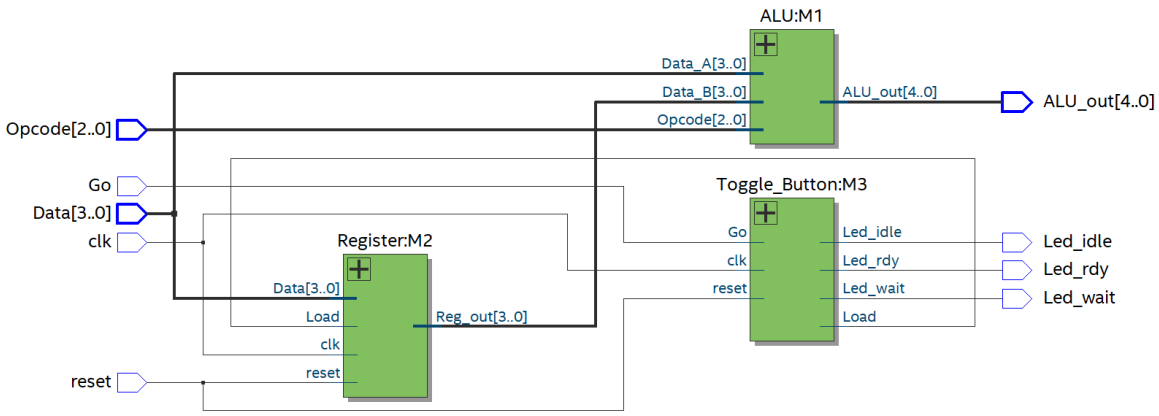
Lab 2 Report - Design and Simulation of ALU with sequential machine controlled datapath

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

```
module Lab2 (
    output [4:0] ALU_out,
    output Led_idle, Led_wait, Led_rdy,
    input [3:0] Data,
    input [2:0] Opcode,
    input Go,
    input clk,reset
);
    wire [3:0] Reg_out;

    ALU             M1 (ALU_out, Data, Reg_out, Opcode);
    Register        M2 (Reg_out, Data, Load, clk, reset);
    Toggle_Button   M3 (Load, Led_idle, Led_wait, Led_rdy, Go, clk, reset);

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 behaver of Register model are controlled by clock, the data will keeping in one clock cycle.

Toggle Bottom Design and Testbench

Toggle Bottom Design

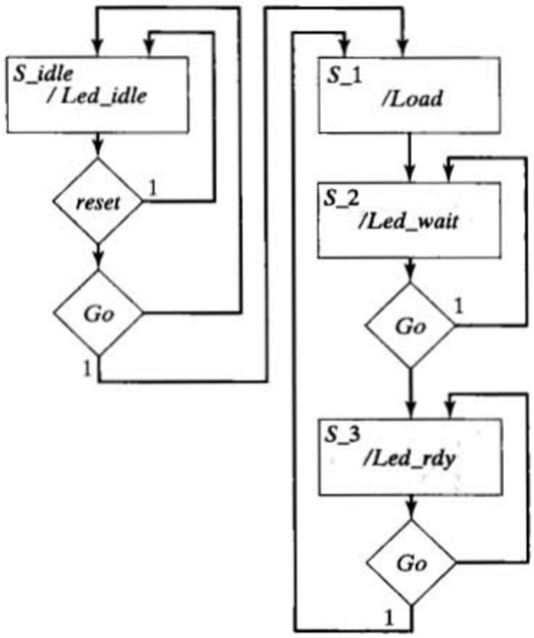
```
1  module Toggle_Button (output reg Load, Led_idle, Led_wait, Led_rdy, input Go, clk, reset);
2      reg [1:0] state = 0;
3
4      always @(posedge clk) begin
5
6          if(reset) state = 0;
7          else begin
8              if(Go && state == 0) state = 2'b1;
9              else if(state == 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_idel, 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.

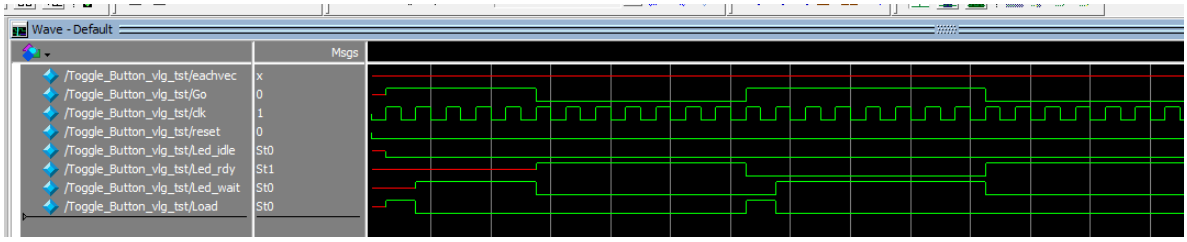
Line 7 to Line 14: This part we using `if...else...` statements to control the state of this model, the state are following the above ASM chart.

