

《数字系统设计》实验报告

ALU 及数据通路控制时序电路设计及仿真 Design and Simulation of ALU with sequential machine controlled datapath

小组成员（按姓氏首字母排序）：

姓名	UCD 学号	北工大学号
CHEN DINGRUI	17206208	17371131
GONG CHEN	17205914	17371117
LIU ZIYANG	16206553	16371322
SHI BO	17205870	17374120
XU ZHIKUN	17205897	17371130
ZHU YANXING	17205900	17371118

评语及分项得分

分项	实验预习及 实验方案	出勤	实验结果 及分析	现场问答	实验报告
评语					
得分					

最终评定得分：_____

2019 年 12 月 10 日

Lab 2 Report - Design and Simulation of ALU with sequential machine controlled datapath

Lab 2 Report - Design and Simulation of ALU with sequential machine controlled datapath

Lab Targets

Circuit Diagram

Code and Testbench

Top Module

ALU Design

Register Design

Toggle Bottom Design and Testbench

Toggle Bottom Design

Toggle Bottom Testbench

Top Module Testbench

Resource Allocation

Summary

Source Code

Top Module - Lab2.v

ALU - ALU.v

Register - Register.v

Toggle Bottom - Toggle_Button.v

Toggle Bottom Test Bench

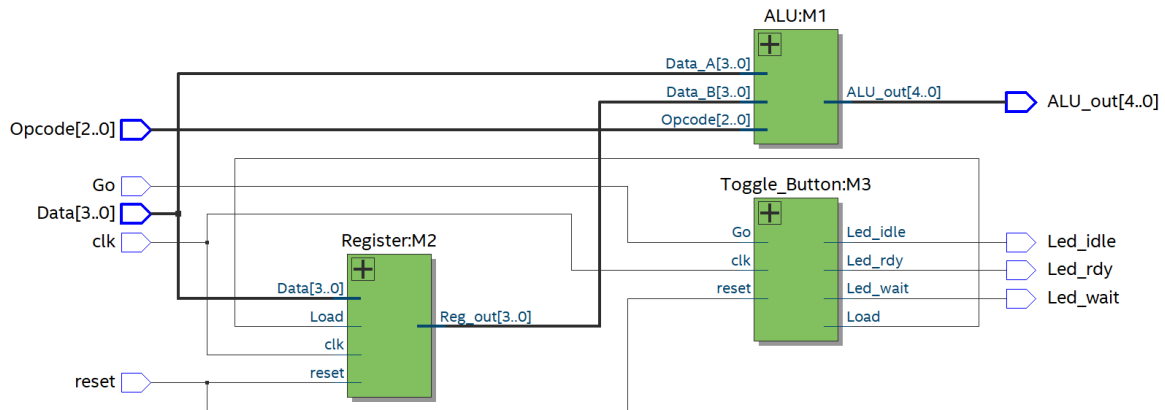
Top Module Test Bench

Lab Targets

Design an 8-bit Arithmetic Logical Unit (ALU), based on the method of top-down module system design.

Use the simple ALU from the previous experiment and write other circuits to complete the circuit design and test code writing. Under the EDA platform - ModelSim, complete the design input, compilation, and functional simulation verification.

Circuit Diagram



Code and Testbench

Top Module

```

1  module Lab2 (
2      output [4:0] ALU_out,
3      output Led_idle, Led_wait, Led_rdy,
4      input [3:0] Data,
5      input [2:0] Opcode,
6      input Go,
7      input clk, reset
8  );
9      wire [3:0] Reg_out;
10
11     ALU          M1 (ALU_out, Data, Reg_out, Opcode);
12     Register     M2 (Reg_out, Data, Load, clk, reset);
13     Toggle_Button M3 (Load, Led_idle, Led_wait, Led_rdy, Go, clk, reset);
14
15 endmodule

```

This model named Lab1, has five input port and four output ports. Which are

1. ALU_out: A 5-bit output, which is result of number after ALU process.
2. Led_idel, Led_wait, Led_rdy: These three are LED to indicate the internal status of the system.
3. Data: A 4-bit input, the data be sent to ALU.
4. Opcode: A 3-bit input, to control the ALU function.
5. Go: It is a button to control the system.
6. clk: A clock signal.
7. reset: Hardware reset.

ALU Design

This part using the same design in Lab 1.

