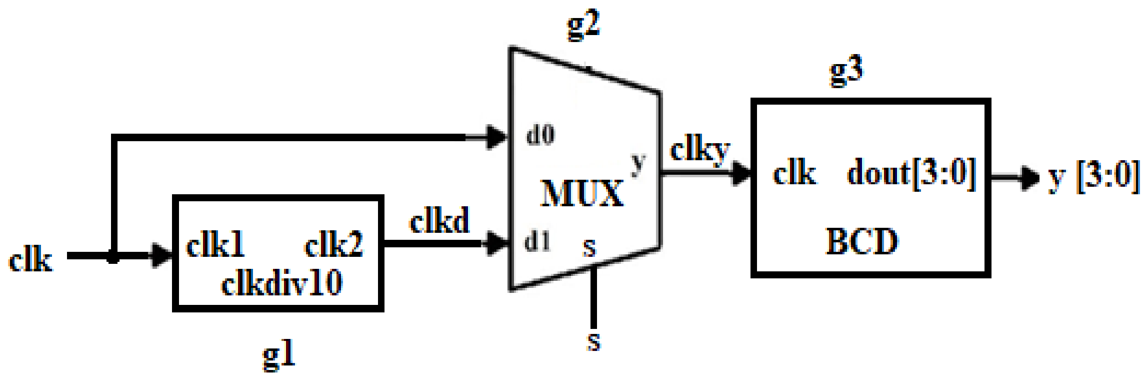


Figure 1. Two-Speed BCD counter circuit

Lab Procedure

Step 1. Design of Clock Divider of clkdiv10 module

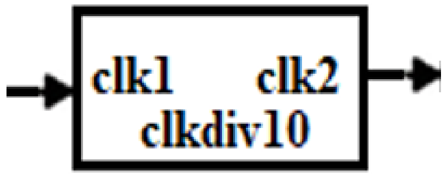


Figure 2. clkdiv10 module block diagram

The clkdiv10 design has one input clk1 and one output clk2. The frequency of clk2 is 10 times slower than the frequency of clk1.

Implement this design in Verilog and write testbench to run simulations. Check the simulation waveform to verify that the frequency of clk2 is 10 times slower than the frequency of clk1.

Step 2. Design of MUX module

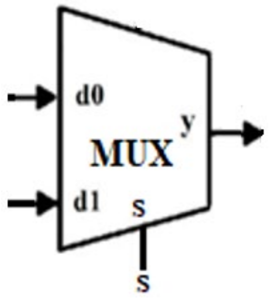


Figure 3. 2-to-1 multiplexer diagram

The 2-to-1 multiplexer consists of two inputs D0 and D1, one selection input S and one output Y. According to the logic value of the selection signal S, D0 or D1 will be passed to the output.

Implement this design in Verilog and write a testbench to run simulations.

Step 3. BCD Counter Design

The BCD counter circuit has one input clk, and four-bit output signal dout. The dout signal counts from 0 to 9 and then repeats.

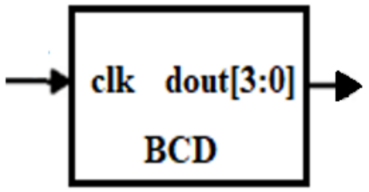
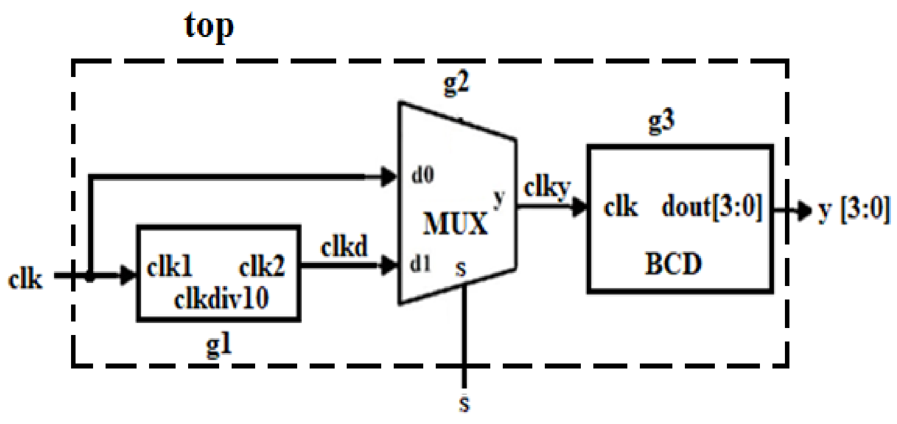


Figure 4. BCD counter diagram

Implement this design in Verilog and write testbench to run simulations. Check the simulation waveform to verify that the BCD counter can output a value from 0 to 9 on each rising edge of the clk signal, and then repeat the execution.

