Laboratory 02

CPE 166 Advanced Logic Design

Section 03

Vigomar Kim Algador

Fall 2022

TABLE OF CONTENT

INTRODUCTION

This laboratory is an introduction on Verilog design and the use of Vivado. There will be different concepts that will be using in each project and understanding the use of it as well as utilizing and making it our own code. To further understand the concept of Verilog designing, this laboratory is divided into four parts. The first part of this laboratory shows how to design a 3-by-3 binary combination multiplier. This introduced the concept and use of half adder and full adder in general. For the second part of the laboratory, it shows the 8-bit carry select adder where got introduced using the additional concepts of multiplexer and 4-bit ripple carry adder which also consists of half adder and full adder. For the third part of the laboratory, the student is required to design a two-speed BCD counter which shows the multiplexer, a clock divider, and BCD counter. Finally, the last part of this laboratory introduced the automatic beverage vending machine which the whole concept is to design a finite state machine.

PART 1: 3 BY 3 BINARY COMBINATIONAL ARRAY MULTIPLIER

This part of the laboratory is to build a 3 by 3 combination array multiplier using the Verilog. The design for this multiplier consists of full adder and half adder. This laboratory shows the step-by-step process on designing the multiplier with the use of full adder and half adder. Below is the full diagram for this laboratory.

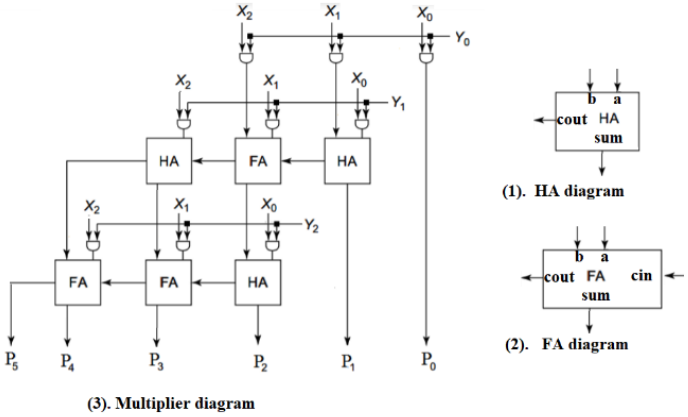

Figure 1. 3 by 3 binary combinational array multiplier diagram structure

Step 1: Half Adder Design

The first step that the student did is to design a half adder which adds two 1-bit binary inputs, named a and b, and generate two outputs, named sum signal and carry cout signal. Below is the truth table for the half adder as well as the circuit and logic symbol for it.

input		output	
a	b	cout	sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0
Logic equations: cout = a b sum a ⊕ b			

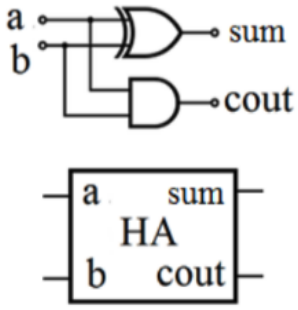

Figure 2: Half Adder truth table, circuit, and schematic diagram

After studying the circuit diagram and truth table, I was able to design using Verilog, write testbench, and run the simulations. Below is the full Verilog and testbench design as well as the simulated output.

```
1  `timescale 1ns / 1ps
2
3  // Vigomar Kim Algador
4  // CpE166 Section 03
5  // Laboratory 02 Part 01
6  // Half Adder
7
8  module HalfAdder(a,b,cout,sum);
9  input a, b;
10 output cout, sum;
11
12 assign cout = a&b;
13 assign sum = a^b;
14 endmodule
```

```
1  `timescale 1ns / 1ps
2
3  // Vigomar Kim Algador
4  // CpE166 Section 03
5  // Laboratory 02 Part 01
6  // Half Adder Testbench
7
8  module HalfAdder_tb;
9  reg a,b;
10 wire cout;
11 wire sum;
12 HalfAdder g1(.a(a),.b(b),.cout(cout),.sum(sum));
13
14 initial
15 begin
16 a = 0; b = 0;
17 #10 a = 1; b = 0;
18 #10 a = 0; b = 1;
19 #10 a = 1; b = 1;
20 #10 $stop;
21 end
22 endmodule
```

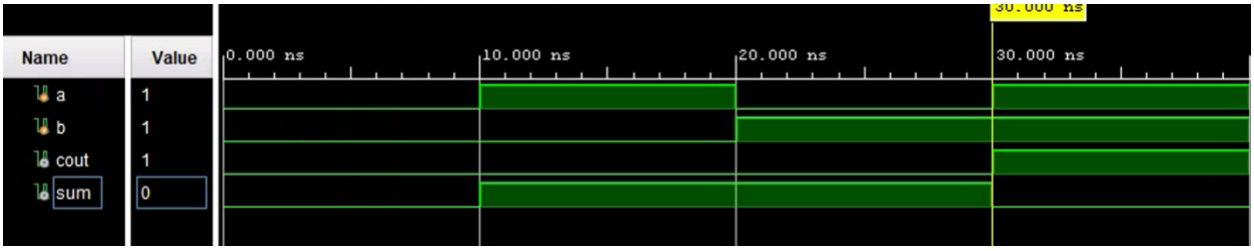

Figure 3: Half adder Verilog (top left), testbench (top right), and simulation (bottom)

