Prelab for Lab6

Xiaoyu Zhou, 1004081147

Part I

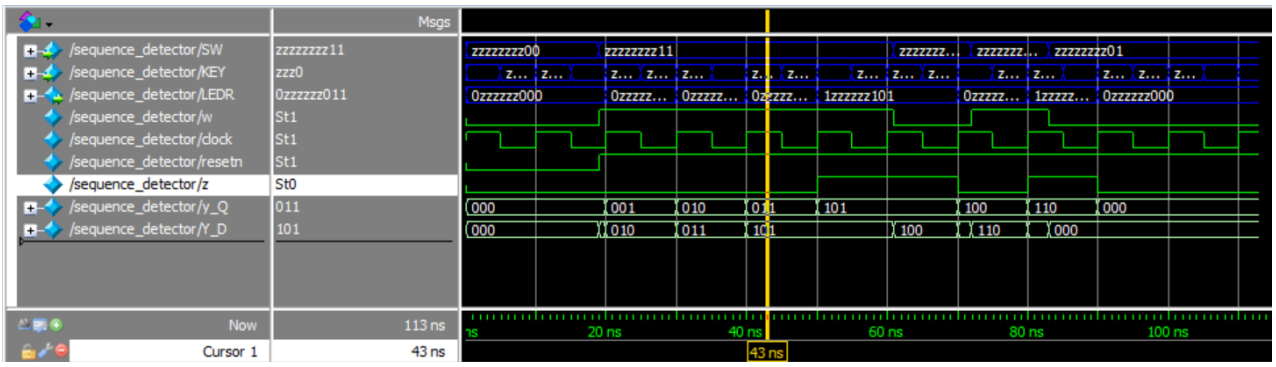
2. The **resetn** signal is a synchronous reset and it is active low. When I do the simulation, I need to begin with setting **resetn** to be 0.

3. Verilog Code:

```
// SW[0]:           reset signal
// SW[1]:           input signal (w)

// KEY[0]:          clock

// LEDR[2:0]:     current state
// LEDR[9]:         output (z)

module sequence_detector(SW, KEY, LEDR);
	input [9:0] SW;
	input [3:0] KEY;
	output [9:0] LEDR;

	wire w, clock, resetn, z;

	reg [2:0] y_Q, Y_D; // y_Q represents current state, Y_D represents next state

	localparam A = 3'b000, B = 3'b001, C = 3'b010, D = 3'b011, E = 3'b100, F = 3'b101, G = 3'b110;

	// Connect inputs and outputs to internal wires
	assign w = SW[1];
	assign clock = ~KEY[0];
	assign resetn = SW[0];
	assign LEDR[9] = z;
	assign LEDR[2:0] = y_Q;

	// State table
	// The state table should only contain the logic for state transitions
	// Do not mix in any output logic.    The output logic should be handled separately.
	// This will make it easier to read, modify and debug the code.
	always @(*)
	begin     // Start of state_table
		case (y_Q)
			A: begin
				if (!w) Y_D = A;
				else Y_D = B;
			   end
			B: begin
				if (!w) Y_D = A;
				else Y_D = C;
			   end
			C: begin
				if (!w) Y_D = E;
				else Y_D = D;
			   end
			D: begin
				if(!w) Y_D = E;
				else Y_D = F;
			   end
			E: begin
				if(!w) Y_D = A;
				else Y_D = G;
			   end
			F: begin
				if(!w) Y_D = E;
				else Y_D = F;
			   end
			G: begin
				if(!w) Y_D = A;
				else Y_D = C;
			   end
			default: Y_D = A;
		endcase
```