Step 4. Final Two-Speed BCD Counter Design

The figure below shows the final design required at step 4.



Design Name	Port Names	Port Direction	Port Size
top.v	clk	input	1 bit
	s	input	1 bit
	y	output	4 bits

Figure 5. The final two-speed BCD counter diagram and the interface information

The final two-speed BCD counter circuit requires one clkdiv10 instance, one MUX instances, and one BCD instance. You can name the final design top.v, which requires 1 bit input clk, 1 bit input s, and 4 bit output y. When s is logic 0, the BCD counter will run at the input clock frequency. When s is logic 1, the BCD counter will run 10 times slower. The output y generates a value from 0 to 9, and then repeats.

Implement this design in Verilog and write testbench to run simulations.

Demo Requirement

You need to demonstrate the simulation waveform of the final two-speed BCD counter design to your lab instructor.

CPE166 Lab 2 Part 3
By: Prof. Pang

Note: Before starting this experiment, all the necessary knowledge required to complete this work has been introduced in the CPE166 lecture session. The following examples are for you to refresh your learning.

Sample Verilog Codes:

```
module example (clkin, clkout);  // // clk division circuit:  divide by 8

input     clkin;

output    clkout;

reg       clkout;

reg [2:0]  cnt;

initial   cnt = 0;

always@(posedge clkin)

begin

  if (cnt == 7)

  begin

    cnt <= 0;

    clkout <= 1;

  end

  else if (cnt < 3)

  begin

    cnt <= cnt + 1;

    clkout <= 1;

  end

  else

  begin

    cnt <= cnt + 1;

    clkout <= 0;

  end

end

endmodule
```

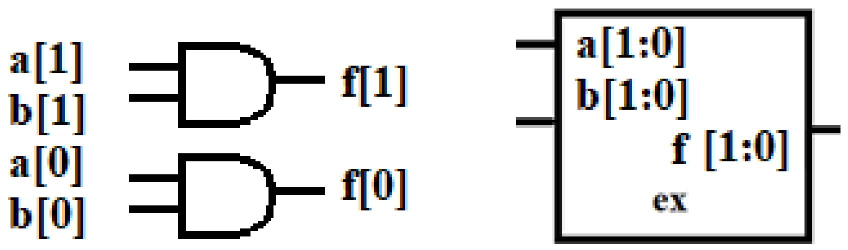
CPE166 Lab 2 Part 3
By: Prof. Pang

```
`timescale 1ns/1ps

module  example_tb;

reg    clkin;

wire   clkout;

integer  k;

example    uut ( clkin, clkout );

initial  clkin = 0;

always  #2 clkin = ~ clkin;

initial   begin

   k=0;

   while( k != 30 )

   begin

     @(posedge clkin);

      $display($time, " ns, k=%d, clkout=%b", k, clkout);

      k = k + 1;

   end

   #5 $stop;

end

endmodule
```

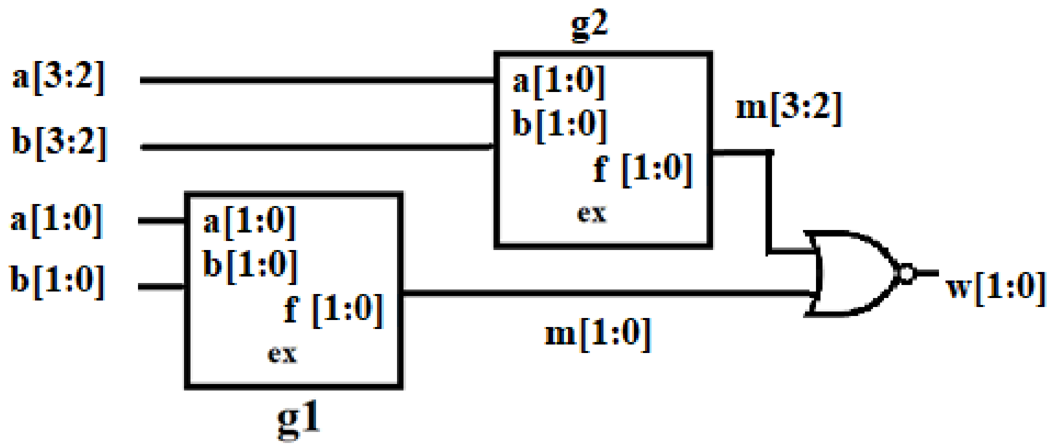
CPE166 Lab 2 Part 3
By: Prof. Pang

Sample Circuits:

(1).