Step 2: Full Adder Design

The next step will be to design a full adder that adds three 1-bit binary inputs a, b, and cin, and then generates two outputs named sum signal and carry out signal. The design for this full adder requires the use of 2 half adder module that did in step 1, and one OR gate. Below is the truth table for the full adder as well as the circuit and logic symbol.

input			output	
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: cout = a b sum a ⊕ b				

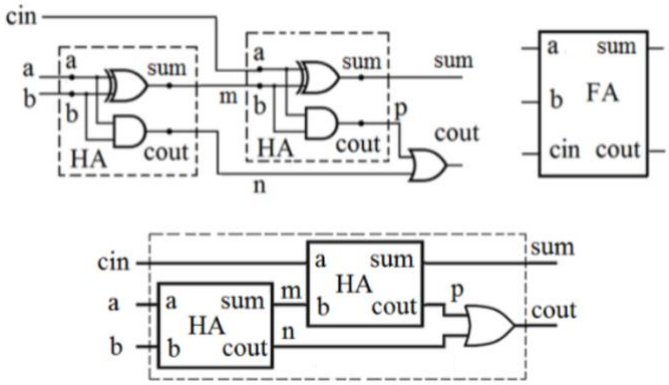

Figure 4: Full Adder truth table, circuit, and schematic diagram

After studying the circuit diagram and truth table, I was able to design using Verilog, write testbench, and run the simulations. Below is the full Verilog and testbench design as well as the simulated output.

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 01
// Full Adder

module FullAdder(a,b,cin,cout,sum);
input a,b,cin;
output cout,sum;
wire m,n,p;
HalfAdder g1(.a(a),.b(b),.cout(n),.sum(m));
HalfAdder g2(.a(cin),.b(m),.cout(p),.sum(sum));

assign cout = p|n;

endmodule
```

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 01
// Full Adder Testbench

module FullAdder_tb;
reg a,b,cin;
wire cout,sum;
FullAdder u1(.a(a),.b(b),.cin(cin),.cout(cout),.sum(sum));
integer k;

initial
begin
$monitor($time, "  a=%b, b=%b, cin=%b", a, b, cin);
for(k=0; k<8; k=k+1)
begin
    {a,b,cin}=k;
    #10;
end
#10 $stop;
end
endmodule
```

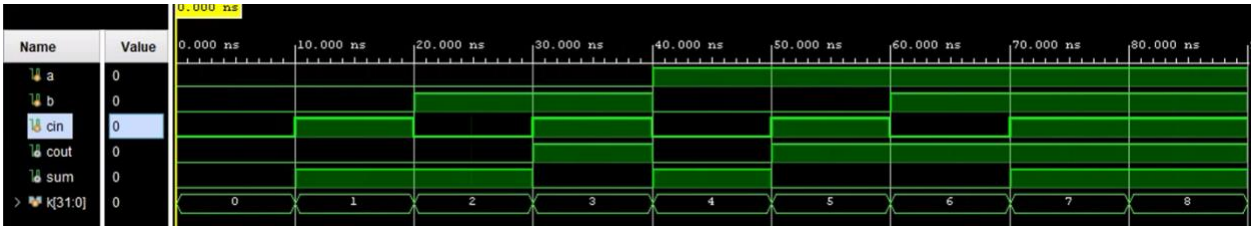

Figure 5: Full Adder Verilog (top left), testbench (top right), and simulation (bottom)

Step 3: Final Combination Multiplier Design

The last step will be designing the multiplier using the half adder and full adder from step 1 and step 2 and gates. The design for the 3 by 3 combination array multiplier will be consisting of nine AND gates, three Half Adder modules, and three Full Adder module that we analyzed from the schematic diagram shown in figure 1. After analyzing the design, I was able to design using Verilog, write testbench, and run the simulations. Below is the full Verilog and testbench design as well as the simulated output.

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 01
// Multiplier

module Multiplier(x,y,p);
input [2:0]x,y;
output [5:0]p;
wire[5:0]cin;
wire[1:0]sum;
wire[5:0]p;
wire[8:0]q;

assign q[0]=x[0]&y[0];
assign q[1]=x[1]&y[0];
assign q[2]=x[2]&y[0];
assign q[3]=x[0]&y[1];
assign q[4]=x[1]&y[1];
assign q[5]=x[2]&y[1];
assign q[6]=x[0]&y[2];
assign q[7]=x[1]&y[2];
assign q[8]=x[2]&y[2];

assign p[0]=q[0];
HalfAdder ha1(.a(q[3]),.b(q[1]), .cout(cin[0]), .sum(p[1]));
FullAdder fa1(.a(q[4]),.b(q[2]),.cin(cin[0]),.cout(cin[1]),.sum(sum[0]));
HalfAdder ha2(.a(q[5]),.b(cin[1]), .cout(cin[2]), .sum(sum[1]));
HalfAdder ha3(.a(q[6]),.b(sum[0]), .cout(cin[3]), .sum(p[2]));
FullAdder fa2(.a(sum[1]),.b(q[7]),.cin(cin[3]),.cout(cin[4]),.sum(p[3]));
FullAdder fa3(.a(cin[2]),.b(q[8]),.cin(cin[4]),.cout(p[5]),.sum(p[4]));
endmodule
```

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 01
// Multiplier Testbench

module Multiplier_tb;
reg [2:0]x,y;
wire [5:0]p;
integer i,j;

Multiplier ul(.x(x),.y(y),.p(p));

initial begin
x=3'b000;
y=3'b000;

for(i=0;i<8;i=i+1)
begin
    x=i;
    #10;
    for(j=0;j<8;j=j+1)
    begin
        y=j;
        #10;
    end
end
#10 $stop;
end
endmodule
```

Figure 6: Multiplier Verilog (top left), testbench (top right), and simulation (bottom)

Result Discussion

For this part of this laboratory, I was able to test and verify the output I got for the 3 by 3 combination array multiplier. Using the knowledge I have from the lecture class, I was able to easily understand the concept of the multiplier as well as the full adder and the half adder. I just a little got stucked when designing the final combination multiplier as I need to combine all the block diagram of full adder and half adder.