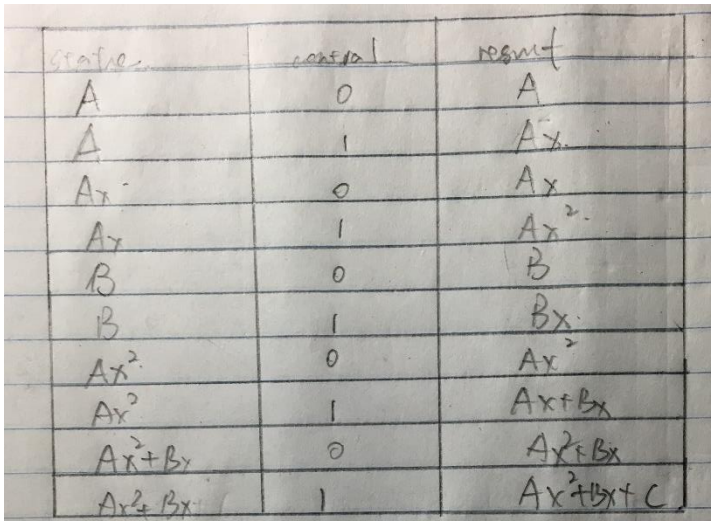
```
    end         // End of state_table

    // State Register (i.e., FFs)
    always @(posedge clock)
    begin     // Start of state_FFs (state register)
        if(resetn == 1'b0)
            y_Q <= A;
        else
            y_Q <= Y_D;
    end         // End of state_FFs (state register)

    // Output logic
    // Set z to 1 to turn on LED when in relevant states
    assign z = ((y_Q == F) || (y_Q == G));    // To be completed by you!
endmodule
```