Register Design

```
1 module Register (output reg [3:0] Reg_out, input [3:0] Data, input Load, clk, reset);
2     always @(posedge clk) begin
3
4         if(reset) Reg_out = 4'b0;
5         else begin
6             if(Load) Reg_out <= Data ;
7         end
8     end
9 endmodule
```

This model named Register. Having 5 ports.

1. Reg_out: One 4-bit register type output, sending the data to ALU model.
2. Data: One 4-bit input, sending the data in the register.
3. Load: When **load** is HIGH, the data can write into the register. When load is LOW, the data can read out of the register.
4. clk: Clock signal.
5. reset: When **reset** is HIGH, the content of the register should be empty.

Line 2: Statement `always` here to tell that the following codes always at the positive edges of the clock signal, `clk`.

Line 4: If **reset** signal is HIGH, then clears the register. In our implementation, sending zero to output directly.

Line 5 to Line 7: If **reset** signal is LOW and **Load** signal is HIGH, sending input data to Reg_out. According to behavior of Register model are controlled by clock, the data will keeping in one clock cycle.

Toggle Bottom Design and Testbench

Toggle Bottom Design

```
1 module Toggle_Button (output reg Load, Led_idle, Led_wait, Led_rdy, input Go, clk, reset);
2     reg [1:0] state = 0;
3
4     always @(posedge clk) begin
5
6         if(reset) state = 0;
7         else begin
8             if(Go && state == 0) state = 2'b1;
9             else if(state == 1) state = 2'b10;
10            else if(!Go && state == 2'b10) state = 2'b11;
11            else if (Go && state == 2'b11) begin
12                state = 2'b1;
13                Led_rdy = 0;
14            end
15        end
16
17        case(state)
18            2'b0:          begin
19                            Led_idle = 1;
20                            Led_wait = 0;
21                            Led_rdy = 0;
```

```

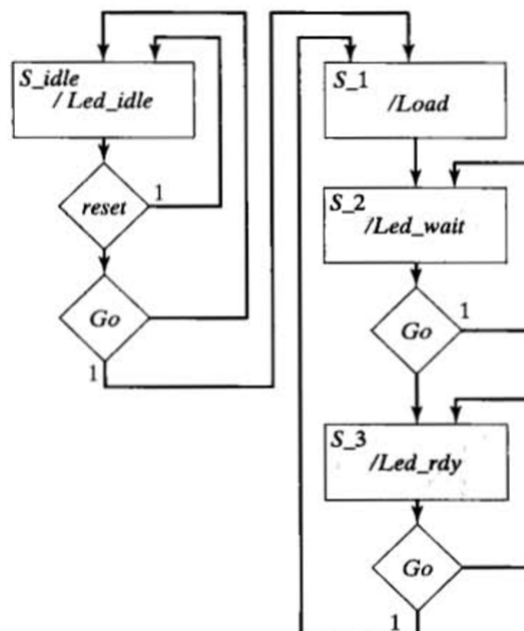
22             Load = 0;
23         end
24
25     2'b1:      begin
26                 Led_idle = 0;
27                 Load = 1;
28             end
29
30     2'b10:     begin
31                 Load = 0;
32                 Led_wait = 1;
33             end
34
35     2'b11:     begin
36                 Led_wait = 0;
37                 Led_rdy = 1;
38             end
39     endcase
40 end
41 endmodule

```

This model named Register. Having 7 ports,

1. Load: One bit register type output, using for register model.
2. Led_idle, Led_wait, Led_rdy: These three are LED to indicate the internal status of the system.
3. Go: It is a button to control the system.
4. clk: Clock signal.
5. reset: When **reset** is HIGH, this model go to idle.

The following is flow chart of this model



Line 6: If **reset** signal is HIGH, the state of this model goes to initial state - S_idle. In our code, it is state 0.

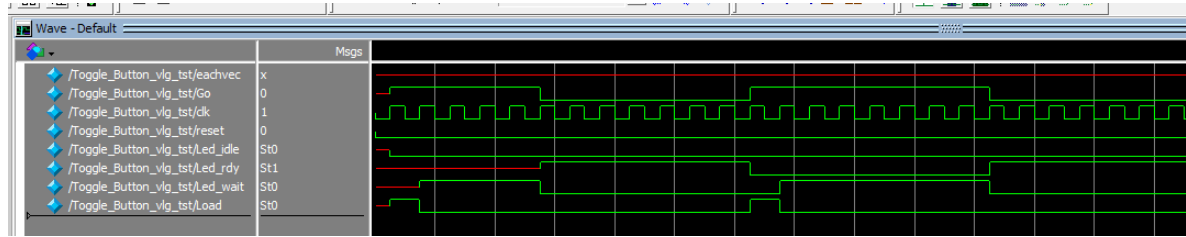
Line 7 to Line 14: This part we using if...else... statements to control the state of this model, the state are following the above ASM chart.