PART 2: 8-BIT CARRY SELECT ADDER

In this part of the laboratory, the purpose of this is to design an 8-bit carry select adder. This design consists of three 4-bit ripple carry adders (RCA4) and two multiplexers (MUX and MUXB).

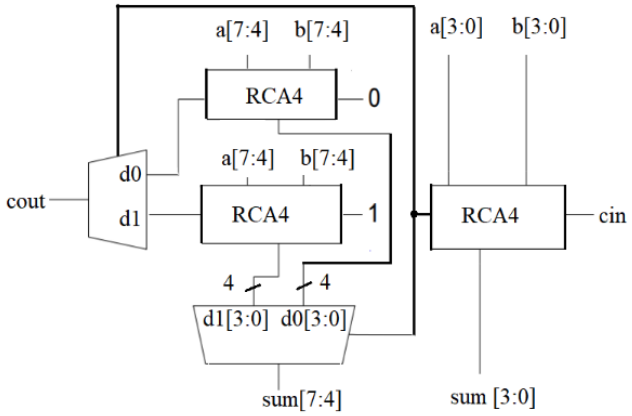

Figure 7: 8-bit carry-select adder circuit

Step 1: 4-Bit Ripple Carry Adder

4-bit ripple carry adder produces arithmetic sum of two binary numbers, named a and b. The 4-bit ripple carry adder consists of 4 full adders connected with the carry output from each of full adder and connected to carry input of the next full adder which gives a chain effect. The full diagram is shown below.

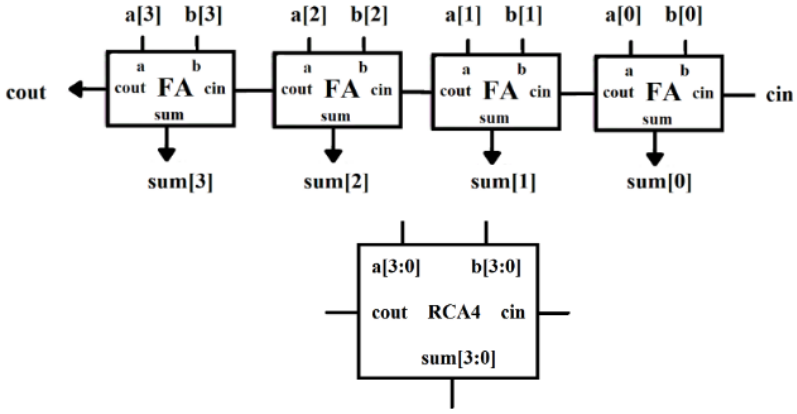

Figure 8: 4-bit ripple carry adder circuit and block diagram

After analyzing the circuit and diagram, I was able to design the RCA4 using the full adder from the previous part of this laboratory. Additionally, I used the same half adder for the full adder. After, I created the RCA4 using 4 full adders. Below is the full Verilog, testbench design, and the simulated output.

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 02
// 4-bit Ripple Carry Adder

module RCA4(a, b, cin, cout, sum);
input [3:0] a,b;
input cin;
output [3:0] sum;
output cout;
wire [2:0] m;

FullAdder h1(.cout(m[0]),.sum(sum[0]),.a(a[0]),.b(b[0]),.cin(cin));
FullAdder h2(.cout(m[1]),.sum(sum[1]),.a(a[1]),.b(b[1]),.cin(m[0]));
FullAdder h3(.cout(m[2]),.sum(sum[2]),.a(a[2]),.b(b[2]),.cin(m[1]));
FullAdder h4(.cout(cout),.sum(sum[3]),.a(a[3]),.b(b[3]),.cin(m[2]));

endmodule
```

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 02
// 4-bit Ripple Carry Adder

module RCA4_tb;
reg [3:0] a, b;
reg cin;
wire [3:0] sum;
wire cout;
integer k;

RCA4 u1 (a, b, cin, cout, sum);

initial
begin
    $monitor($time,"ns, k=%d, a=%d, b=%d, cin=%d, cout=%d, sum=%d", k, a, b, cin, cout, sum);
    for(k=0; k<512; k=k+1)
    begin
        #10 {a,b,cin}=k;
    end
    #10 $stop;
end
endmodule
```