(2).



Sample Implementation in Verilog:

```
module ex (a, b, f);

input    [1:0]  a, b;

output  [1:0]  f;

assign   f = a & b;

endmodule
```

CPE166 Lab 2 Part 3
By: Prof. Pang

```
`timescale 1ns/1ps

module  ex_tb;

reg   [1:0]    a, b;

wire  [1:0]   y;

ex    uut ( a, b, y);

integer   k;

initial   begin

   $monitor($time, " ns, a=%b, b=%b,  y=%b", a, b,  y);

    a= 2'b00;  b= 2'b00;

  #10             b= 2'b01;

  #10             b= 2'b10;

  #10             b= 2'b11;

  #10    a= 2'b01;  b= 2'b00;

  #10             b= 2'b01;

  #10             b= 2'b10;

  #10             b= 2'b11;

  #10    a= 2'b10;  b= 2'b00;

  #10             b= 2'b01;

  #10             b= 2'b10;

  #10             b= 2'b11;

  #10    a= 2'b11;  b= 2'b00;

  #10             b= 2'b01;

  #10             b= 2'b10;

  #10             b= 2'b11;

  #5  $stop;

end

endmodule
```

CPE166 Lab 2 Part 3
By: Prof. Pang

```
module ex2 (a, b, w);

input    [3:0]  a, b;

output  [1:0]  w;

wire      [3:0]  m;

ex     g1 ( .a ( a[1:0] ), .b( b[1:0] ),  .f (  m[1:0] ) );

ex     g2 ( .a ( a[3:2] ), .b( b[3:2] ),  .f (  m[3:2] ) );

assign    w =   ~ ( m[3:2] |  m[1:0] );

endmodule



`timescale 1ns/1ps

module  ex2_tb;

reg   [3:0]    a, b;

wire  [1:0]   y;

ex    uut ( a, b, y);

integer   k;

initial   begin

    $monitor($time, " ns, a=%b, b=%b,  y=%b", a, b,  y);

     for (k=0; k<256; k=k+1)

     begin

         {a, b} = k;

          #5;

      end

      #5  $stop;

end

endmodule
```

CPE166 Lab 2 Part 4
By: Prof. Pang

Lab 2

Part 4: Automatic Beverage Vending Machine

The token price of a beverage is 5 cents, and our specially designed beverage vending machine only accepts tokens of 1 cent, 2 cents and 5 cents. The task of the laboratory is to design a finite state machine for the automatic beverage vending machine. The following table shows the interface signals required for this design.

Table 1. Automatic Beverage Vending Machine Interface Signals

Port Names	Port Direction	Port Size
clk	input	1 bit
reset	input	1 bit
one	input	1 bit
two	input	1 bit
five	input	1 bit
d	output	1 bit
r	output	3 bits

- When the input "reset" signal is logic high, it resets the machine to a starting state.
- When the input "one" signal is logic high, it indicates that the 1-cent token has been inserted into the vending machine.
- When the input "two" signal is logic high, it indicates that the 2-cent token has been inserted into the vending machine.
- When the input "five" signal is logic high, it indicates that the 5-cent token has been inserted into the vending machine.
- At any time, only one token can be inserted into the vending machine.
- When the output "d" signal is logic high, the vending machine returns a drink, and the output "r" signal displays the total coin token returned by the vending machine.
- When the output "d" signal is logic low, the output "r" signal should be zero.

Implement this finite state machine design in Verilog and write testbench to run simulations.

Demo Requirement

You need to demonstrate the final simulation waveform of your design to your lab instructor.

CPE166 Lab 2 Part 4
By: Prof. Pang

Lab Procedure

Step 1. Draw State Diagram

When the reset signal is logic high, the state machine goes to the first idle state. The idle state means the current received coin token is 0. The output drink is 0 and the returned out token signal r is 0.