4. Simulation.

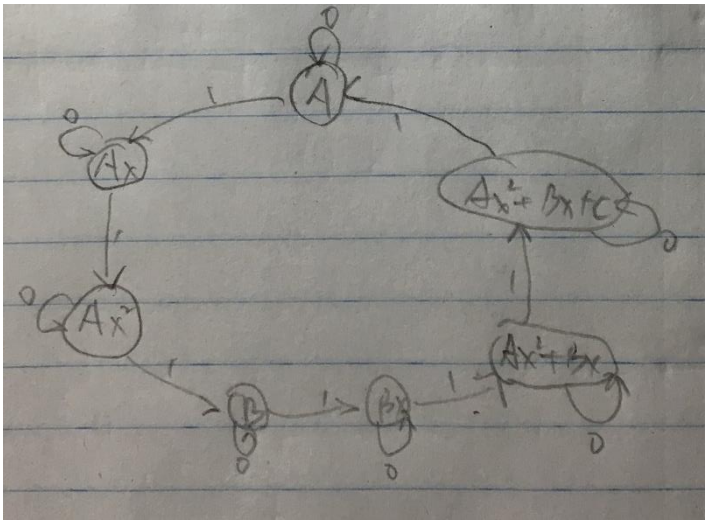


Part II

2. Table



3. State

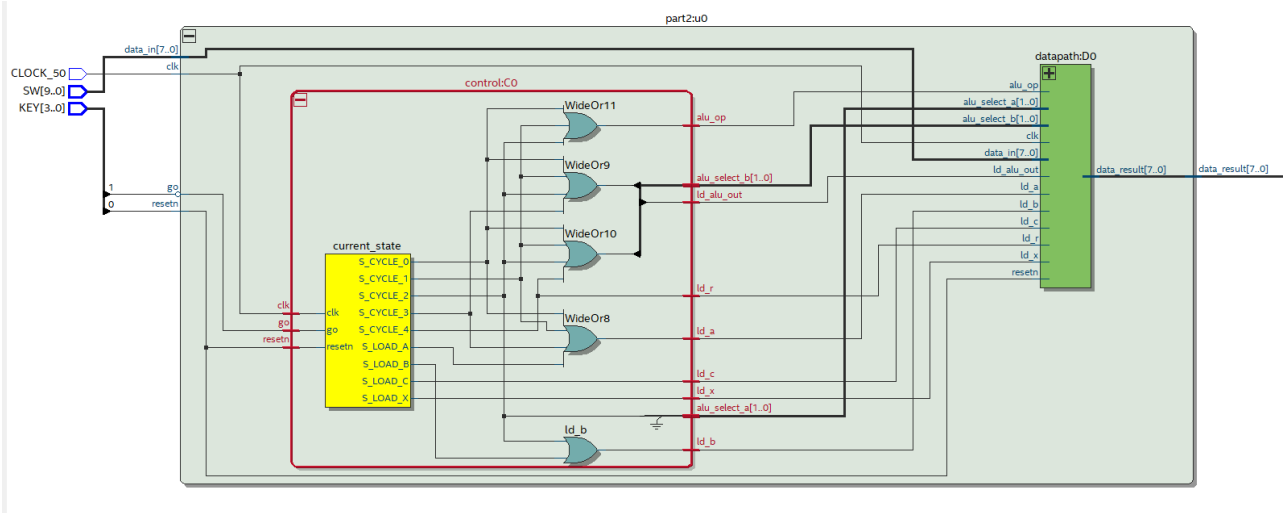


4. Verilog

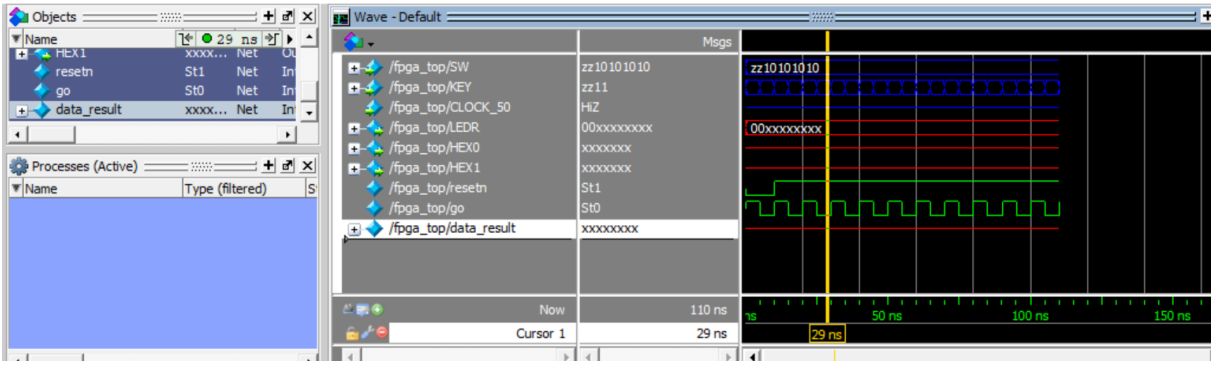
```
module control(
	input clk,
	input resetn,
	input go,

	output reg	ld_a, ld_b, ld_c, ld_x, ld_r,
	output reg	ld_alu_out,
	output reg [1:0]	alu_select_a, alu_select_b,
	output reg alu_op
	);

	reg [3:0] current_state, next_state;

	localparam	S_LOAD_A		= 4'd0,
			S_LOAD_A_WAIT	= 4'd1,
			S_LOAD_B		= 4'd2,
			S_LOAD_B_WAIT	= 4'd3,
			S_LOAD_C		= 4'd4,
			S_LOAD_C_WAIT	= 4'd5,
			S_LOAD_X		= 4'd6,
			S_LOAD_X_WAIT	= 4'd7,
			S_CYCLE_0		= 4'd8,
			S_CYCLE_1		= 4'd9,
				S_CYCLE_2		= 4'd10,
				S_CYCLE_3		= 4'd11,
			S_CYCLE_4		= 4'd12;

	// Next state logic aka our state table
	always@(*)
	begin: state_table
		case (current_state)
			S_LOAD_A: next_state = go ? S_LOAD_A_WAIT : S_LOAD_A; // Loop in current state until value is input
			S_LOAD_A_WAIT: next_state = go ? S_LOAD_A_WAIT : S_LOAD_B; // Loop in current state until go signal goes
low
			S_LOAD_B: next_state = go ? S_LOAD_B_WAIT : S_LOAD_B; // Loop in current state until value is input
			S_LOAD_B_WAIT: next_state = go ? S_LOAD_B_WAIT : S_LOAD_C; // Loop in current state until go signal goes
low
			S_LOAD_C: next_state = go ? S_LOAD_C_WAIT : S_LOAD_C; // Loop in current state until value is input
			S_LOAD_C_WAIT: next_state = go ? S_LOAD_C_WAIT : S_LOAD_X; // Loop in current state until go signal goes
low
			S_LOAD_X: next_state = go ? S_LOAD_X_WAIT : S_LOAD_X; // Loop in current state until value is input
			S_LOAD_X_WAIT: next_state = go ? S_LOAD_X_WAIT : S_CYCLE_0; // Loop in current state until go signal goes
low
			S_CYCLE_0: next_state = S_CYCLE_1;
				S_CYCLE_1: next_state = S_CYCLE_2;
				S_CYCLE_2: next_state = S_CYCLE_3;
				S_CYCLE_3: next_state = S_CYCLE_4;
			S_CYCLE_4: next_state = S_LOAD_A; // we will be done our two operations, start over after
		default:	next_state = S_LOAD_A;
		endcase
	end // state_table


	// Output logic aka all of our datapath control signals
	always @(*)
	begin: enable_signals
		// By default make all our signals 0
		ld_alu_out = 1'b0;
		ld_a = 1'b0;
		ld_b = 1'b0;
		ld_c = 1'b0;
		ld_x = 1'b0;
		ld_r = 1'b0;
		alu_select_a = 2'b00;
		alu_select_b = 2'b00;
		alu_op		= 1'b0;

		case (current_state)
			S_LOAD_A: begin
				ld_a = 1'b1;
				end
			S_LOAD_B: begin
				ld_b = 1'b1;
				end
			S_LOAD_C: begin
				ld_c = 1'b1;