Line 17 to Line 39: This part is a case statement. Cooperating with state register in Line 2. For each state, this part statements used for control output ports.

Toggle Bottom Testbench

```
1  `timescale 1 ps/ 1 ps
2  module Toggle_Button_vlg_tst();
3  // constants
4  // general purpose registers
5  reg eachvec;
6  // test vector input registers
7  reg Go;
8  reg clk;
9  reg reset;
10 // wires
11 wire Led_idle;
12 wire Led_rdy;
13 wire Led_wait;
14 wire Load;
15
16 // assign statements (if any)
17 Toggle_Button i1 (
18 // port map - connection between master ports and signals/registers
19     .Go(Go),
20     .Led_idle(Led_idle),
21     .Led_rdy(Led_rdy),
22     .Led_wait(Led_wait),
23     .Load(Load),
24     .clk(clk),
25     .reset(reset)
26 );
27 initial
28 begin
29 // code that executes only once
30 // insert code here --> begin
31     reset = 1'b0;
32     clk = 1'b0;
33     forever #5 clk = ~clk;
34 // --> end
35 $display("Running testbench");
36 end
37
38 always
39 // optional sensitivity list
40 // @(event1 or event2 or .... eventn)
41 begin
42 // code executes for every event on sensitivity list
43 // insert code here --> begin
44 #5 Go = 1'b1;
45 #50 Go = 1'b0;
46 #70 Go = 1'b1;
47 #80 Go = 1'b0;
48 @eachvec;
49 // --> end
50 end
51 endmodule
```

Check the value of the three external LED lights and the Load signal to determine the status of this module. From the figure we can see that the change of the state of this module is in line with our expected ASM diagram.



Top Module Testbench

```

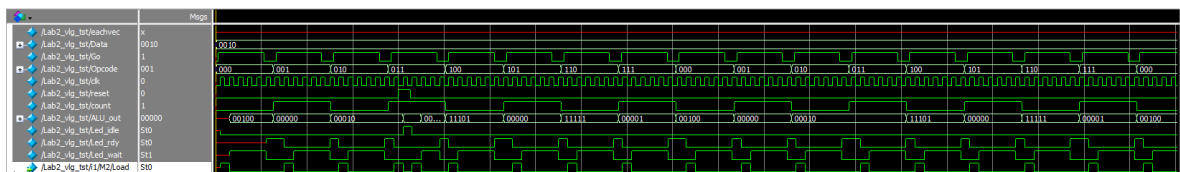
1  module Lab2_vlg_tst();
2  // constants
3  // general purpose registers
4  reg eachvec;
5  // test vector input registers
6  reg [3:0] Data;
7  reg Go;
8  reg [2:0] Opcode;
9  reg clk;
10 reg reset;
11 reg count;
12 // wires
13 wire [4:0] ALU_out;
14 wire Led_idle;
15 wire Led_rdy;
16 wire Led_wait;
17
18 // assign statements (if any)
19 Lab2 i1 (
20 // port map - connection between master ports and signals/registers
21     .ALU_out(ALU_out),
22     .Data(Data),
23     .Go(Go),
24     .Led_idle(Led_idle),
25     .Led_rdy(Led_rdy),
26     .Led_wait(Led_wait),
27     .Opcode(Opcode),
28     .clk(clk),
29     .reset(reset)
30 );
31 initial
32 begin
33 // code that executes only once
34 // insert code here --> begin
35
36     clk = 0;
37
38     Go = 0;
39     reset = 0;
40     Data = 4'b0;
41     Opcode = 3'b0;
42

