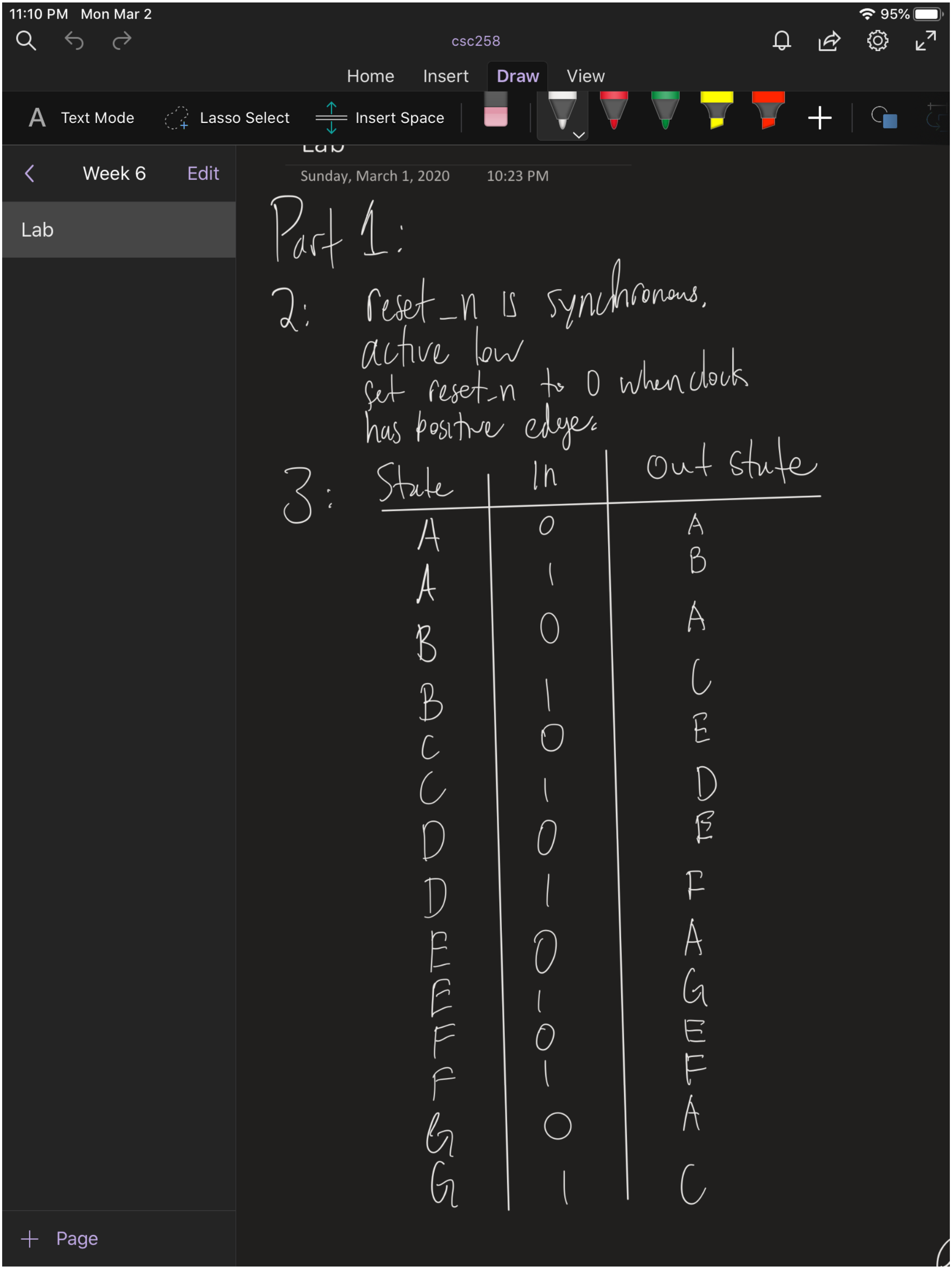


Lab

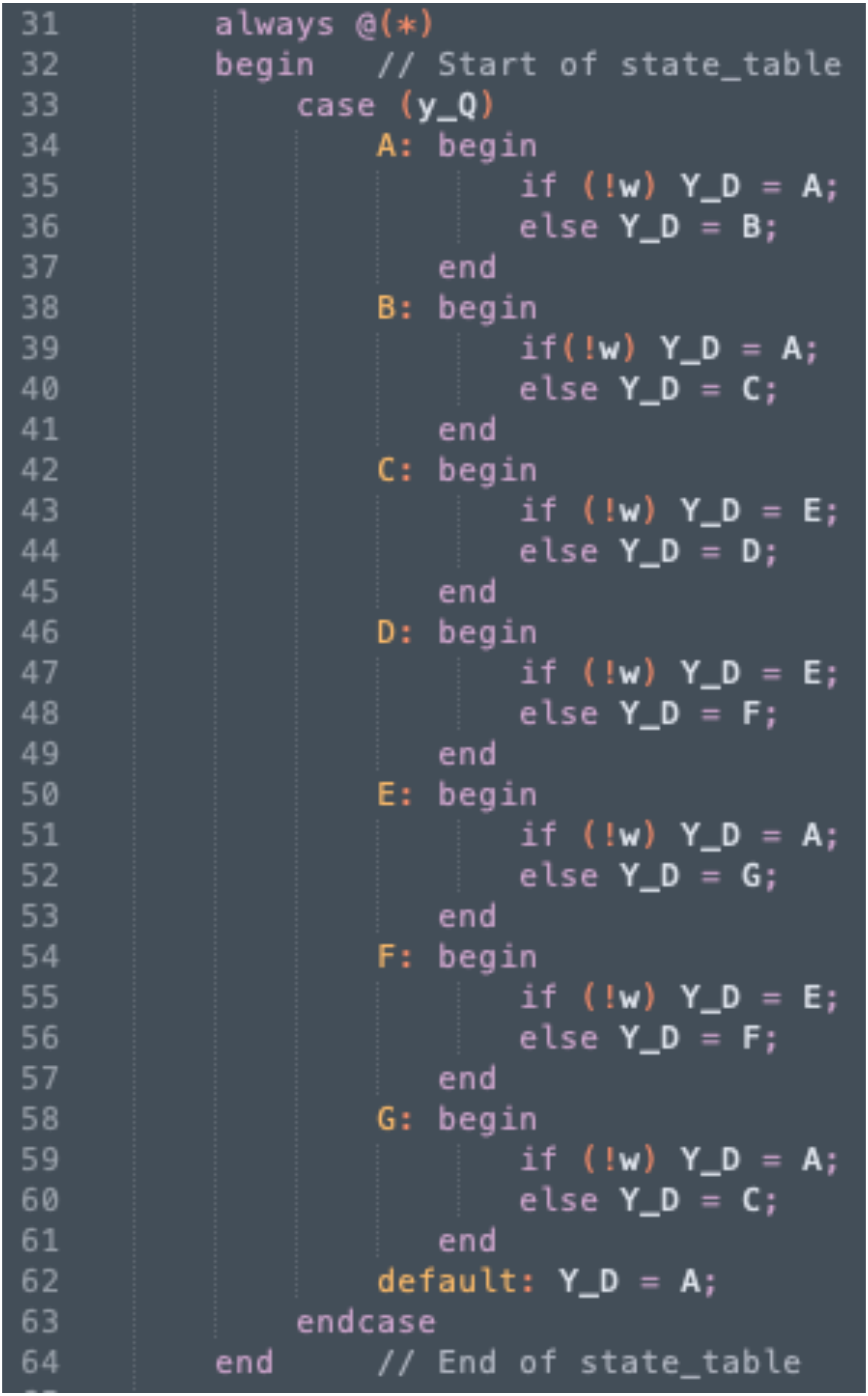
Sunday, March 1, 2020 10:23 PM

Part 1:

2: reset_n is synchronous,
active low
set reset_n to 0 when clock
has positive edge.

3:

State	In	Out state
A	0	A
A	1	B
B	0	A
B	1	C
C	0	E
C	1	D
D	0	E
D	1	F
E	0	A
E	1	G
F	0	E
F	1	F
G	0	A
G	1	C