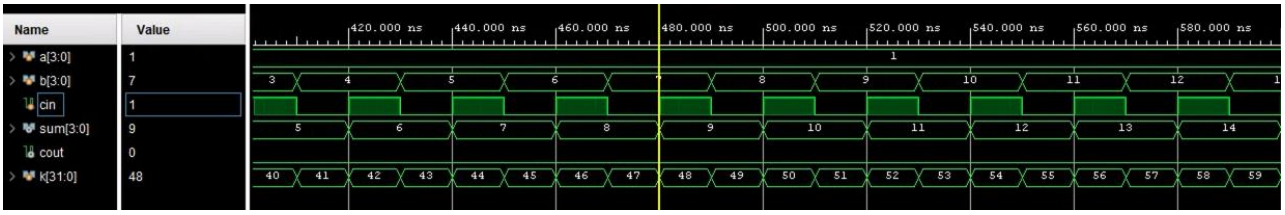

Figure 9. RCA4 Verilog (top left), testbench (top right), and simulation (bottom)

Step 2: Multiplexer (MUX)

After creating the RCA4, then I create a 2-to-1 multiplexer which consists of two inputs D0 and D1, one selection input S and one output Y. Below is the truth table and diagram for this design.

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

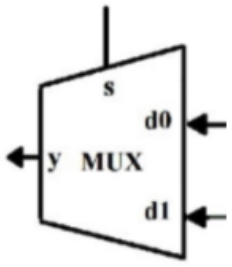

Figure 10: 2-to-1 multiplexer (MUX) truth table and diagram

After analyzing the truth table and the diagram, I was able to write the Verilog as well as the testbench. Below is the full design for the Verilog, testbench, and simulation for the multiplexer.

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 02
// Multiplexer (MUX)

module MUX(s, d0, d1, y);
input s, d0, d1;
output y;
reg y;
always@(d1 or d0 or s)
begin
if (s == 0)
    y = d0;
else
    y = d1;
end
endmodule
```

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 02
// Multiplexer (MUX)

module MUX_tb;
reg d1, d0, s;
wire y;
integer k;

MUX u1(d1, d0, s, y);

initial
begin
$monitor($time, "ns, d1=%b, d0=%b, s = %b, y=%b", d1, d0, s, y);
for(k=0; k<8; k=k+1)
    #10 {d0,d1,s}=k;
#10 $stop;
end
endmodule
```

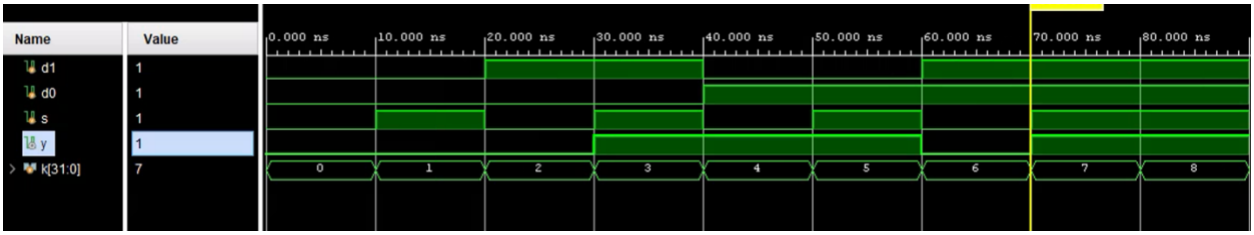

Figure 11: MUX Verilog (top left), testbench (top right), and simulation (bottom)

Step 3: Multiplexer (MUXB)

After with the first multiplexer, I need to create another multiplexer similar to step 2. The difference between the first multiplexer to this one is we need to declare the input d1, d0, and the output y as 4-bit data. Below is the simplified truth table for this design.

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

Figure 12: 4-bit Multiplexer truth table

Similar to the last step, I was able to copy and modify the Verilog and testbench for this multiplexer. Below is the full design Verilog, testbench, and simulation for the 4-bit multiplexer (MUXB).

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 02
// Multiplexer (MUXB)

module MUXB(d1, d0, s, y);
input [3:0] d1, d0;
input s;
output [3:0] y;
reg [3:0] y;

always@(d1 or d0 or s)
begin
if(s) y = d1;
else y = d0;
end
endmodule
```

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 02
// Multiplexer (MUXB)

module MUXB_tb;
reg [3:0] d1,d0;
reg s;
wire [3:0] y;
integer k;

MUXB k1(d1, d0, s, y);

initial
begin
$monitor($time, "ns, d1=%b, d0=%b, s = %b, y=%b", d1, d0, s, y);
for(k=0; k<512; k=k+1)
    #10 {d0,d1,s}=k;
#10 $stop;
end
endmodule
```

Figure 13: MUXB Verilog (top left), testbench (top right), and simulation (bottom)

Step 4: Final 8-bit Carry Select Adder

After I design all the parts for this part of the laboratory, the last step is to combine all of the parts and design it into 8-bit carry select adder. Below is the Verilog design, testbench, and the simulated output for the CSA8.

