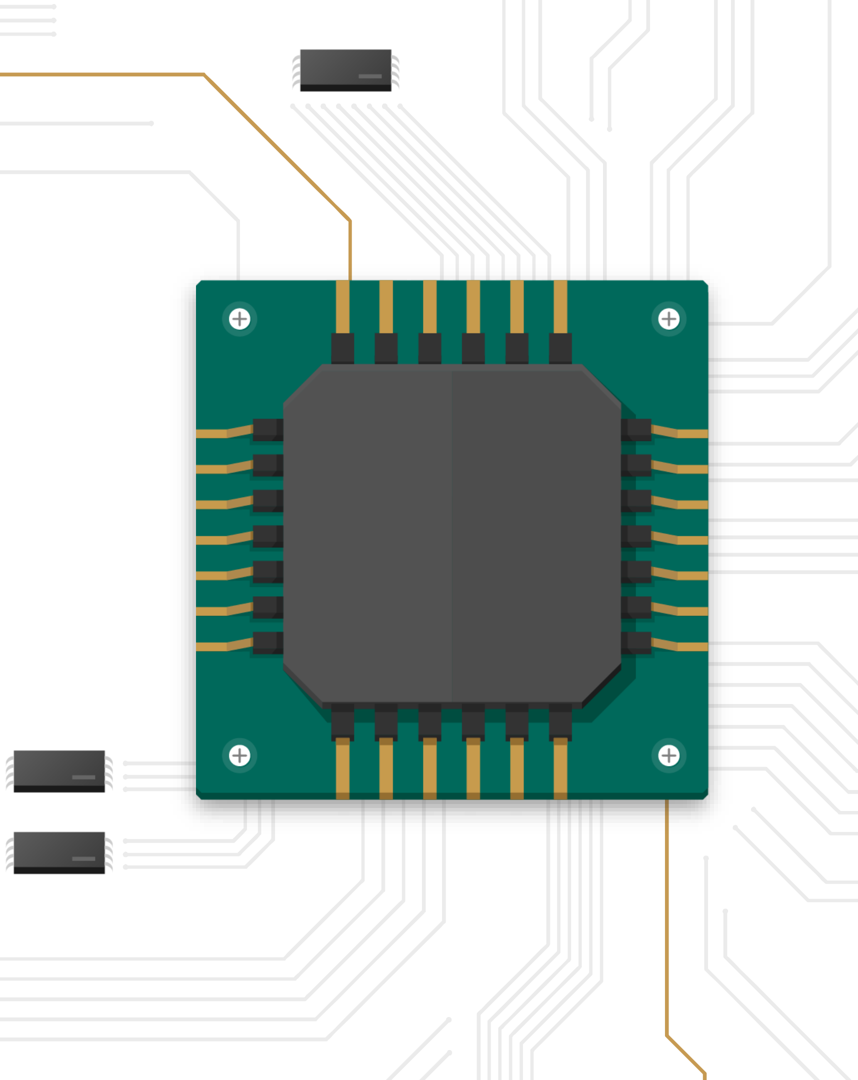
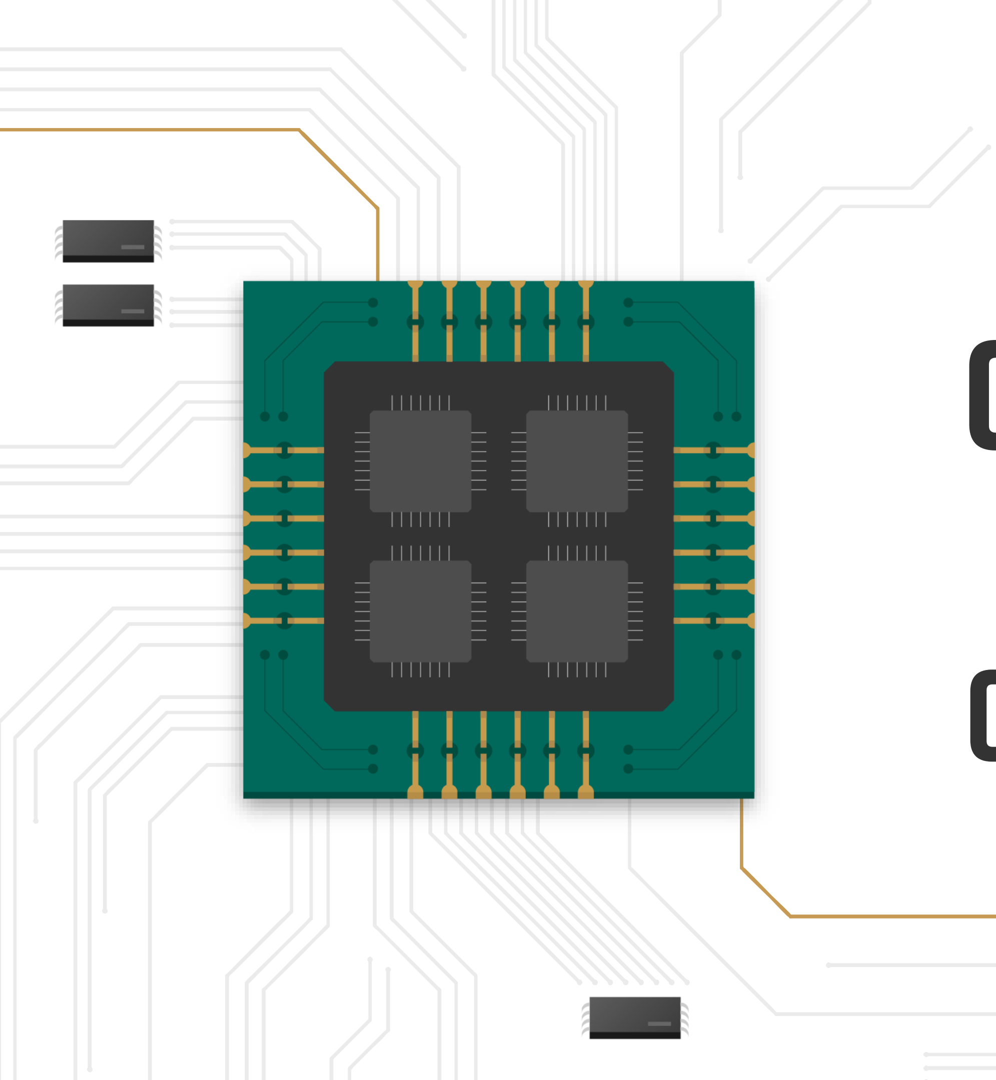
대한상공회의소
서울기술교육센터