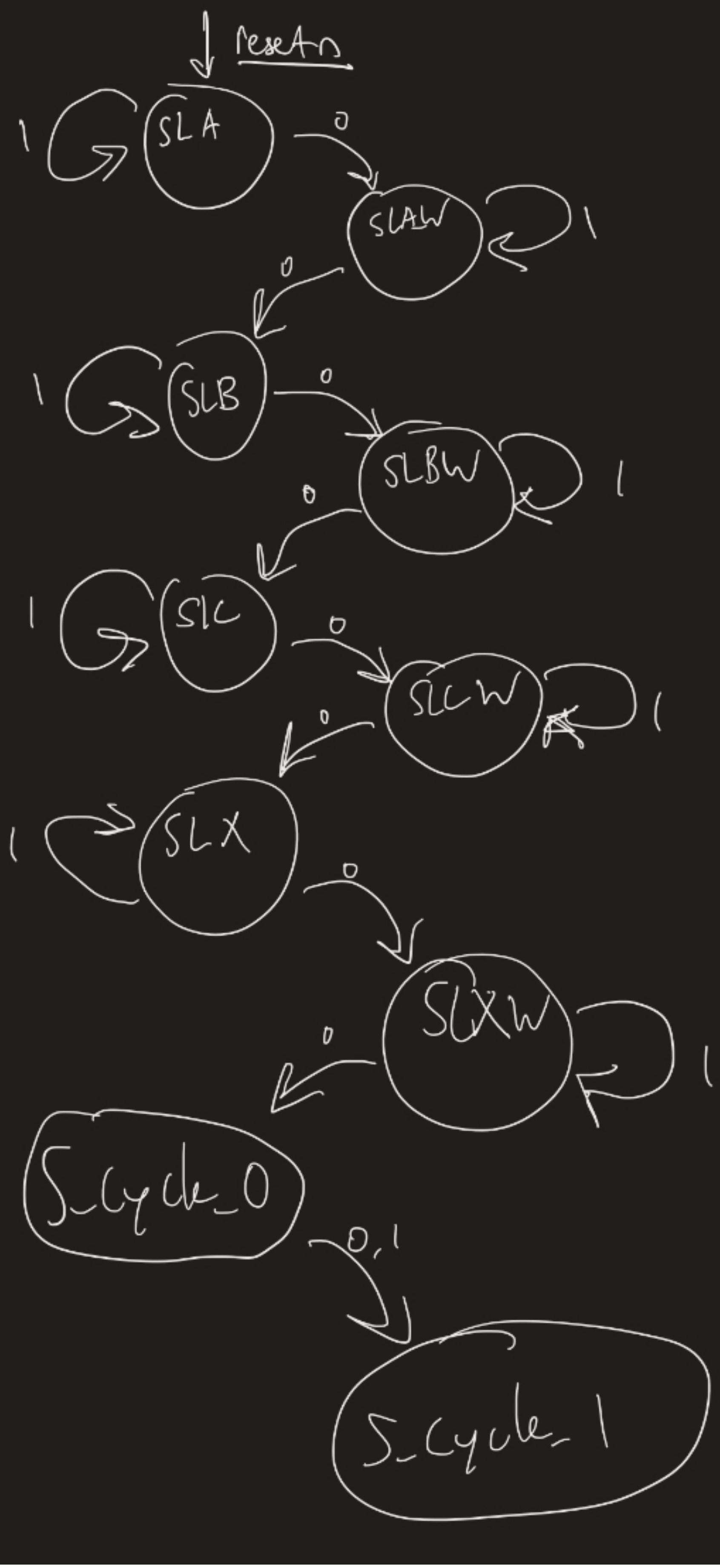
```
always @(*)
begin   // Start of state_table
    case (y_Q)
        A: begin
               if (!w) Y_D = A;
               else Y_D = B;
           end
        B: begin
               if(!w) Y_D = A;
               else Y_D = C;
           end
        C: begin
               if (!w) Y_D = E;
               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
```

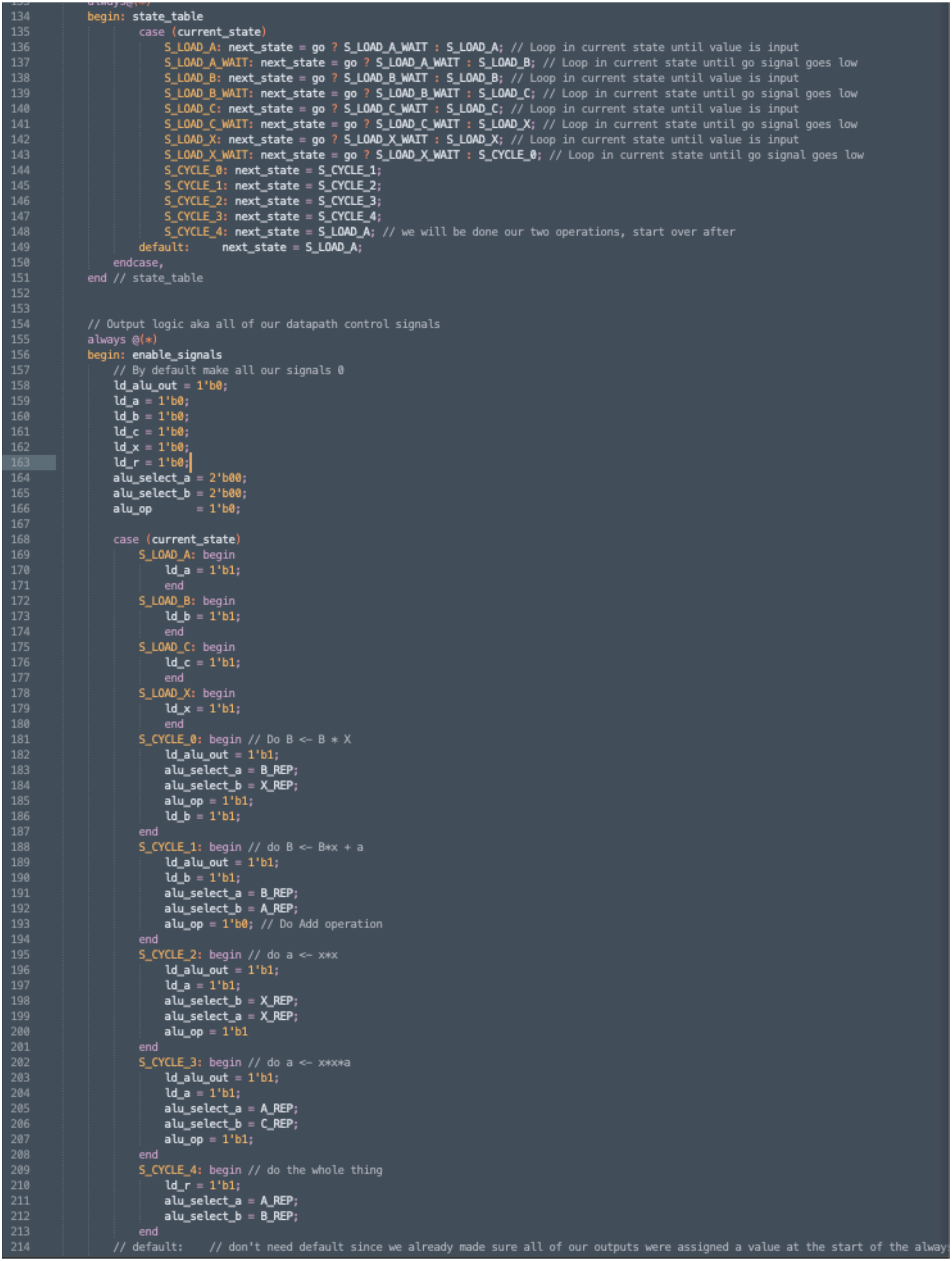
Part 2:

$Cx^2 + bx + a$ $\quad R_a \quad R_b \quad R_c \quad X$

High level functions	signals (relevant	register state (post cycle)
$b * x$ - save in R_b	ld_alu_out 1 ld_alu_a b ld_alu_b x alu_op 1 ld_b 1	$R_a = a$ $R_b = b \cdot x$ $R_c = c$ $R_x = x$ $R_{alu\,out} = b * x$ data resut?
$R_b + a$ - Save in R_b	ld_alue_out 1 ld_alu_a b ld_alu_b a alu_op 0 ld_b 1	$R_a = a$ $R_b = b \cdot x + a$ $R_c = c$ $R_x = x$ $R_{alu\,out} = b * x + a$ data result?
$x \cdot x$ - Save in R_a	ld_alu_out 1 ld_du_a a ld_alu_a a alu_op 1 ld_a 1	$R_a = x \cdot x$ $R_b = b \cdot x + a$ $R_c = c$ $R_x = x$ $R_{alu} = x \cdot x$ dadue result?
$R_a * C$ - Save in R_a	ld_alu_out 1 ld_alu_a a ld_alu_c c alu_op 1 ld_a 1	$R_a = x \cdot x \cdot c$ $R_b = b \cdot x + a$ $R_c = c$ $R_x = x$ $R_{alu} = x \cdot x \cdot c$ data resut?
$R_a + R_b$ - Save in data result.	ld_alu_out 0 ld_alu_a a ld_alu_b b alu_op 0 ld_r 1	$R_a = x \cdot x \cdot c$ $R_b = b \cdot x + \cancel{a}$ $R_x = x$ $R_c = c$ $R_{alu} = x \cdot x \cdot c + b \cdot x + a$ data_result = $R_a + R_b$

Fsm of provided (task 3)





```
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 B <- B * X
            ld_alu_out = 1'b1;
            alu_select_a = B_REP;
            alu_select_b = X_REP;
            alu_op = 1'b1;
            ld_b = 1'b1;
        end
        S_CYCLE_1: begin // do B <- B*x + a
            ld_alu_out = 1'b1;
            ld_b = 1'b1;
            alu_select_a = B_REP;
            alu_select_b = A_REP;
            alu_op = 1'b0; // Do Add operation
        end
        S_CYCLE_2: begin // do a <- x*x
            ld_alu_out = 1'b1;
            ld_a = 1'b1;
            alu_select_b = X_REP;
            alu_select_a = X_REP;
            alu_op = 1'b1
        end
        S_CYCLE_3: begin // do a <- x*x*a
            ld_alu_out = 1'b1;
            ld_a = 1'b1;
            alu_select_a = A_REP;
            alu_select_b = C_REP;
            alu_op = 1'b1;
        end
        S_CYCLE_4: begin // do the whole thing
            ld_r = 1'b1;
            alu_select_a = A_REP;
            alu_select_b = B_REP;
        end
    // default:    // don't need default since we already made sure all of our outputs were assigned a value at the start of the alway
```
]