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

```
`timescale 1 ps/ 1 ps
module Toggle_Button_vlg_tst();
// constants
// general purpose registers
reg eachvec;
// test vector input registers
reg Go;
reg clk;
reg reset;
// wires
wire Led_idle;
wire Led_rdy;
wire Led_wait;
wire Load;

// assign statements (if any)
Toggle_Button i1 (
// port map - connection between master ports and signals/registers
    .Go(Go),
    .Led_idle(Led_idle),
    .Led_rdy(Led_rdy),
    .Led_wait(Led_wait),
    .Load(Load),
    .clk(clk),
    .reset(reset)
);
initial
begin
// code that executes only once
// insert code here --> begin
    reset = 1'b0;
    clk = 1'b0;
    forever #5 clk = ~clk;
// --> end
$display("Running testbench");
end

always
// optional sensitivity list
// @(event1 or event2 or .... eventn)
begin
// code executes for every event on sensitivity list
// insert code here --> begin
#5 Go = 1'b1;
#50 Go = 1'b0;
#70 Go = 1'b1;
#80 Go = 1'b0;
@eachvec;
// --> end
end
endmodule
```

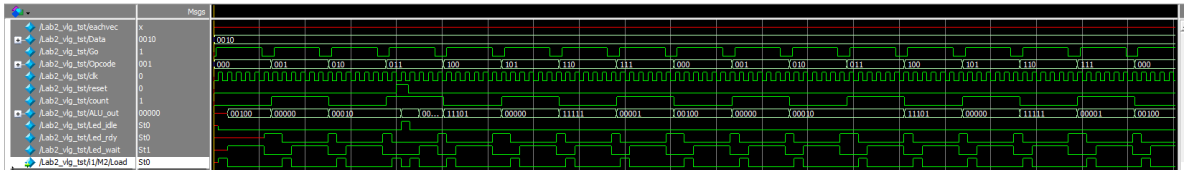
Check the value of the three external LED lights and the Load signal to determine the status of this module. From the figure we can see that the change of the state of this module is in line with our expected ASM diagram.



Top Module Testbench

```
module Lab2_vlg_tst();
// constants
// general purpose registers
reg eachvec;
// test vector input registers
reg [3:0] Data;
reg Go;
reg [2:0] Opcode;
reg clk;
reg reset;
reg count;
// wires
wire [4:0]  ALU_out;
wire Led_idle;
wire Led_rdy;
wire Led_wait;

// assign statements (if any)
Lab2 i1 (
// port map - connection between master ports and signals/registers
    .ALU_out(ALU_out),
    .Data(Data),
    .Go(Go),
    .Led_idle(Led_idle),
    .Led_rdy(Led_rdy),
    .Led_wait(Led_wait),
    .Opcode(Opcode),
    .clk(clk),
    .reset(reset)
);
initial
begin
// code that executes only once
// insert code here --> begin

    clk = 0;

    Go = 0;
    reset =0;
    Data = 4'b0;
    Opcode = 3'b0;