				end
			S_LOAD_X: begin
				ld_x = 1'b1;
				end
			S_CYCLE_0: begin // Do A <- A * x
				ld_alu_out = 1'b1; ld_a = 1'b1; // store result back into A
				alu_select_a = 2'b00; // Select register A
```

```
                alu_select_b = 2'b11; // Also select register A
                alu_op = 1'b1; // Do multiply operation
            end
                S_CYCLE_1: begin // Do A <- Ax * x
                ld_alu_out = 1'b1; ld_a = 1'b1; // store result back into A
                alu_select_a = 2'b00; // Select register A
                alu_select_b = 2'b11; // Also select register A
                alu_op = 1'b1; // Do multiply operation
                end
                S_CYCLE_2: begin // Do B <- B * x
                ld_alu_out = 1'b1; ld_b = 1'b1; // store result back into B
                alu_select_a = 2'b01; // Select register B
                alu_select_b = 2'b11; // Also select register x
                alu_op = 1'b1; // Do multiply operation
                end
                S_CYCLE_3: begin // Do A <- Axx + Bx
                ld_alu_out = 1'b1; ld_a = 1'b1; // store result back into A
                alu_select_a = 2'b00; // Select register A
                alu_select_b = 2'b01; // Also select register B
                alu_op = 1'b0; // Do multiply operation
                end
            S_CYCLE_4: begin
                ld_r = 1'b1; // store result in result register
                alu_select_a = 2'b00; // Select register A
                alu_select_b = 2'b10; // Select register C
                alu_op = 1'b0; // Do Add operation
            end
        // default:       // don't need default since we already made sure all of our outputs were assigned a value at the start of the always block
        endcase
    end // enable_signals

    // current_state registers
    always@(posedge clk)
    begin: state_FFs
        if(!resetn)
            current_state <= S_LOAD_A;
        else
            current_state <= next_state;
    end // state_FFS
endmodule
```

5.



6. Simulation


Objects
Name
29 ns
HEX1
resetn
St1
Net
go
St0
Net
data_result
xxxx...
Net
Processes (Active)
Name
Type (filtered)
Wave - Default
Msgs
/fpga_top/SW
zz10101010
/fpga_top/KEY
zz11
/fpga_top/CLOCK_50
HiZ
/fpga_top/LEDR
00xxxxxxxx
/fpga_top/HEX0
xxxxxxx
/fpga_top/HEX1
xxxxxxx
/fpga_top/resetn
St1
/fpga_top/go
St0
/fpga_top/data_result
xxxxxxxx
Now
110 ns
Cursor 1
29 ns
50 ns
100 ns
150 ns