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 02
// Final 8-bit Carry Select Adder

module CSA8(a , b, cin, cout, sa, sb, sum);
input [7:0] a, b;
input cin;
output [7:0] sa, sb, sum;
output cout;
wire c1, c2, c3, c4, m;

RCA4 r1(.a(a[3:0]), .b(b[3:0]), .cin(0), .cout(c1), .sum(sa[3:0]));
RCA4 r2(.a(a[3:0]), .b(b[3:0]), .cin(1), .cout(c2), .sum(sb[3:0]));

MUXB mb1(.d1(sb[3:0]), .d0(sa[3:0]), .s(cin), .y(sum[3:0]));
MUX m1(.d1(c2), .d0(c1), .s(cin), .y(m));

RCA4 r3(.a(a[7:4]), .b(b[7:4]), .cin(0), .cout(c3), .sum(sa[7:4]));
RCA4 r4(.a(a[7:4]), .b(b[7:4]), .cin(1), .cout(c4), .sum(sb[7:4]));

MUXB mb2(.d1(sb[7:4]), .d0(sa[7:4]), .s(m), .y(sum[7:4]));
MUX m2(.d1(c4), .d0(c3), .s(m), .y(cout));

endmodule
```

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 02
// Final 8-bit Carry Select Adder

module CSA8_tb;
reg cin;
reg [7:0] a,b;
wire cout;
wire [7:0] sum;
integer k;

CSA8 u1(a, b, cin, cout, sa, sb, sum);

initial begin
for(k=0;k<131072;k=k+1)
    #5 {cin,a,b}=k;
#10 $stop;
end
endmodule
```

Figure 14: CSA8 Verilog (top left), testbench (top right), and simulation (bottom)

Result Discussion

For this part of the laboratory, I was able to design each module and combined it into 8-bit carry adder. I was able to test the result and check for the inputs a and b having the result for the sum. The whole point of this laboratory is to learn how to design and use multiplexer and the 4-bit ripple carry adder.

PART 3: TWO-SPEED BCD COUNTER

In this part of this laboratory, the student must create a binary coded decimal (BCD) counter that counts from 0 to 9 and then repeats. Additionally, this experiment is to design the counter in which can choose two different speeds. This laboratory will be using 3 different block diagram parts, a clock divider, 2-to-1 multiplexer, and the BCD counter. Below is the full circuit of a two-speed BCD counter.

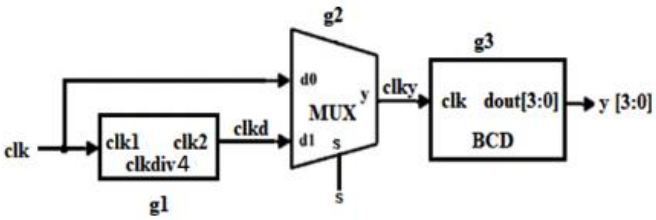

Figure 15: Two-Speed BCD counter circuit

Step 1: Clock Divider

For the first step, I must design a clock divider in which the frequency output is 4 times slower than the input. This clock divider will be named clkdiv4 which has one input clk1 and one output clk2. Below is the block diagram of clkdiv4.

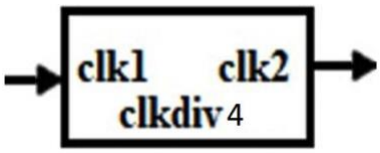

Figure 16: clkdiv4 module block diagram

After learning the concept of clock divider, I was able to design the Verilog as well as the testbench. Below is the Verilog design, testbench, and simulated output for the clkdiv4.

```
// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 03
// clock divider clkdiv4

module clkdiv4(clk1,clk2);
input clk1;
output clk2;
reg clk2;
reg[2:0] cnt;

initial cnt = 0;
initial clk2 = 0;

always@(posedge clk1)
begin
    if(cnt == 3)
    begin
        cnt <= 0;
        clk2 <= 1;
    end
    else if(cnt < 1)
    begin
        cnt <= cnt + 1;
        clk2 <= 1;
    end
    else
    begin
        cnt <= cnt + 1;
        clk2 <= 0;
    end
end
endmodule
```

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 03
// clock divider clkdiv4

module clkdiv4_tb;
reg clk1;
wire clk2;
integer k;

clkdiv4 uut(clk1, clk2);
initial clk1 = 0;
always #2 clk1 =~ clk1;
initial begin
 k = 0;
 while(k!=50)
 begin
    @(posedge clk1);
    $display($time, "ns, k=%d, clk2=%b",k,clk2);
    k=k+1;
    end
    #5 $stop;
end
endmodule
```

Figure 17: clkdiv4 Verilog (top left), testbench (top right), and simulation (bottom)

Step 2: Multiplexer (MUX)

For the next step, I need to design a 2-to-1 multiplexer which is the same as multiplexer from the part 2 of this laboratory. It consists of two inputs D0 and D1, one selection input S and one output Y. Below is the simplified truth table and a diagram.

s	y
0	d0
1	d1

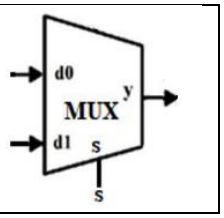

Figure 18: 2-to-1 Multiplexer truth table and diagram

I was able to use the same multiplexer and modified some code from the part 2 of this laboratory. Below is the Verilog design, testbench, and simulated result.

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 03
// Multiplexer (mux)

module mux(s, d0, d1, y);
input s, d0, d1;
output y;
reg y;
always@(s or d0 or d1)
begin
if (s) y = d1;
else y = d0;
end
endmodule
```