# Stop Watch Using Basys 3



|  |  |
|---|---|
| 소속 | AI 시스템 반도체 설계 2기 |
| 이름 | 최현우 |

# Table of contents



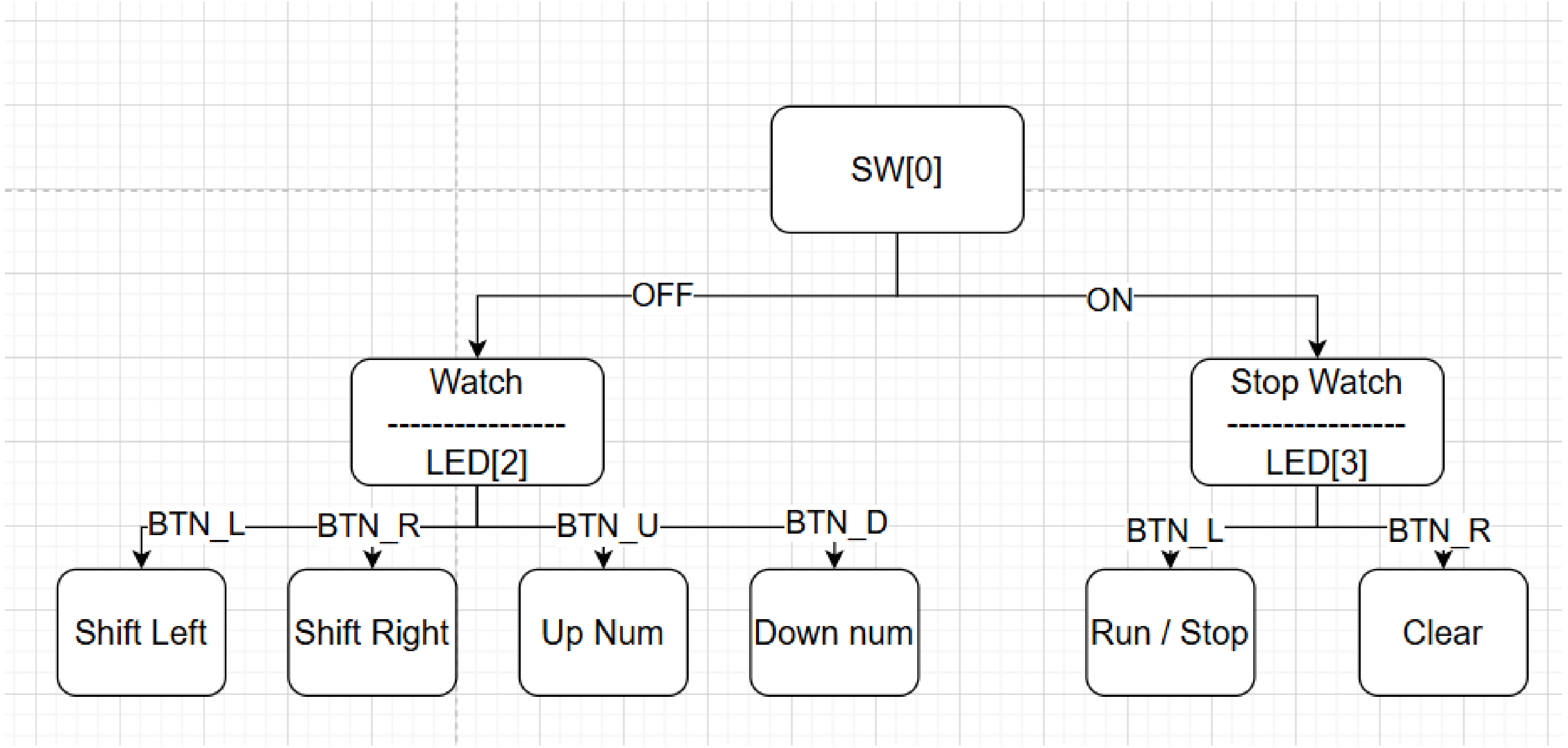
## 01

## Overview

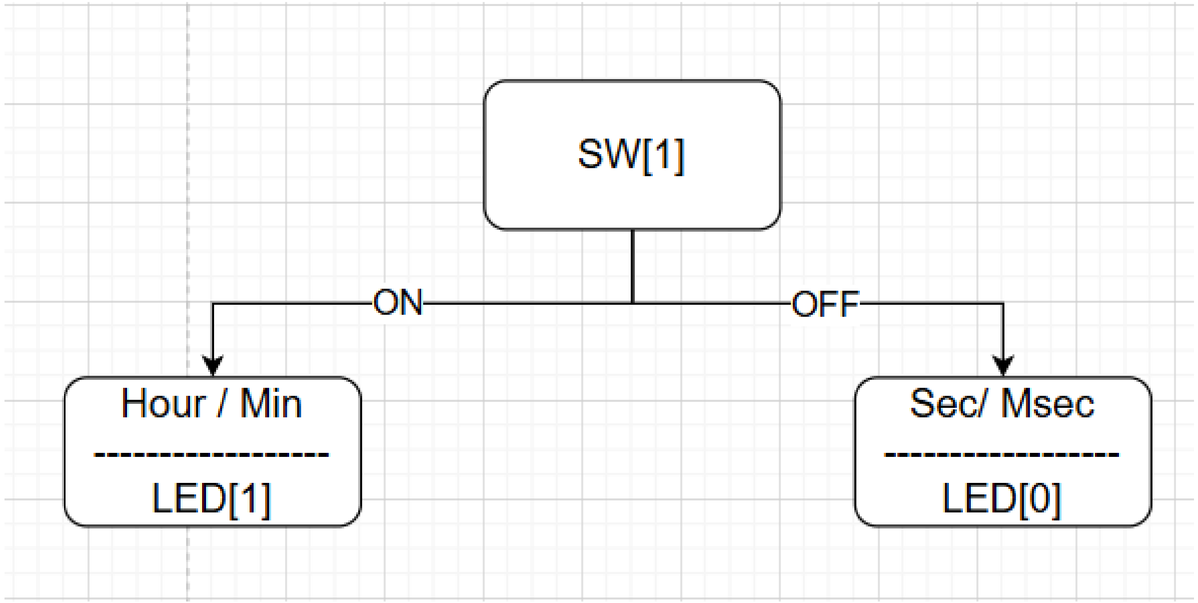
# Hardware & Environment

| | |
|---|---|
| **Hardware** | Basys 3 |
| **Environment** | Vivado 2023.2 & VSCODE |
| **Language** | Verilog |

# Function Specifications 1



# Function Specifications 2



# Function Update 1

| Origin | Update |
|---|---|
| 시계가 동작할 때 시간 조정<br><br>→ 정확한 시간을 맞추기 어려움 | 시계를 멈췄을 때만, 시간 조정 가능<br><br>→ SW[2] 사용 |

## Function Update 2

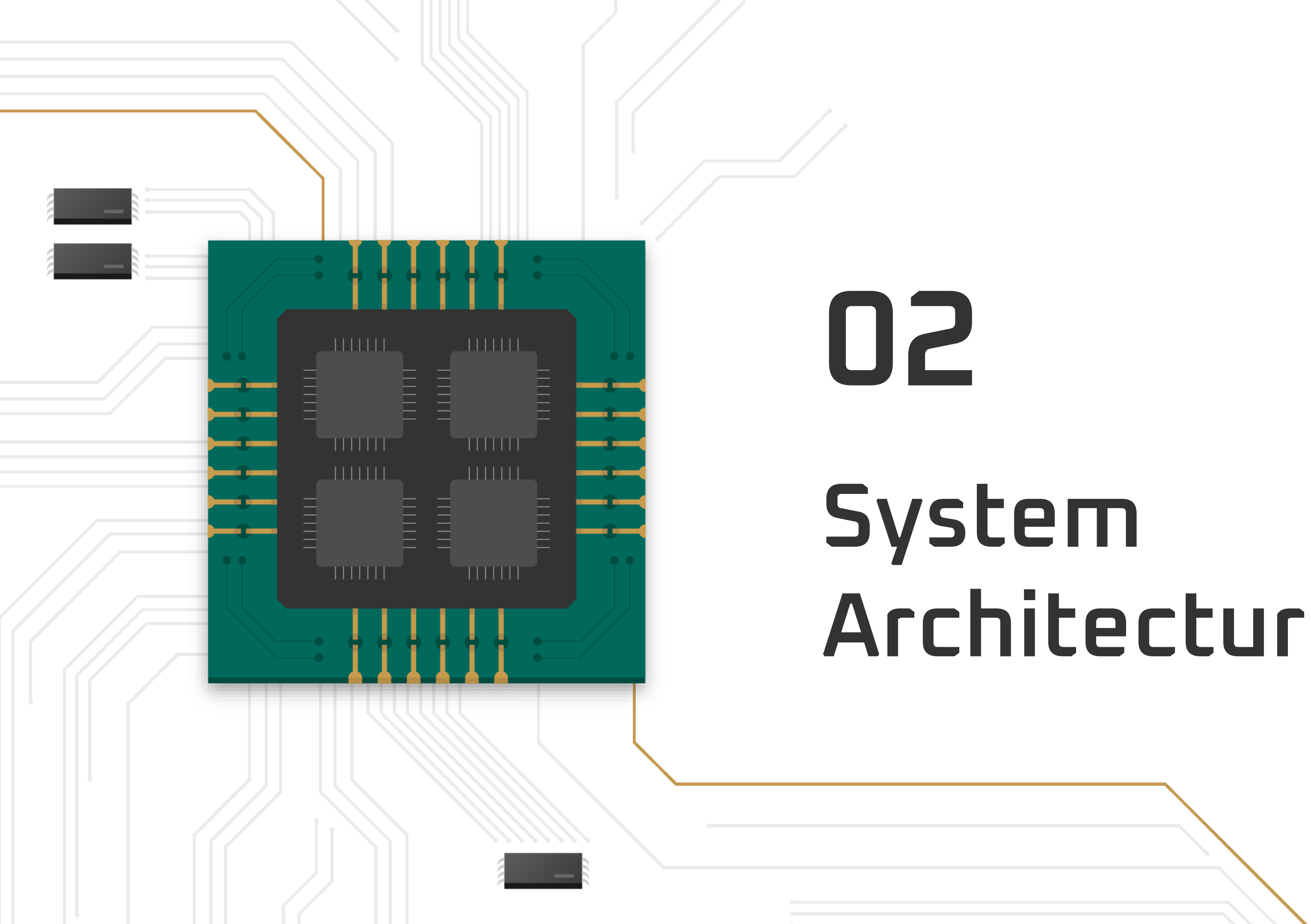
| Origin | Update |
|---|---|
| 좌, 우 버튼을 조정할 시간 선택<br><br>→ 시간의 일의 자리, 십의 자리 세부 조정 불가<br><br>→ 해당 시간 창에서 다른 시간을 바꿈 | 1. 시간의 십의 자리 및 일의 자리 세부 조정 가능<br><br>2. 해당 시간 창에서는 해당 시간만 조정 가능<br><br>Ex) sec창에선 sec만 조정 가능 |