```

```
    forever #5 clk = ~clk;

// --> end
$display("Running testbench");
end

initial begin
#200 reset = 1;
#13 reset = 0;
end


always
// optional sensitivity list
// @(event1 or event2 or .... eventn)
begin
// code executes for every event on sensitivity list
// insert code here --> begin

for(count = 3'b0 ; count<= 3'b111 ;count = count+1)
    begin
        Data = 4'b0010;
        #3 Go = 1;
        #50 Go = 0;
        #10 Opcode= Opcode + 1;
    end

@eachvec;
// --> end
end
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.

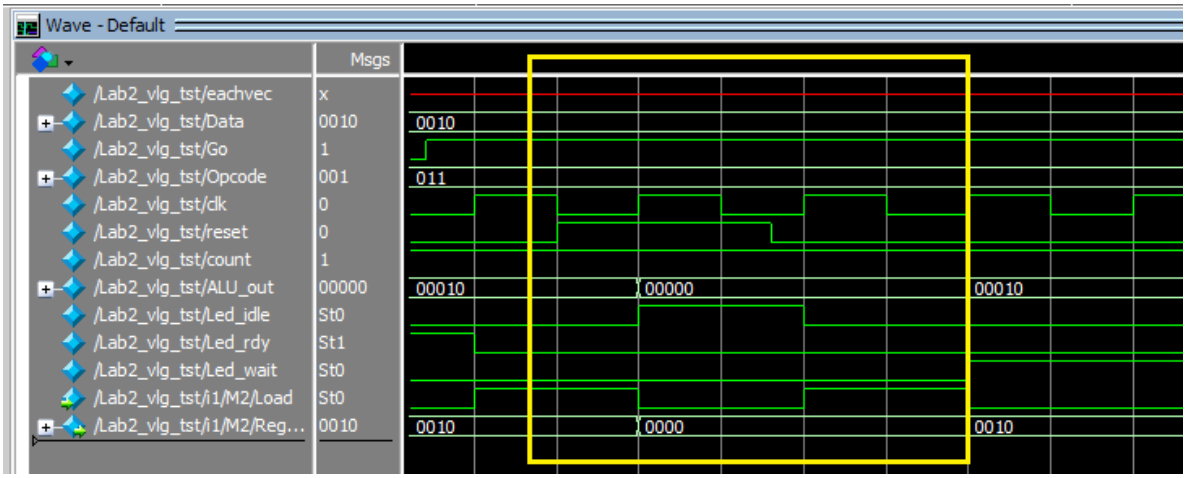
When **Go** signal is HIGH, in the next clock, **Led_rdy** from HIGH to LOW, at the same time, **Load** is from LOW to HIGH, to control the register store the data.

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,
```

```
    output Led_idle, Led_wait, Led_rdy,
    input [3:0] Data,
    input [2:0] Opcode,
    input Go,
    input clk,reset
);
    wire [3:0] Reg_out;

    ALU             M1 (ALU_out, Data, Reg_out, Opcode);
    Register            M2 (Reg_out, Data, Load, clk, reset);
    Toggle_Button   M3 (Load, Led_idle, Led_wait, Led_rdy, Go, clk, reset);

endmodule

```

ALU - ALU.v

```
module ALU (output reg [4:0] ALU_out, input [3:0] Data_A, Data_B, input [2:0] Opcode);
    parameter   Add     = 3'b000,   //  A + B
                Subtract    = 3'b001,   //  A - B
                Or_AB       = 3'b010,   //  A | B
                And_AB  = 3'b011,   //  A & B
                Not_A       = 3'b100,   //  ~ A
                Exor        = 3'b101,   //  A ^ B
                Exnor       = 3'b110,   //  A ~ ^ B
                Ror_A       = 3'b111;   //  | A

    always@(*)

    case(Opcode)
        Add:            ALU_out =  Data_A + Data_B;
        Subtract:   ALU_out =  Data_A - Data_B;
        Or_AB:      ALU_out =  Data_A | Data_B;
        And_AB:     ALU_out =  Data_A & Data_B;
        Not_A:      ALU_out =  ~ Data_A;
        Exor:           ALU_out =  Data_A ^ Data_B;
        Exnor:      ALU_out =  Data_A ~^ Data_B;
        Ror_A:      ALU_out <= {Data_A[0],Data_A[3:1]};
    endcase
endmodule
```

Register - Register.v

```
module Register (output reg [3:0] Reg_out, input [3:0] Data, input Load, clk, reset);
    always @(posedge clk) begin

        if(reset) Reg_out = 4'b0;
        else begin
            if(Load) Reg_out <= Data ;
        end
    end
endmodule

```

Toggle Bottom - Toggle_Button.v

```
module Toggle_Button (output reg Load, Led_idle, Led_wait, Led_rdy, input Go, clk, reset);
    reg [1:0] state = 0;

    always @(posedge clk) begin

        if(reset) state = 0;
        else begin
            if(Go && state == 0) state = 2'b1;
            else if(state == 1) state = 2'b10;
            else if(!Go && state == 2'b10) state = 2'b11;
            else if (Go && state == 2'b11) begin
                state = 2'b1;
                Led_rdy = 0;
            end
        end

        case(state)
            2'b0:               begin
                                    Led_idle = 1;
                                    Led_wait = 0;
                                    Led_rdy = 0;
                                    Load = 0;
                                end

            2'b1:           begin
                                    Led_idle = 0;
                                    Load = 1;
                                end

            2'b10:          begin
                                    Load = 0;
                                    Led_wait = 1;
                                end

            2'b11:          begin
                                    Led_wait = 0;
                                    Led_rdy = 1;
                                end
        endcase
    end
```

```
endmodule