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 03
// Multiplexer (mux)

module mux_tb;
reg s, d0, d1;
wire y;
integer k;

mux uut(.d0(d0),.d1(d1),.s(s),.y(y));

initial
begin
$monitor($time, "ns, d1=%b, d0=%b, s = %b, y=%b", d1, d0, s, y);
for(k=0; k<8; k=k+1)
begin
    {d0,d1,s}=k;
    #10;
   end
#10 $stop;
end
endmodule
```

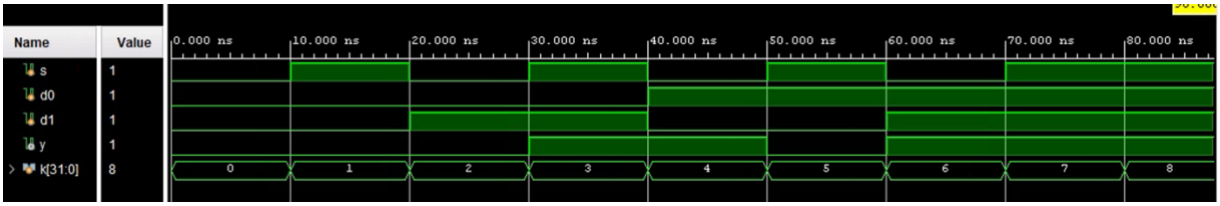

Figure 19: mux Verilog (top left), testbench (top right), and simulation (bottom)

Step 3: BCD Counter

For the next step, I need to design a BCD counter which has one input clk and 4-bit output signal dout. The purpose of dout signal is to count from 0 to 9 and repeats. Below is the diagram for the BCD Counter

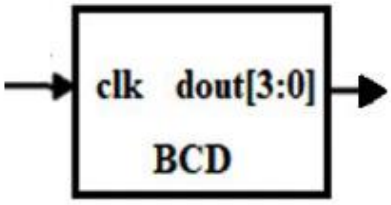

Figure 20: BCD counter diagram

After analyzing and learning the concept for BCD counter, I was able to design the Verilog and testbench which shown below.

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 03
// BCD counter

module bcd(clk,dout);
input clk;
output [3:0] dout;
reg [3:0] dout,cnt;

initial dout=0;
initial cnt=0;

always @(posedge clk)
begin
if(dout == 9)
begin
    dout <= 0;
end
else
begin
    cnt <= cnt+1;
    dout = dout+1;
end
end
endmodule
```

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 03
// BCD Counter

module bcd_tb();
reg clk;
wire [3:0] dout;

bcd uut(clk, dout);

initial
begin
clk=0;
forever #5 clk=~clk;
end
endmodule
```

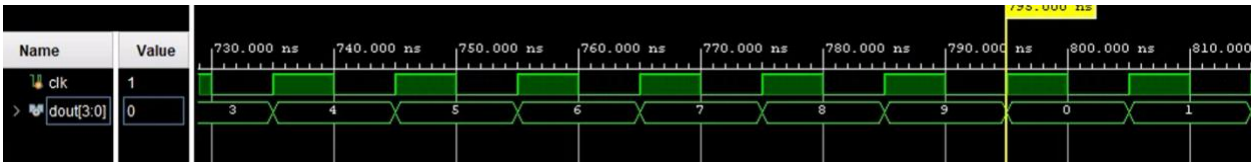

Figure 21: bcd module Verilog (top left), testbench (top right), and simulation (bottom)

Step 4: Final Two-Speed BCD Counter

For the last step of this part of the laboratory, I need to combine all the things I did from step 1 to step 3 to create the two-speed BCD counter. This design requires 1-bit input clk, 1-bit input s, and 4-bit output y. Below is the full diagram and interface information for this counter.

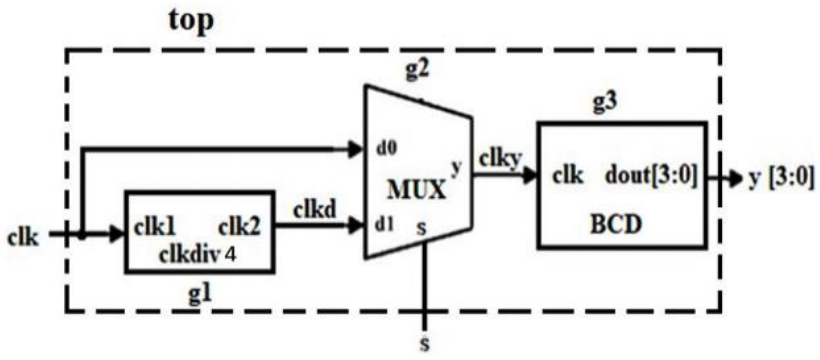

Figure 22: Final two-speed BCD counter diagram

Table 1. Final two-speed BCD counter interface information

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

After analyzing, I designed the whole Verilog as well as testbench for this part of the laboratory. Below is the Verilog, testbench, and as well as the simulated output.

```
1  `timescale 1ns / 1ps
2
3  // Vigomar Kim Algador
4  // CpE166 Section 03
5  // Laboratory 02 Part 03
6  // Final two-speed BCD counter
7
8  module top(clk,s,y);
9  input clk,s;
10 output [3:0]y;
11 wire clkd,clky;
12 clkdiv4 g1(.clk1(clk), .clk2(clkd));
13 mux g2(.d0(clk), .d1(clkd),.s(s),.y(clky));
14 bcd g3(.clk(clky), .dout(y));
15
16 endmodule
17
```