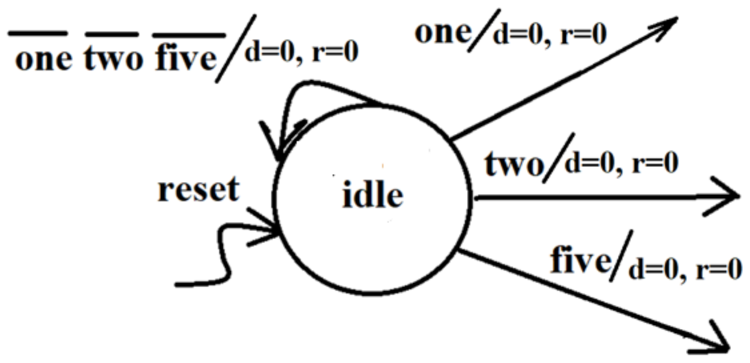


Figure 1. Idle state

Figure 1 shows the state diagram of the idle state. In the idle state, if the token is not received, the next state will still be the idle state. If a token of 1 cent is received, the "one" signal will be logic high and the finite state machine needs to enter another state. Similarly, if a token of 2 cents or a token of 5 cents is received, the finite state machine needs to enter a different state.

You can add more states to this design. In each state, you need to determine which state will become the next state based on the input signal values of "one", "two" and "five". In addition, you need to determine the output values of "d" and "r".

<u>Step 2. Verilog Design</u>

Complete the Verilog design of this project.

The Verilog design code snippet for the idle state is shown below:

```
always@(current_state or one or two or five)

begin

    case(current_state)

    idle:        if (one)   next_state =  … ;

                 else if (two) next_state =    …  ;

                 else if (five)  next_state =   … ;

                 else             next_state  =  idle;

     …        // describe more different states

      endcase

end


always@(current_state or one or two or five)

begin

    case(current_state)

    idle:      if (one)

                begin

                    d  = 0;

                    r   = 0;

                end

                 else if (two)

                  begin

                     d  = 0;

                      r  = 0;

                 end

                 else if (five)
```

```
                    begin
                          d = 0;
                          r  = 0;
                    end
                    else
                     begin
                          d  = 0;
                          r   = 0;
                      end
     …        // describe more different states
      endcase
end
```

Step 3. Testbench Simulation

Write testbench to run simulations of your project.

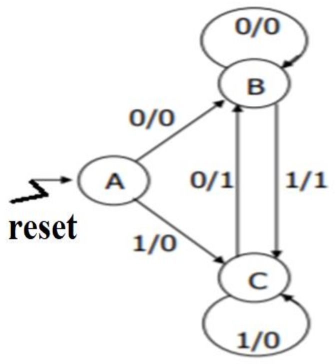
Demo Requirement

You need to demonstrate the simulation waveform to your lab instructor.

CPE166 Lab 2 Part 4
By: Prof. Pang

Note: Before starting this experiment, you have learned how to code a finite state machine in Verilog. This project is a design project. You need to follow the instructions above to complete the work. The following examples are for you to refresh your learning of finite state machine design.

Sample Example:



```
module fsm (reset, x, clk,  y, cs, ns);     //cs, and ns here are used for simulation

input      reset, x, clk;

output     y;

output  [1:0]  cs, ns;

reg          y;

reg [1:0]  cs, ns;

parameter A=2'b00,B=2'b01, C=2'b10;

always@(posedge clk or posedge reset)

begin

   if(reset) cs <= A;

   else  cs <= ns;

end

always@(cs or x)

begin

  case(cs)

     A:  y = 0;

     B:  if(x) y = 1;

          else y = 0;
```

```
    C:   if(x) y = 0;
         else y = 1;
    default:  y =0;
  endcase
end


always@(cs or x)
begin
     case(cs)
     A:  if(x) ns = C;
          else ns =B;
     B:  if(x) ns = C;
         else ns = B;
     C:  if(x) ns = C;
         else ns = B;
     default: ns = A;
     endcase
end
endmodule
```

CPE166 Lab 2 Part 4
By: Prof. Pang

```
`timescale 1ns/1ps

module fsm_tb;

reg    reset, x, clk;

wire   y;

wire [1:0] cs, ns;


fsm  uut ( reset, x, clk, y, cs, ns );


initial clk = 0;

always  #10 clk=~clk;


initial begin

   $monitor($time, " ns, x=%b, cs=%b, y=%b, ns=%b", x, cs, y, ns);

   reset = 1;  x = 0;

   #25 reset = 0; x = 1;

   #40  x = 0;

   #40  x = 1;

   #40  x = 0;

   #40  x = 1;

   #40  $stop;

end


endmodule
```