```

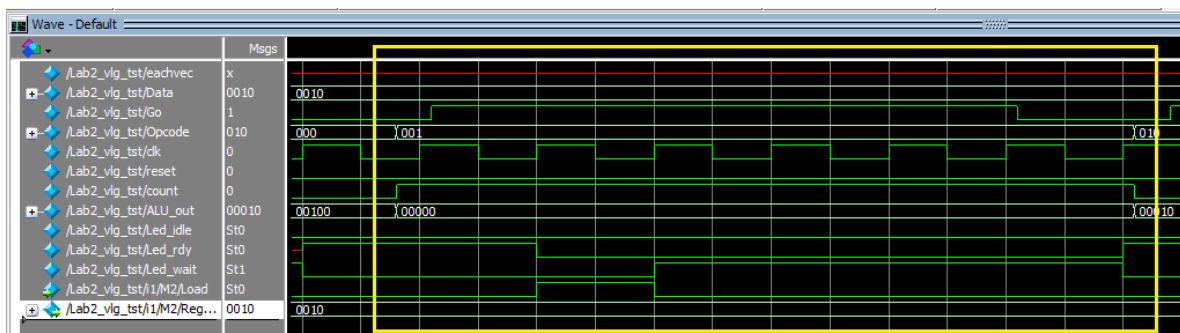
```

43     forever #5 clk = ~clk;
44
45 // --> end
46 $display("Running testbench");
47 end
48
49 initial begin
50 #200 reset = 1;
51 #13 reset = 0;
52 end
53
54
55 always
56 // optional sensitivity list
57 // @(event1 or event2 or .... eventn)
58 begin
59 // code executes for every event on sensitivity list
60 // insert code here --> begin
61
62 for(count = 3'b0 ; count<= 3'b111 ;count = count+1)
63     begin
64         Data = 4'b0010;
65         #3 Go = 1;
66         #50 Go = 0;
67         #10 Opcode= Opcode + 1;
68     end
69
70 @eachvec;
71 // --> end
72 end
73 endmodule

```



We using one operation explain in detail.



From the above figure we can see that **Data** is 0010, **Reg_out** which is last data in register is 0010, and **Opcode** is 001 which function is subtract.

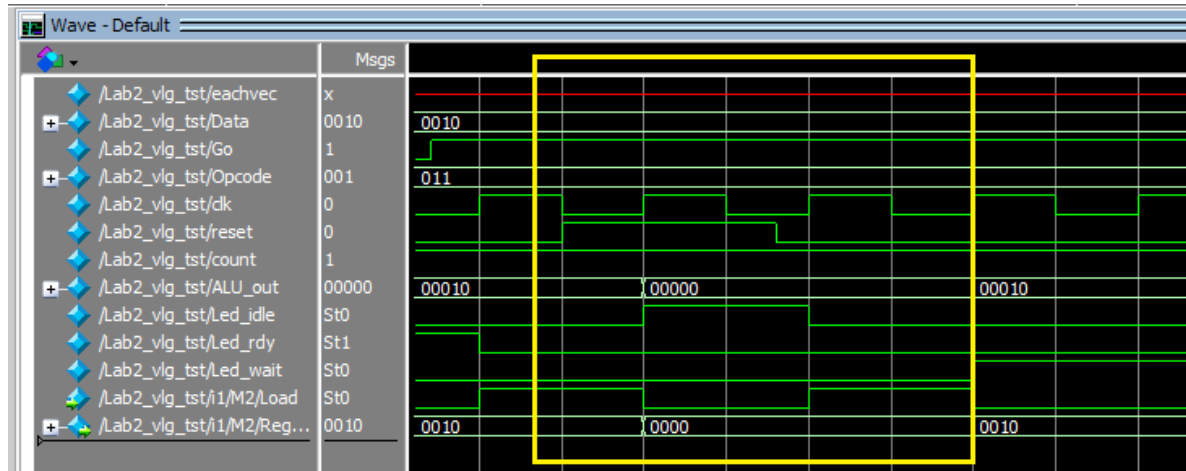
When **Go** signal is HIGH, in the next clock, **Led_rdy** from HIGH to LOW, at the same time, **Load** is from LOW to HIGH, to control the register store the data.

After next clock, **Load** signal becomes LOW, and **Led_wait** becomes HIGH, it is meaning the system is waiting to start calculate.

$$0010_2 - 0010_2 = 00000_2$$

We can seen the output is 00000, it is correct.

The, we test reset function:



From the wave figure we can seen after **reset** is HIGH, in next positive edge of clock, **Reg_out** is 0000, **Load** is LOW, **Led_idle** is HIGH, and **ALU_out** is 00000.

This result is our expected.

Resource Allocation

Liu Ziyang: Toggle Bottom module design and testbench, and experimental report writing.

Xu Zhikun: Top module design and testbench.

Zhu Yanxing: Top module design and toggle botton testbench.

Chen Dingrui: Register design and testbench.

Gong Chen: Toggle bottom module design and top module testbench.

Summary

In this lab, we design an ALU and its data channel by mastering the top-down module design method using Verilog HDL and the Quartus II EDA platform for design input, compilation, simulation of the whole process.