## Function Update 3

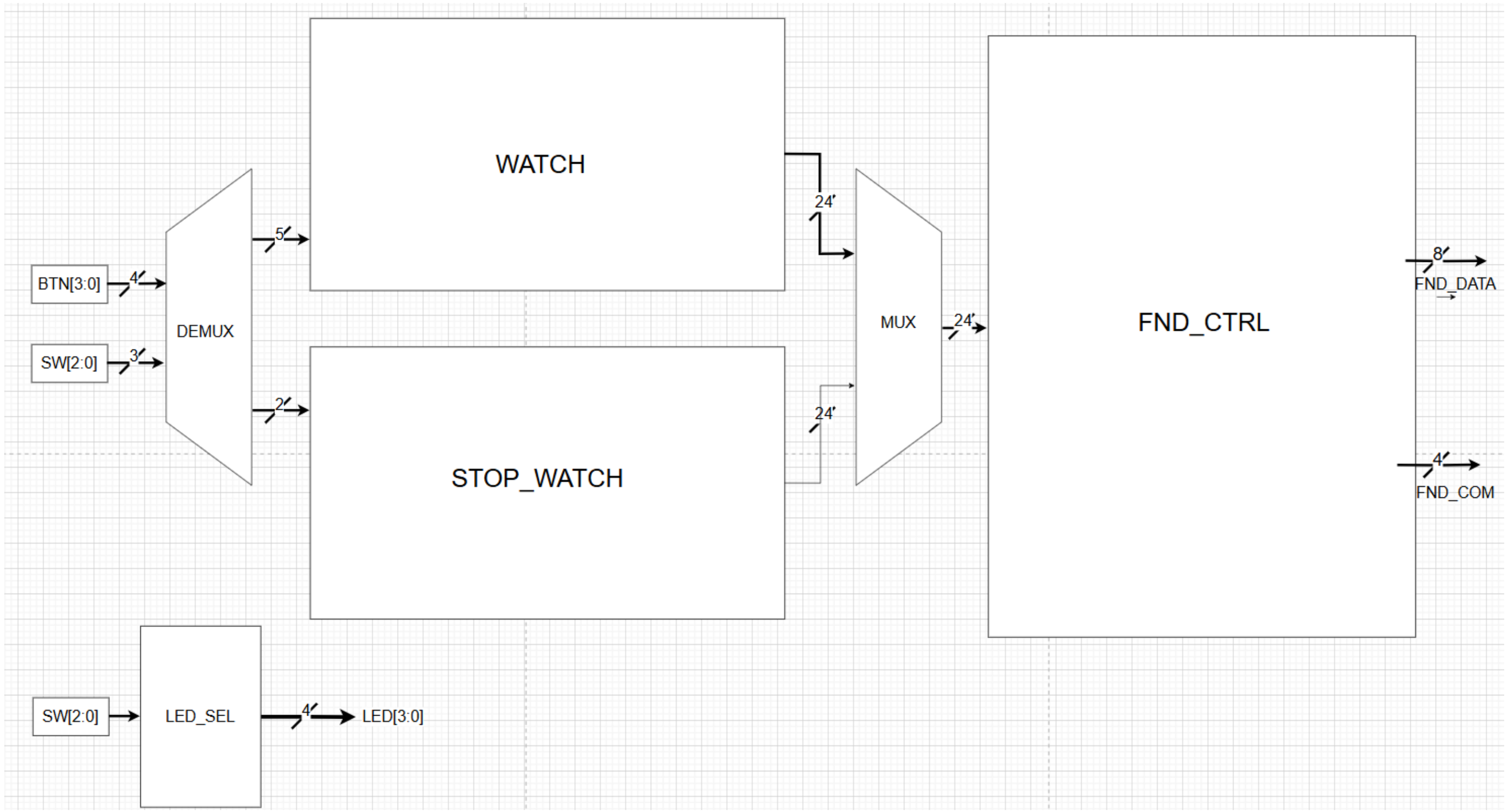
| Origin | Update |
|---|---|
| 현재 조정하고 있는 시간을 알 수 없음 | 현재 조정 중인 시간 창이 점멸<br><br>→ PWM 사용 |