```
1  `timescale 1ns / 1ps
2
3  // Vigomar Kim Algador
4  // CpE166 Section 03
5  // Laboratory 02 Part 03
6  // Final two-speed BCD counter
7
8  module top_tb;
9  reg clk,s;
10 wire [3:0]y;
11
12 top uut(clk, s, y);
13
14 initial clk = 0;
15 always #2 clk = ~clk;
16 initial begin
17 s = 0;
18 #100 s=1;
19 #100 $stop;
20 end
21 endmodule
```

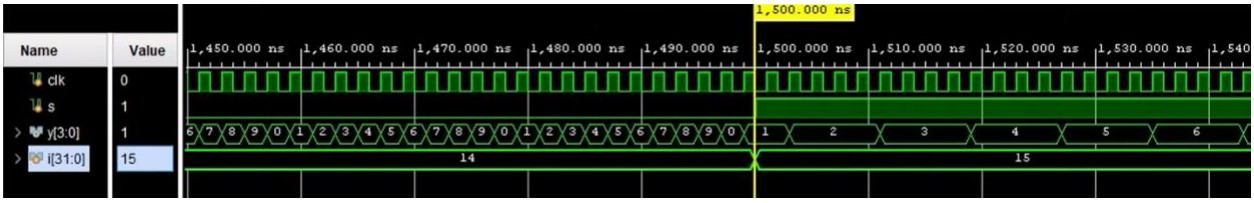

Figure 21: top Verilog (top left), testbench (top right), and simulation (bottom)

Result Discussion

In this part of the laboratory, I was able to observe the simulated output in each step which seems associated with time and division just like the clkdiv4 where makes it 4 times slower for the output and the two-speed BCD run 4 times slower when the s is logic 1 from the use of 2-to-1 multiplexer. The whole concept wasn't really complicated; however I ran into some issues with the testbench and understanding it.

PART 4: AUTOMATIC BEVERAGE VENDING MACHINE

For this part of the laboratory, I was tasked to make an automatic beverage vending machine where the token prices for a beverage will be 5 cents, and this vending machine only accepts tokens of 1 cent, 2 cents, and 5 cents. Below are the interface signals required for this part of the laboratory.

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

Step 1: Draw State Diagram

From the instruction on designing this part of the laboratory, I was able to draw the state diagram for this laboratory shown below.

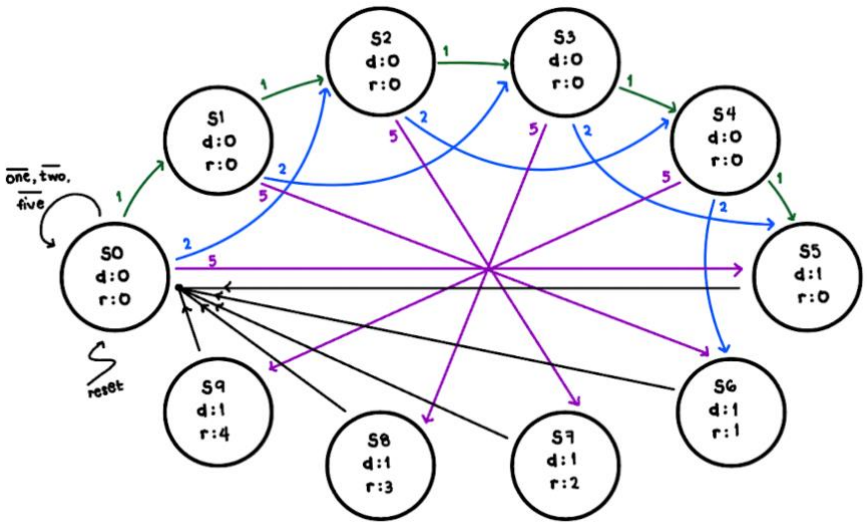

Figure 22: Finite state machine for vending machine

Step 2: Verilog Design

After analyzing the instruction and state machine I drew, I was able to design the Verilog for this vending machine shown below. Additionally, S0 from the finite state machine drawn in step 1 is the same as idleState I wrote in the Verilog design.

```
1  `timescale 1ns / 1ps
2
3  // Vigomar Kim Algador
4  // CpE166 Section 03
5  // Laboratory 02 Part 04
6  // Automatic Beverage Vendhing Manchine
7
8  module ABVM (clk,reset,one,two,five,d,r);
9  input clk,reset,one,two,five;
10 output d;
11 output [2:0] r;
12 reg [3:0]cs, ns;
13 reg[2:0]r;
14 reg d;
15
16 parameter idleState=0, s1=1, s2=2, s3=3, s4=4, s5=5, s6=6, s7=7, s8=8, s9=9;
17
18 always@(posedge clk or posedge reset)
19 begin
20     if(reset) cs <= idleState;
21     else      cs <= ns;
22 end
23
24 always@(cs or one or two or five or posedge reset)
25 begin
26     case(cs)
27     idleState: begin
28         if(one) ns = s1;
29         else if(two) ns = s2;
30         else if(five) ns =s5;
31         else ns = idleState; end
32     s1: begin
33         if(one)ns = s2;
34         else if(two) ns = s3;
35         else if(five) ns =s6;
36         else ns =s1; end
37     s2: begin
38         if(one) ns =s3;
39         else if(two) ns= s4;
40         else if(five) ns =s7;
41         else ns =s2; end
42     s3: begin
43         if(one) ns= s4;
44         else if(two) ns=s5;
45         else if(two) ns=s8;
46         else ns= s3; end
47     s4: begin
48         if(one) ns = s5;
49         else if(two) ns = s6;
50         else if(five) ns = s9;
51         else ns = s4; end
52     s5: ns = idleState;
53     s6: ns = idleState;
54     s7: ns = idleState;
55     s8: ns = idleState;
56     s9: ns = idleState;
57     default: ns = idleState;
58  endcase
59  end
60
61 always@(cs or one or two or five)
62 begin
63 case(cs)
64     idleState: begin
65         if(one) begin d = 0; r = 0; end
66         else if (two) begin d = 0; r = 0; end
67         else if (five) begin d = 1; r = 0; end
68         end
69     s1: begin d=0; r=0; end
70     s2: begin d=0; r=0; end
71     s3: begin d=0; r=0; end
72     s4: begin d=0; r=0; end
73     s5: begin d=1; r=0; end
74     s6: begin d=1; r=1; end
75     s7: begin d=1; r=2; end
76     s8: begin d=1; r=3; end
77     s9: begin d=1; r=4; end
78     default: begin d=1; r=4; end
79 endcase
80 end
81 endmodule
```

Figure 23. ABVM Verilog design

Step 3: Testbench Simulation

After designing the Verilog for this machine, I was able to write the testbench and have a simulated output shown below.