Source Code

Top Module - Lab2.v

```
1 module Lab2 (
2     output [4:0] ALU_out,
```

```

3     output Led_idle, Led_wait, Led_rdy,
4     input  [3:0] Data,
5     input  [2:0] Opcode,
6     input  Go,
7     input  clk,reset
8 );
9     wire [3:0] Reg_out;
10
11     ALU          M1 (ALU_out, Data, Reg_out, Opcode);
12     Register     M2 (Reg_out, Data, Load, clk, reset);
13     Toggle_Button M3 (Load, Led_idle, Led_wait, Led_rdy, Go, clk, reset);
14
15 endmodule
16

```

ALU - ALU.v

```

1 module ALU (output reg [4:0] ALU_out, input [3:0] Data_A, Data_B, input [2:0] Opcode);
2     parameter Add      = 3'b000,    // A + B
3               Subtract  = 3'b001,    // A - B
4               Or_AB     = 3'b010,    // A | B
5               And_AB    = 3'b011,    // A & B
6               Not_A     = 3'b100,    // ~ A
7               Exor      = 3'b101,    // A ^ B
8               Exnor     = 3'b110,    // A ~ ^ B
9               Ror_A     = 3'b111;    // | A
10
11     always@(*)
12
13     case(Opcode)
14         Add:          ALU_out = Data_A + Data_B;
15         Subtract:    ALU_out = Data_A - Data_B;
16         Or_AB:       ALU_out = Data_A | Data_B;
17         And_AB:      ALU_out = Data_A & Data_B;
18         Not_A:       ALU_out = ~ Data_A;
19         Exor:        ALU_out = Data_A ^ Data_B;
20         Exnor:       ALU_out = Data_A ~^ Data_B;
21         Ror_A:       ALU_out <= {Data_A[0],Data_A[3:1]};
22     endcase
23 endmodule

```

Register - Register.v

```

1  module Register (output reg [3:0] Reg_out, input [3:0] Data, input Load, clk, reset);
2      always @(posedge clk) begin
3
4          if(reset) Reg_out = 4'b0;
5          else begin
6              if(Load) Reg_out <= Data ;
7          end
8      end
9  endmodule
10

```

Toggle Bottom - Toggle_Button.v

```

1  module Toggle_Button (output reg Load, Led_idle, Led_wait, Led_rdy, input Go, clk, reset);
2      reg [1:0] state = 0;
3
4      always @(posedge clk) begin
5
6          if(reset) state = 0;
7          else begin
8              if(Go && state == 0) state = 2'b1;
9              else if(state == 1) state = 2'b10;
10             else if(!Go && state == 2'b10) state = 2'b11;
11             else if (Go && state == 2'b11) begin
12                 state = 2'b1;
13                 Led_rdy = 0;
14             end
15         end
16
17         case(state)
18             2'b0:          begin
19
20                 Led_idle = 1;
21                 Led_wait = 0;
22                 Led_rdy = 0;
23                 Load = 0;
24             end
25             2'b1:          begin
26
27                 Led_idle = 0;
28                 Load = 1;
29             end
30             2'b10:         begin
31
32                 Load = 0;
33                 Led_wait = 1;
34             end
35             2'b11:         begin
36
37                 Led_wait = 0;
38                 Led_rdy = 1;
39             end
40         endcase
41     end