## 02

## System Architecture

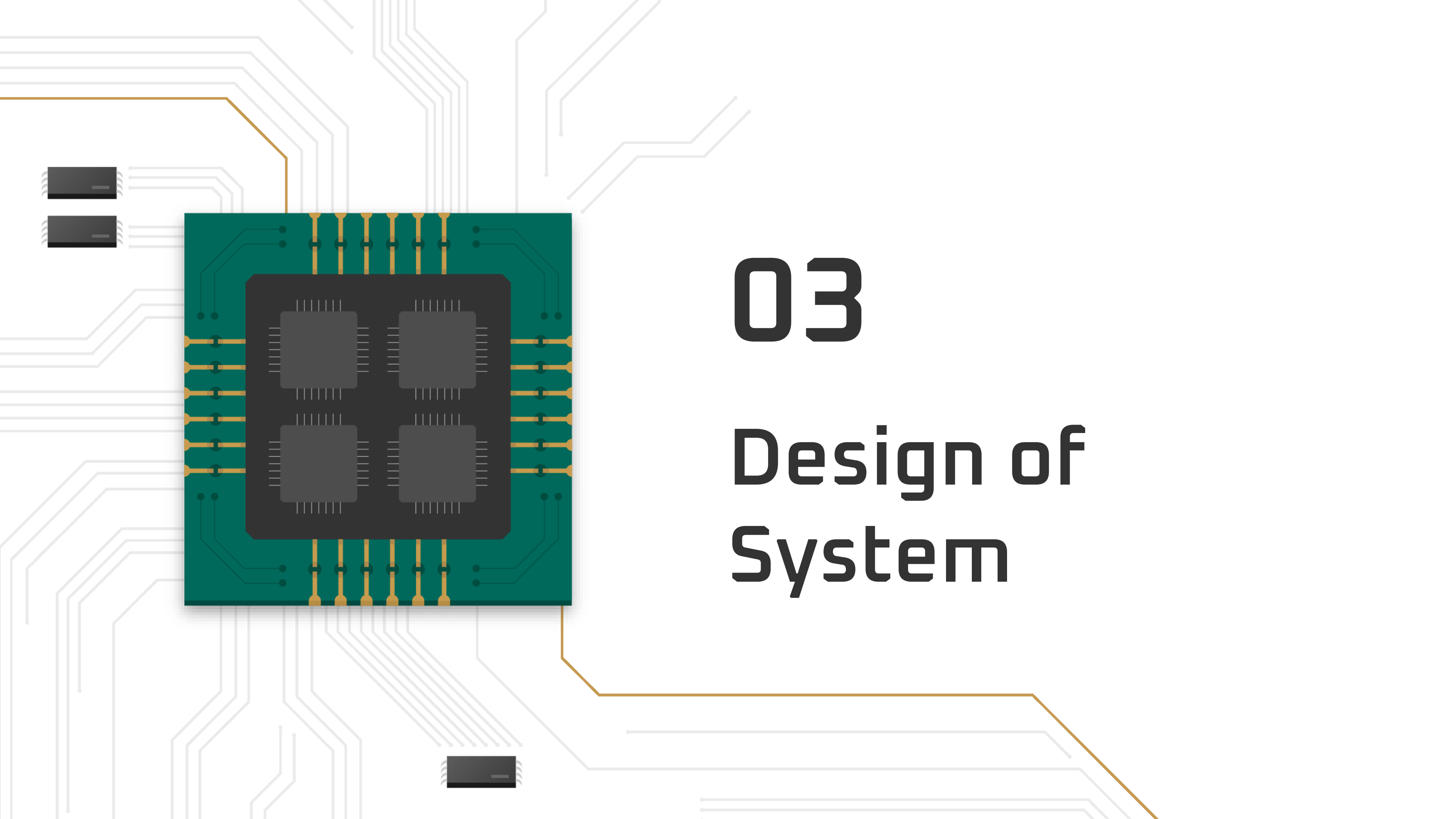


# Block Diagram(OverView)

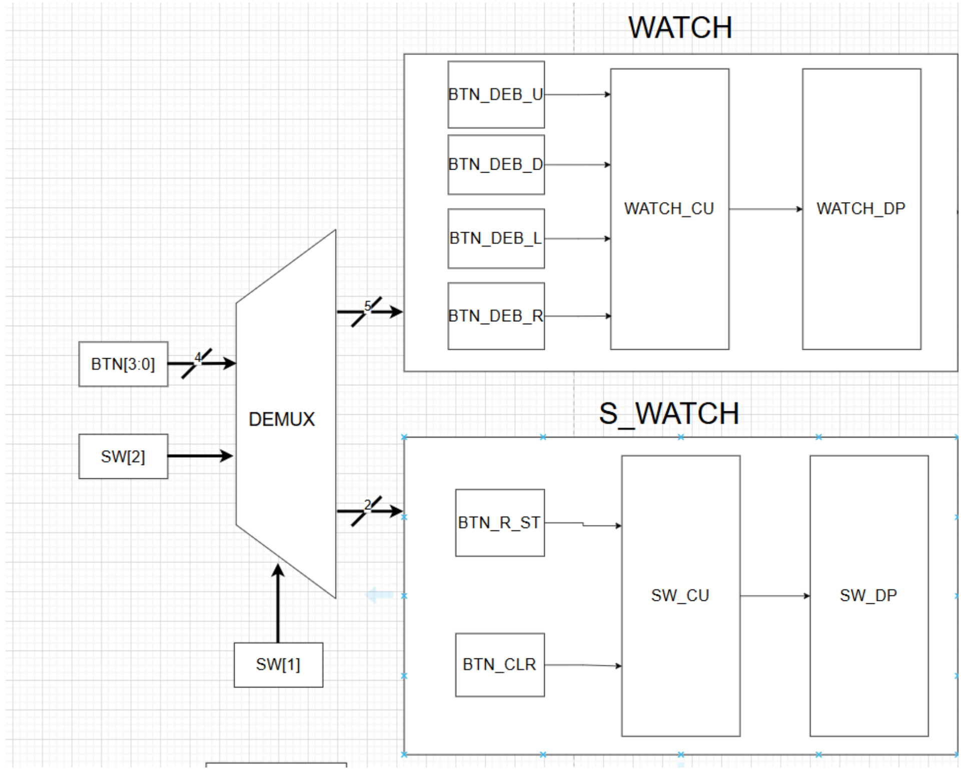


## 03

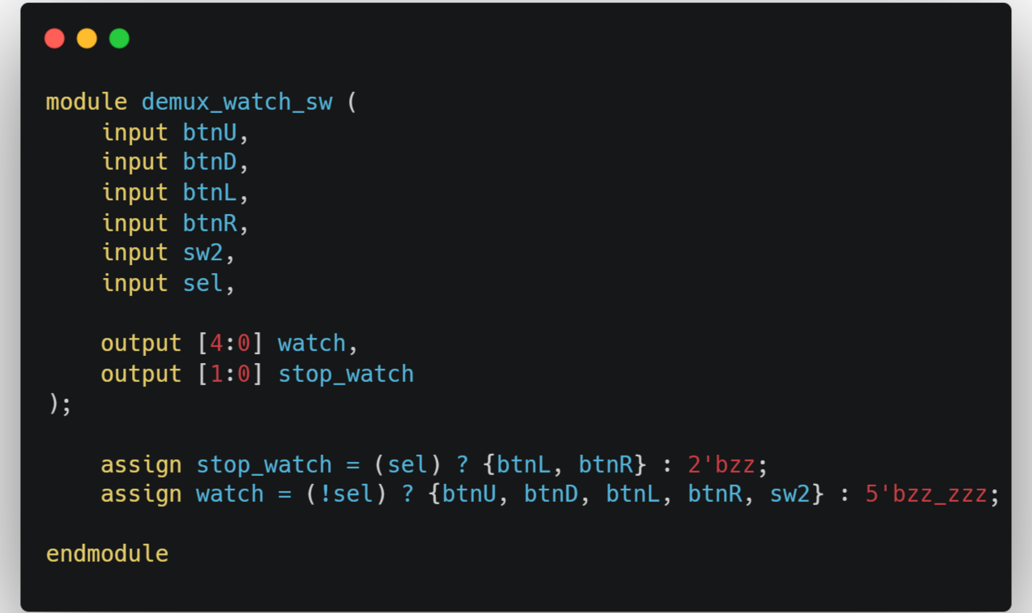
## Design of System



# DEMUX

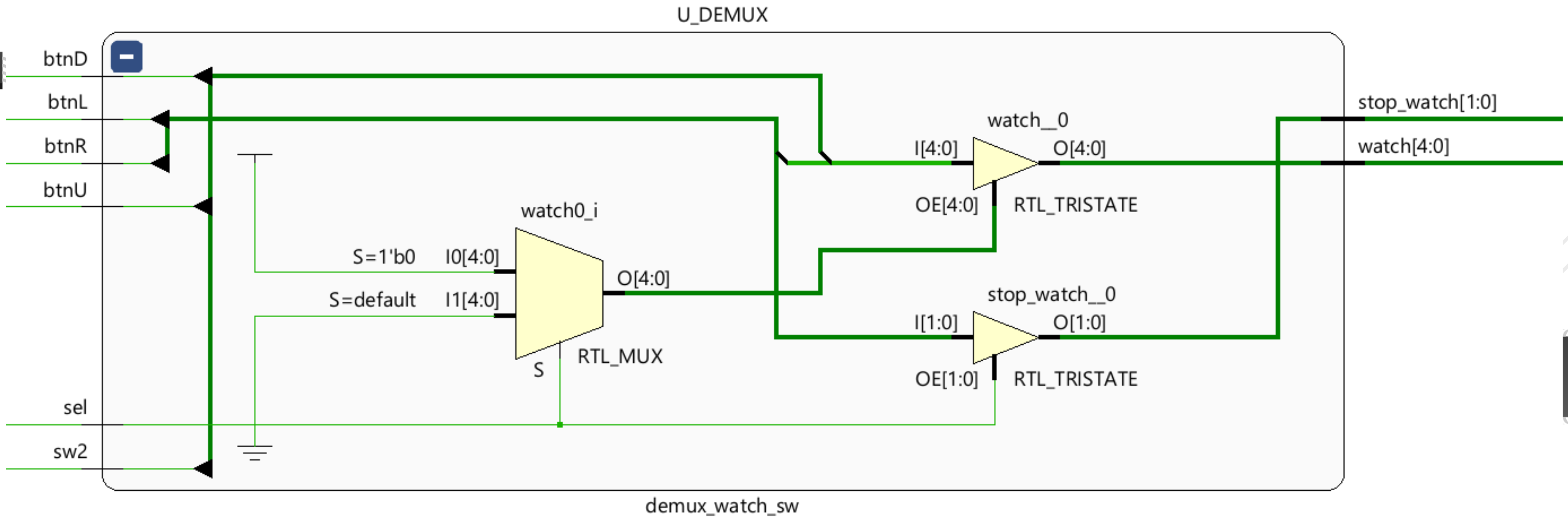

WATCH
BTN_DEB_U
BTN_DEB_D
BTN_DEB_L
BTN_DEB_R
WATCH_CU
WATCH_DP
5
BTN[3:0]
4
DEMUX
SW[2]
2
S_WATCH
BTN_R_ST
BTN_CLR
SW_CU