```

Toggle Bottom Test Bench

```
// Copyright (C) 2018  Intel Corporation. All rights reserved.
// Your use of Intel Corporation's design tools, logic functions
// and other software and tools, and its AMPP partner logic
// functions, and any output files from any of the foregoing
// (including device programming or simulation files), and any
// associated documentation or information are expressly subject
// to the terms and conditions of the Intel Program License
// Subscription Agreement, the Intel Quartus Prime License Agreement,
// the Intel FPGA IP License Agreement, or other applicable license
// agreement, including, without limitation, that your use is for
// the sole purpose of programming logic devices manufactured by
// Intel and sold by Intel or its authorized distributors.  Please
// refer to the applicable agreement for further details.

// ***************************************************************************
// This file contains a Verilog test bench template that is freely editable to
// suit user's needs .Comments are provided in each section to help the user
// fill out necessary details.
// ***************************************************************************
// Generated on "10/29/2019 20:29:16"

// Verilog Test Bench template for design : Toggle_Button
//
// Simulation tool : ModelSim-Altera (Verilog)
//

`timescale 1 ps/ 1 ps
module Toggle_Button_vlg_tst();
// constants
// general purpose registers
reg eachvec;
// test vector input registers
reg Go;
reg clk;
reg reset;
// wires
wire Led_idle;
wire Led_rdy;
wire Led_wait;
wire Load;

// assign statements (if any)
Toggle_Button i1 (
// port map - connection between master ports and signals/registers
    .Go(Go),
    .Led_idle(Led_idle),
    .Led_rdy(Led_rdy),
    .Led_wait(Led_wait),
```

```
    .Load(Load),
    .clk(clk),
    .reset(reset)
);
initial
begin
// code that executes only once
// insert code here --> begin
    reset = 1'b0;
    clk = 1'b0;
    forever #5 clk = ~clk;
// --> end
$display("Running testbench");
end

always
// optional sensitivity list
// @(event1 or event2 or .... eventn)
begin
// code executes for every event on sensitivity list
// insert code here --> begin
#5 Go = 1'b1;
#50 Go = 1'b0;
#70 Go = 1'b1;
#80 Go = 1'b0;
@eachvec;
// --> end
end
endmodule


```

Top Module Test Bench

```
// Copyright (C) 2018  Intel Corporation. All rights reserved.
// Your use of Intel Corporation's design tools, logic functions
// and other software and tools, and its AMPP partner logic
// functions, and any output files from any of the foregoing
// (including device programming or simulation files), and any
// associated documentation or information are expressly subject
// to the terms and conditions of the Intel Program License
// Subscription Agreement, the Intel Quartus Prime License Agreement,
// the Intel FPGA IP License Agreement, or other applicable license
// agreement, including, without limitation, that your use is for
// the sole purpose of programming logic devices manufactured by
// Intel and sold by Intel or its authorized distributors.  Please
// refer to the applicable agreement for further details.

// ***************************************************************************
// This file contains a Verilog test bench template that is freely editable to
// suit user's needs .Comments are provided in each section to help the user
// fill out necessary details.
// ***************************************************************************
```

```
// Generated on "11/11/2019 20:11:36"

// Verilog Test Bench template for design : Lab2
//
// Simulation tool : ModelSim-Altera (Verilog)
//

`timescale 1 ps/ 1 ps
module Lab2_vlg_tst();
// constants
// general purpose registers
reg eachvec;
// test vector input registers
reg [3:0] Data;
reg Go;
reg [2:0] Opcode;
reg clk;
reg reset;
reg count;
// wires
wire [4:0]  ALU_out;
wire Led_idle;
wire Led_rdy;
wire Led_wait;

// assign statements (if any)
Lab2 i1 (
// port map - connection between master ports and signals/registers
    .ALU_out(ALU_out),
    .Data(Data),
    .Go(Go),
    .Led_idle(Led_idle),
    .Led_rdy(Led_rdy),
    .Led_wait(Led_wait),
    .Opcode(Opcode),
    .clk(clk),
    .reset(reset)
);
initial
begin
// code that executes only once
// insert code here --> begin

    clk = 0;

    Go = 0;
    reset =0;
    Data = 4'b0;
    Opcode = 3'b0;

    forever #5 clk = ~clk;

// --> end
$display("Running testbench");
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
```