```

```
41 endmodule
42
```

Toggle Bottom Test Bench

```
1 // Copyright (C) 2018 Intel Corporation. All rights reserved.
2 // Your use of Intel Corporation's design tools, logic functions
3 // and other software and tools, and its AMPP partner logic
4 // functions, and any output files from any of the foregoing
5 // (including device programming or simulation files), and any
6 // associated documentation or information are expressly subject
7 // to the terms and conditions of the Intel Program License
8 // Subscription Agreement, the Intel Quartus Prime License Agreement,
9 // the Intel FPGA IP License Agreement, or other applicable license
10 // agreement, including, without limitation, that your use is for
11 // the sole purpose of programming logic devices manufactured by
12 // Intel and sold by Intel or its authorized distributors. Please
13 // refer to the applicable agreement for further details.
14
15 // *****
16 // This file contains a Verilog test bench template that is freely editable to
17 // suit user's needs .Comments are provided in each section to help the user
18 // fill out necessary details.
19 // *****
20 // Generated on "10/29/2019 20:29:16"
21
22 // Verilog Test Bench template for design : Toggle_Button
23 //
24 // Simulation tool : ModelSim-Altera (Verilog)
25 //
26
27 `timescale 1 ps/ 1 ps
28 module Toggle_Button_vlg_tst();
29 // constants
30 // general purpose registers
31 reg eachvec;
32 // test vector input registers
33 reg Go;
34 reg clk;
35 reg reset;
36 // wires
37 wire Led_idle;
38 wire Led_rdy;
39 wire Led_wait;
40 wire Load;
41
42 // assign statements (if any)
43 Toggle_Button i1 (
44 // port map - connection between master ports and signals/registers
45     .Go(Go),
46     .Led_idle(Led_idle),
47     .Led_rdy(Led_rdy),
48     .Led_wait(Led_wait),
```

```

49     .Load(Load),
50     .clk(clk),
51     .reset(reset)
52 );
53 initial
54 begin
55     // code that executes only once
56     // insert code here --> begin
57     reset = 1'b0;
58     clk = 1'b0;
59     forever #5 clk = ~clk;
60 // --> end
61 $display("Running testbench");
62 end
63
64 always
65 // optional sensitivity list
66 // @(event1 or event2 or .... eventn)
67 begin
68 // code executes for every event on sensitivity list
69 // insert code here --> begin
70 #5 Go = 1'b1;
71 #50 Go = 1'b0;
72 #70 Go = 1'b1;
73 #80 Go = 1'b0;
74 @eachvec;
75 // --> end
76 end
77 endmodule
78
79

```

Top Module Test Bench

```

1 // Copyright (C) 2018 Intel Corporation. All rights reserved.
2 // Your use of Intel Corporation's design tools, logic functions
3 // and other software and tools, and its AMPP partner logic
4 // functions, and any output files from any of the foregoing
5 // (including device programming or simulation files), and any
6 // associated documentation or information are expressly subject
7 // to the terms and conditions of the Intel Program License
8 // Subscription Agreement, the Intel Quartus Prime License Agreement,
9 // the Intel FPGA IP License Agreement, or other applicable license
10 // agreement, including, without limitation, that your use is for
11 // the sole purpose of programming logic devices manufactured by
12 // Intel and sold by Intel or its authorized distributors. Please
13 // refer to the applicable agreement for further details.
14
15 // *****
16 // This file contains a Verilog test bench template that is freely editable to
17 // suit user's needs .Comments are provided in each section to help the user
18 // fill out necessary details.
19 // *****

```

```

20 // Generated on "11/11/2019 20:11:36"
21
22 // Verilog Test Bench template for design : Lab2
23 //
24 // Simulation tool : ModelSim-Altera (Verilog)
25 //
26
27 `timescale 1 ps/ 1 ps
28 module Lab2_vlg_tst();
29 // constants
30 // general purpose registers
31 reg eachvec;
32 // test vector input registers
33 reg [3:0] Data;
34 reg Go;
35 reg [2:0] Opcode;
36 reg clk;
37 reg reset;
38 reg count;
39 // wires
40 wire [4:0] ALU_out;
41 wire Led_idle;
42 wire Led_rdy;
43 wire Led_wait;
44
45 // assign statements (if any)
46 Lab2 i1 (
47 // port map - connection between master ports and signals/registers
48     .ALU_out(ALU_out),
49     .Data(Data),
50     .Go(Go),
51     .Led_idle(Led_idle),
52     .Led_rdy(Led_rdy),
53     .Led_wait(Led_wait),
54     .Opcode(Opcode),
55     .clk(clk),
56     .reset(reset)
57 );
58 initial
59 begin
60 // code that executes only once
61 // insert code here --> begin
62
63     clk = 0;
64
65     Go = 0;
66     reset = 0;
67     Data = 4'b0;
68     Opcode = 3'b0;
69
70     forever #5 clk = ~clk;
71
72 // --> end
73 $display("Running testbench");
74 end

```

```
75
76 initial begin
77     #200 reset = 1;
78     #13 reset = 0;
79 end
80
81
82 always
83     // optional sensitivity list
84     // @(event1 or event2 or .... eventn)
85 begin
86     // code executes for every event on sensitivity list
87     // insert code here --> begin
88
89     for(count = 3'b0 ; count<= 3'b111 ;count = count+1)
90         begin
91             Data = 4'b0010;
92             #3 Go = 1;
93             #50 Go = 0;
94             #10 Opcode= Opcode + 1;
95         end
96
97 @eachvec;
98 // --> end
99 end
100 endmodule
101
102
```