SW_DP
SW[1]

# DEMUX: CODE

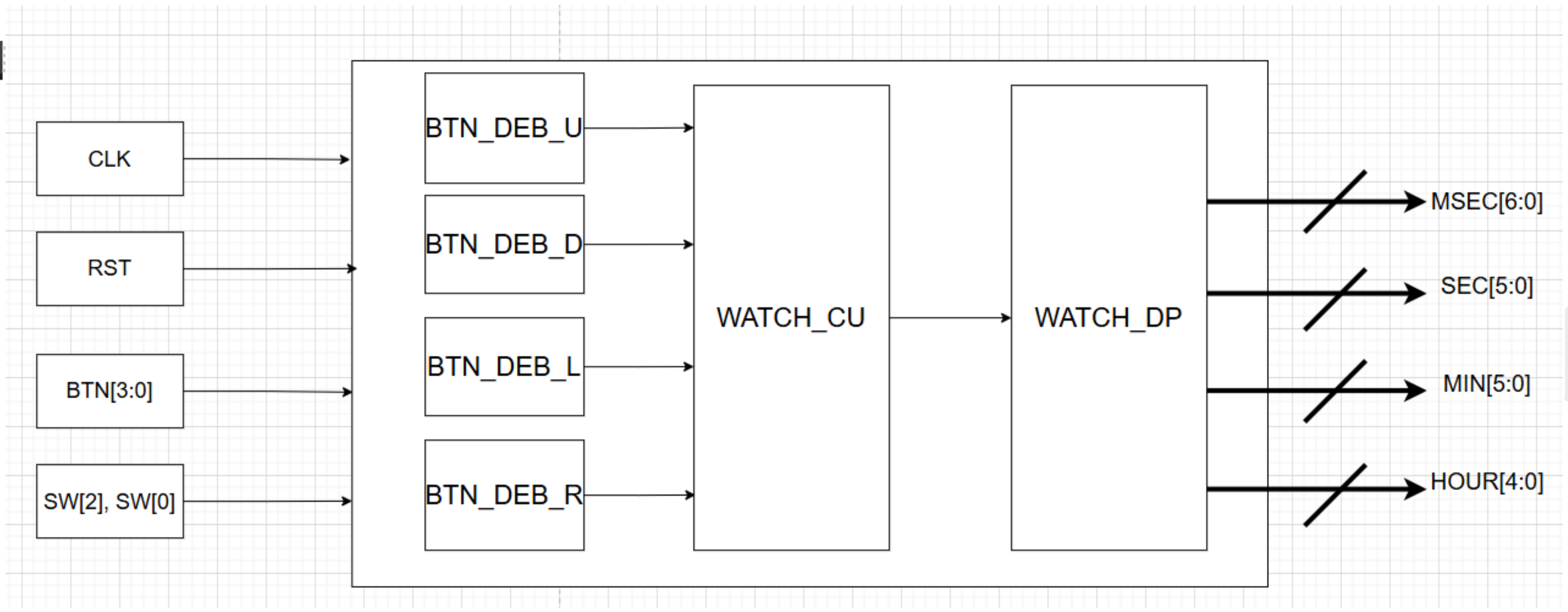


```
module demux_watch_sw (
    input btnU,
    input btnD,
    input btnL,
    input btnR,
    input sw2,
    input sel,

    output [4:0] watch,
    output [1:0] stop_watch
);

    assign stop_watch = (sel) ? {btnL, btnR} : 2'bzz;
    assign watch = (!sel) ? {btnU, btnD, btnL, btnR, sw2} : 5'bzz_zzz;

endmodule
```