```
`timescale 1ns / 1ps

// Vigomar Kim Algador
// CpE166 Section 03
// Laboratory 02 Part 04
// Automatic Beverage Vendhing Manchine

module ABVM_tb();
reg clk, reset, one, two, five;
wire d;
wire [2:0] r;
ABVM u1(.clk(clk), .reset(reset), .one(one), .two(two), .five(five), .d(d), .r(r));
initial clk =0;
always #1 clk = ~clk;

initial begin
one = 0; two = 0; five = 0; reset = 0;
end

initial begin
#2 reset = 1; #2 reset = 0;

#2 one = 1; #2 one = 0;
#2 one = 1; #2 one = 0;
#2 one = 1; #2 one = 0;
#2 one = 1; #2 one = 0;
#2 one = 1; #2 one = 0;     // d=1 r=0

#2 two = 1; #2 two = 0;
#2 two = 1; #2 two = 0;
#2 two = 1; #2 two = 0;     // d=1 r=1

#2 five = 1; #2 five = 0;   // d=1 r=0

#2 one = 1; #2 one = 0;
#2 five = 1; #2 five = 0;   // d=1 r=1

#2 two = 1; #2 two = 0;
#2 five = 1; #2 five = 0;   // d=1 r=2

#2 one = 1; #2 one = 0;
#2 two = 1; #2 two = 0;
#2 five = 1; #2 five = 0;   // d=1 r=3

#2 two = 1; #2 two = 0;
#2 two = 1; #2 two = 0;
#2 five = 1; #2 five = 0;   // d=1 r=4
#15 $stop;
end
endmodule
```

Figure 24: ABVM testbench

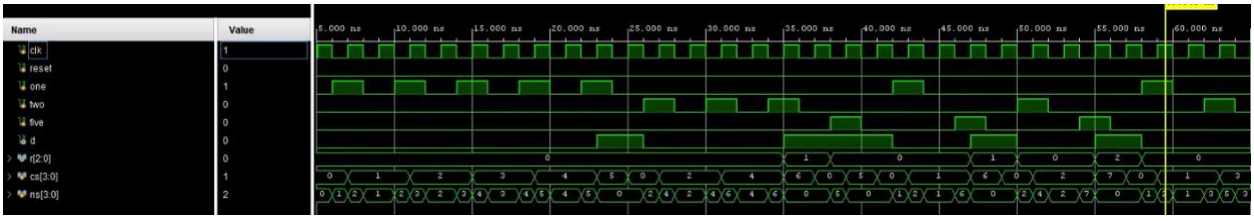

Figure 25: ABVM simulated output

Result Discussion

In this part of the laboratory, I was able to understand the whole concept as it depicts the real-life vending machine. The instructions were pretty clear as I was able to draw the finite state machine easily. On the other hand, I ran some issues with designing the Verilog as well as the testbench which are common syntax error mistakes. Otherwise, I was able to check the output for this part of the laboratory.

CONCLUSION

The whole laboratory discusses the basic concept of Verilog design and testbench, and different topics discussed in the lecture. Additionally, this laboratory introduced the basic use of Vivado and the settings on designing our own projects. I was able to use my knowledge about different topics such as half adder, full adder, multiplexer, clock divider, and state machine, and then utilize them into projects. The difficulties I found in this laboratory is creating a testbench in each part of the laboratory. For the first part of the laboratory, I was able to create a 3 by 3 binary combination multiplier by using the full adder and half adder. These two modules help me to familiarize the basic circuit and understanding logic gates. I was able to understand the basic of adding two binary number with the concept of carry in and carry out for half adder and full adder. For the second part of the laboratory, I learned the new concept of multiplexer and the function of ripple carry adder. For the third part of the laboratory, I was able to understand the concept of clock divider and the BCD counter. Understanding the concept behind it is simple but designing the Verilog as well as the testbench seems a little complicated as expected. For the last part of this laboratory, I was able to interpret the instruction for designing an automatic beverage vending machine and design its finite state machine. Just like the third part, its clearly easy to visualize as I drew the finite state machine for this part of the laboratory than designing the Verilog and testbench. On the other hand, drawing the finite state machine helps me clearly understand what I need to write for the project.