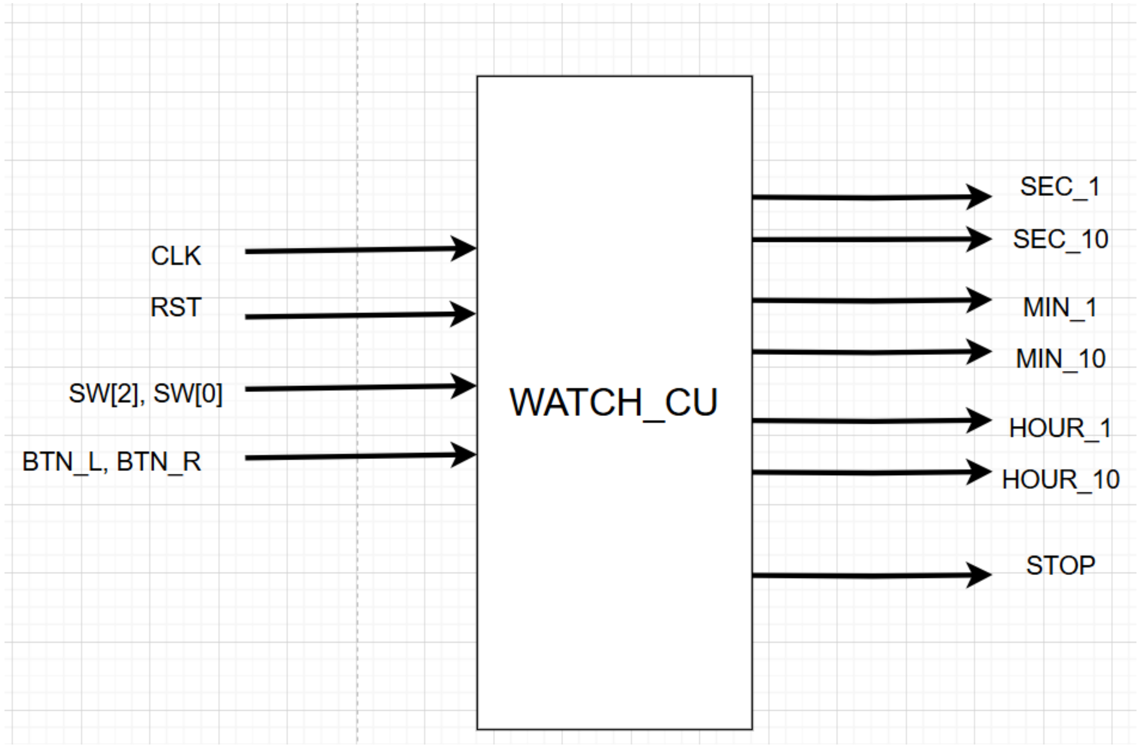
# DEMUX: Schematic

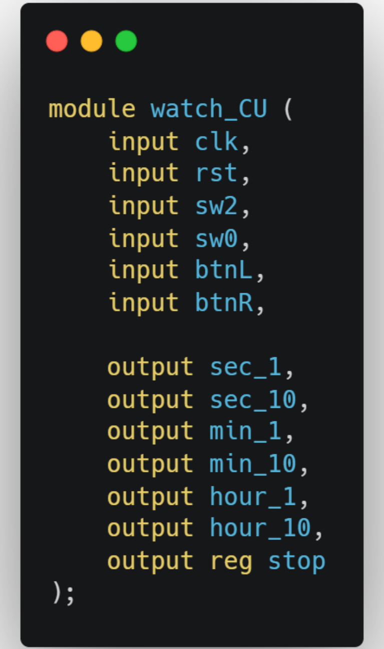


# Watch


CLK
RST
BTN[3:0]
SW[2], SW[0]
BTN_DEB_U
BTN_DEB_D
BTN_DEB_L
BTN_DEB_R
WATCH_CU
WATCH_DP
MSEC[6:0]
SEC[5:0]
MIN[5:0]
HOUR[4:0]

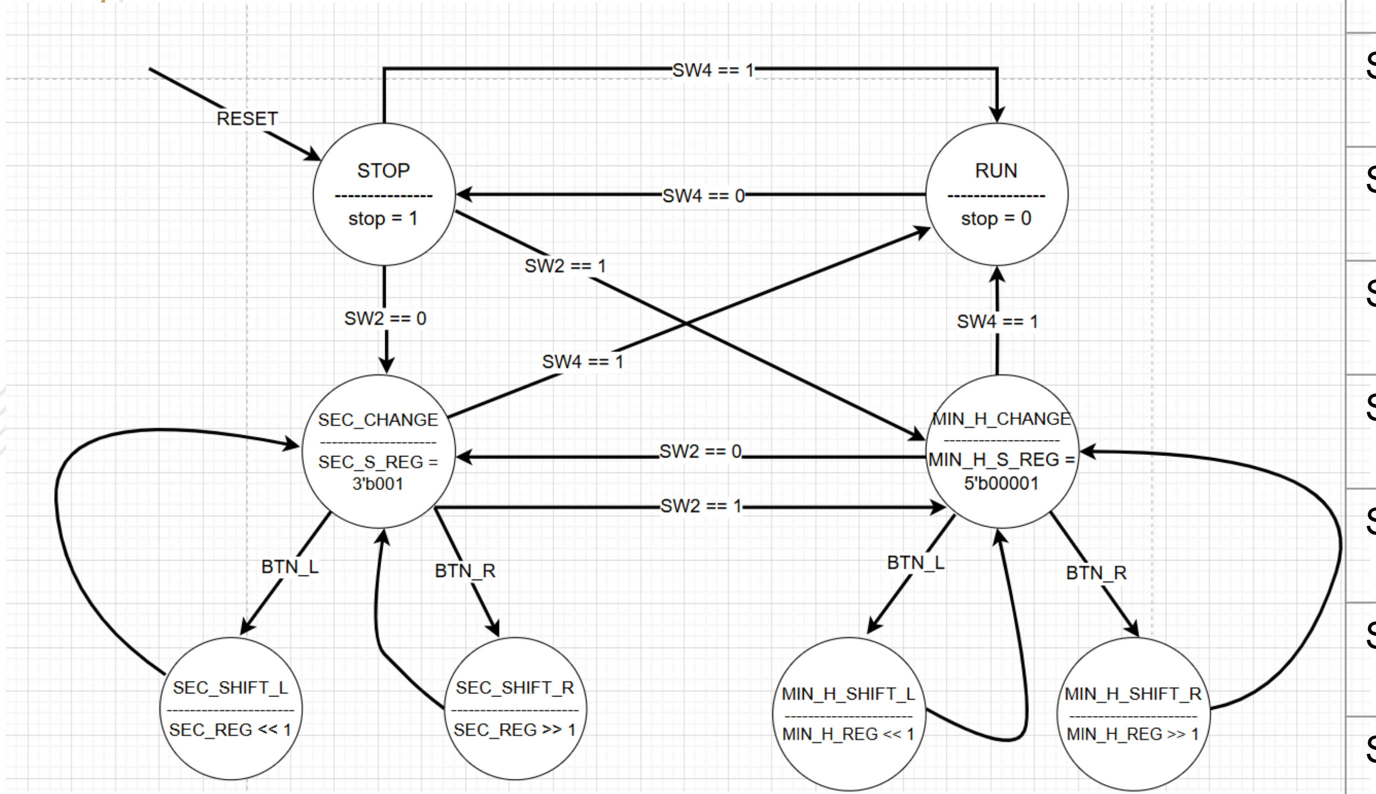
# Watch_CU: Block Diagram





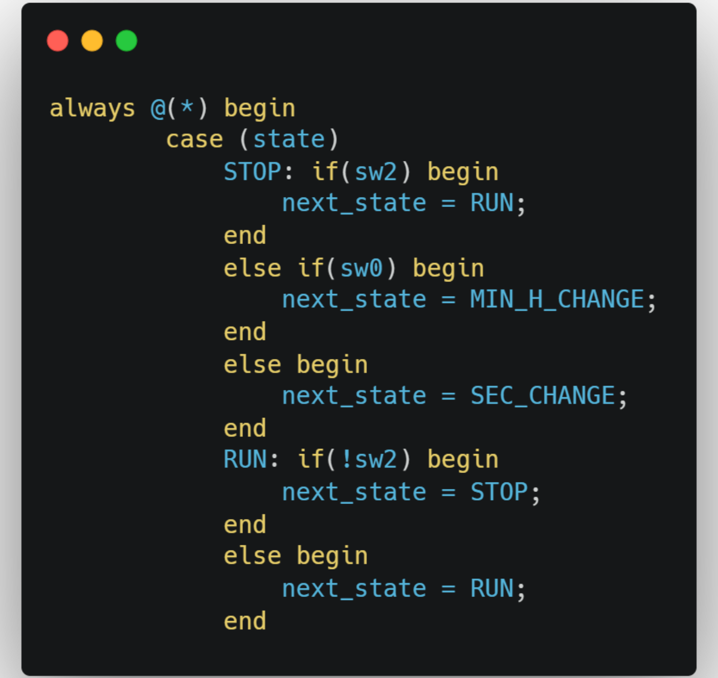
```
module watch_CU (
    input clk,
    input rst,
    input sw2,
    input sw0,
    input btnL,
    input btnR,

    output sec_1,
    output sec_10,
    output min_1,
    output min_10,
    output hour_1,
    output hour_10,
    output reg stop
);
```

# Watch CU: FSM



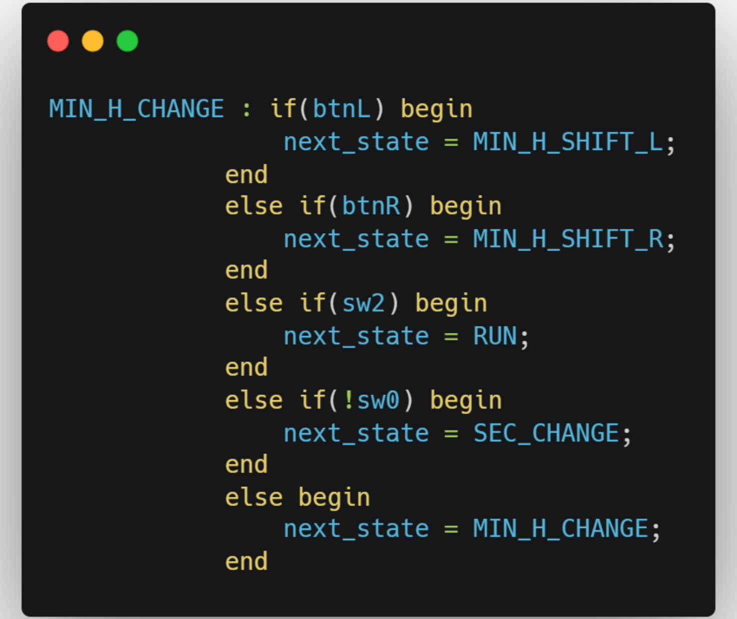
| | |
|---|---|
| State 0 | STOP |
| State 1 | RUN |
| State 2 | SEC_CHANGE |
| State 3 | MIN_H_CHANGE |
| State 4 | SEC_SHIFT_L |
| State 5 | SEC_SHIFT_R |
| State 6 | MIN_H_SHIFT_L |
| State 7 | MIN_H_SHIFT_R |

# Watch_CU: Next state 1

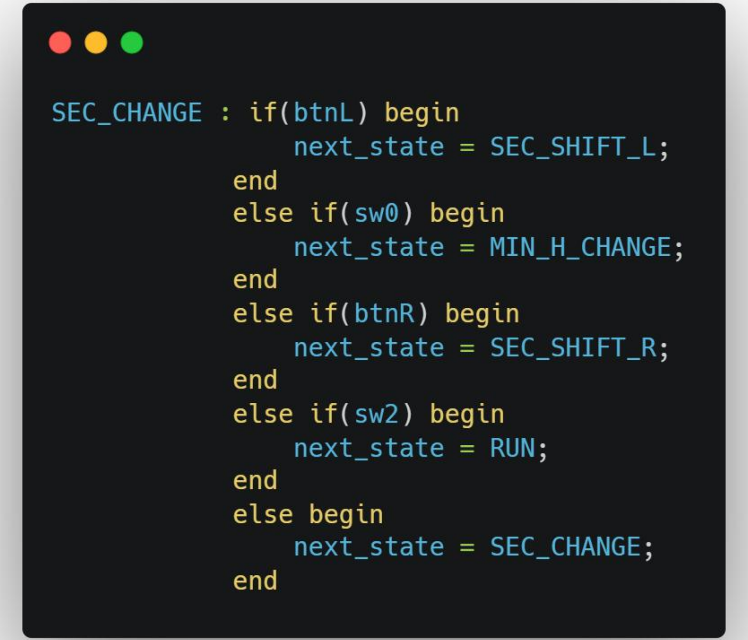


```
always @(*) begin
        case (state)
            STOP: if(sw2) begin
                next_state = RUN;
            end
            else if(sw0) begin
                next_state = MIN_H_CHANGE;
            end
            else begin
                next_state = SEC_CHANGE;
            end
            RUN: if(!sw2) begin
                next_state = STOP;
            end
            else begin
                next_state = RUN;
            end
```

# Watch_CU: Next state 2

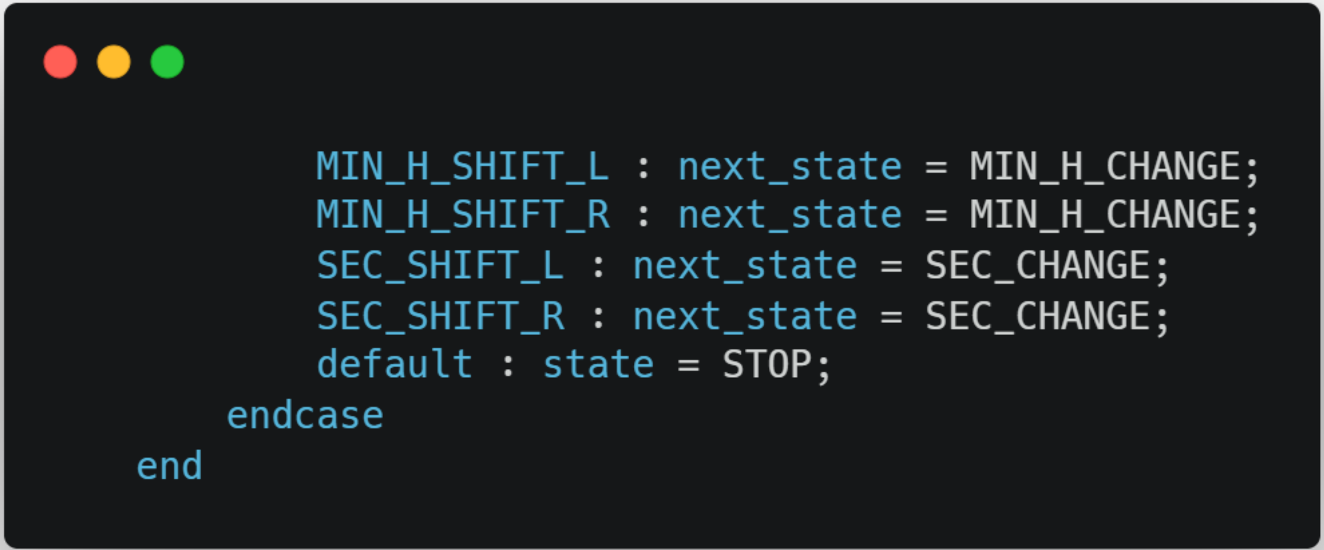


```
MIN_H_CHANGE : if(btnL) begin
                   next_state = MIN_H_SHIFT_L;
               end
               else if(btnR) begin
                   next_state = MIN_H_SHIFT_R;
               end
               else if(sw2) begin
                   next_state = RUN;
               end
               else if(!sw0) begin
                   next_state = SEC_CHANGE;
               end
               else begin
                   next_state = MIN_H_CHANGE;
               end
```



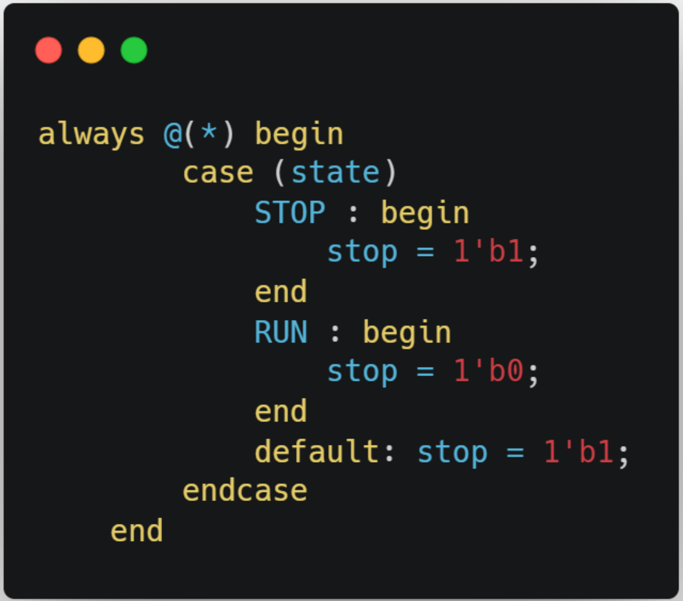
```
SEC_CHANGE : if(btnL) begin
                 next_state = SEC_SHIFT_L;
             end
             else if(sw0) begin
                 next_state = MIN_H_CHANGE;
             end
             else if(btnR) begin
                 next_state = SEC_SHIFT_R;
             end
             else if(sw2) begin
                 next_state = RUN;
             end
             else begin
                 next_state = SEC_CHANGE;
             end
```

# Watch_CU: Next state 3



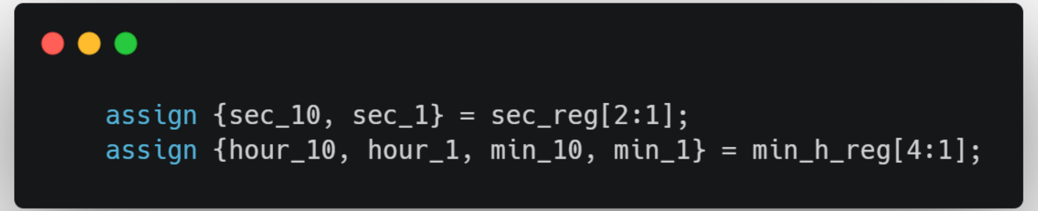
```
            MIN_H_SHIFT_L : next_state = MIN_H_CHANGE;
            MIN_H_SHIFT_R : next_state = MIN_H_CHANGE;
            SEC_SHIFT_L : next_state = SEC_CHANGE;
            SEC_SHIFT_R : next_state = SEC_CHANGE;
            default : state = STOP;
        endcase
    end
```

## Watch_CU: Output Logic 1

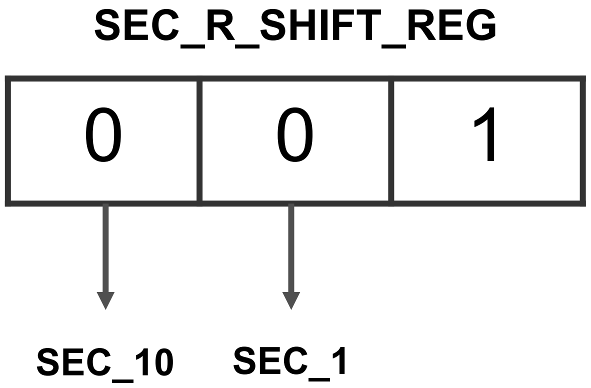


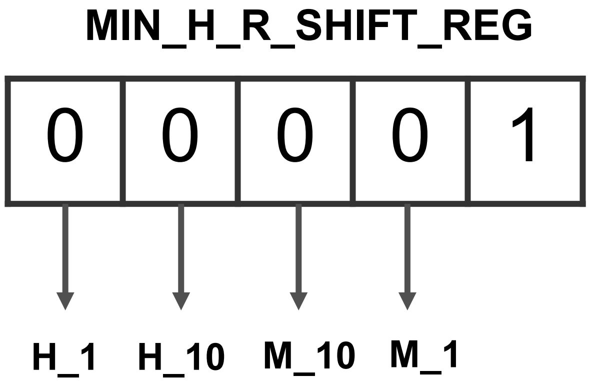
```
always @(*) begin
        case (state)
            STOP : begin
                stop = 1'b1;
            end
            RUN : begin
                stop = 1'b0;
            end
            default: stop = 1'b1;
        endcase
    end
```

# Watch_CU: Output Logic 2

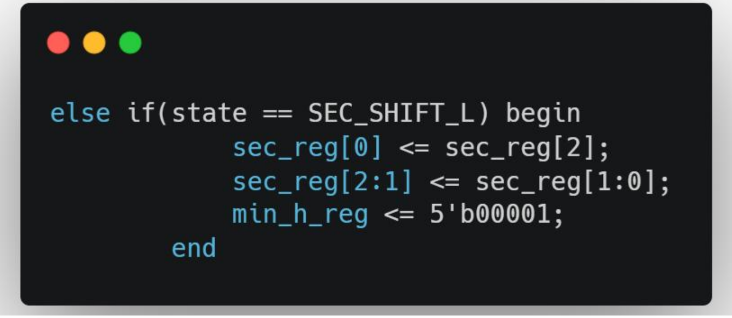


```
assign {sec_10, sec_1} = sec_reg[2:1];
assign {hour_10, hour_1, min_10, min_1} = min_h_reg[4:1];
```

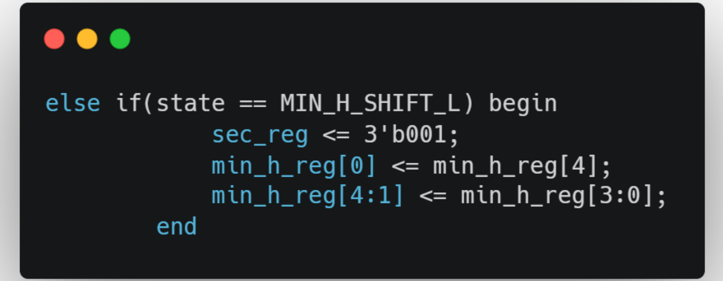




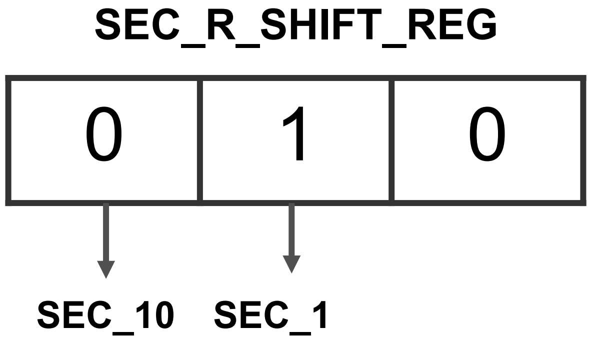
# Watch_CU: Output Logic 3

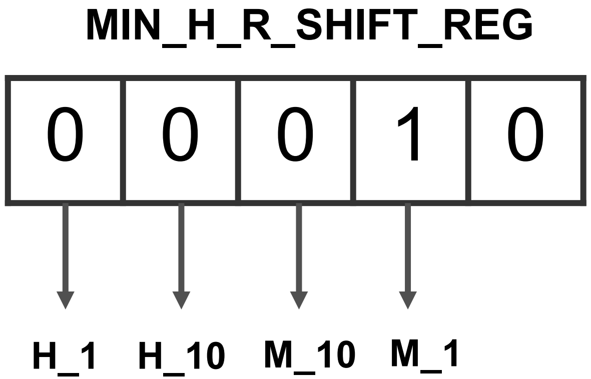


```
else if(state == SEC_SHIFT_L) begin
            sec_reg[0] <= sec_reg[2];
            sec_reg[2:1] <= sec_reg[1:0];
            min_h_reg <= 5'b00001;
        end
```

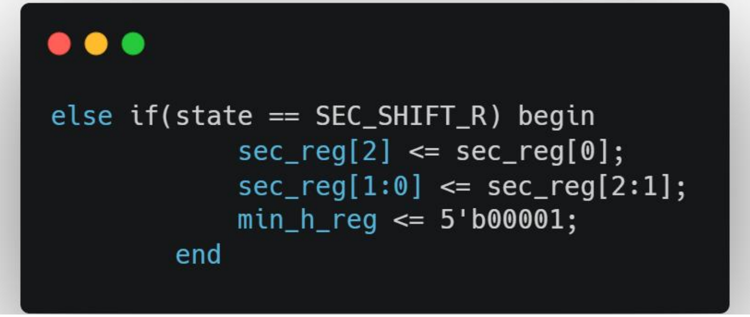


```
else if(state == MIN_H_SHIFT_L) begin
            sec_reg <= 3'b001;
            min_h_reg[0] <= min_h_reg[4];
            min_h_reg[4:1] <= min_h_reg[3:0];
        end
```

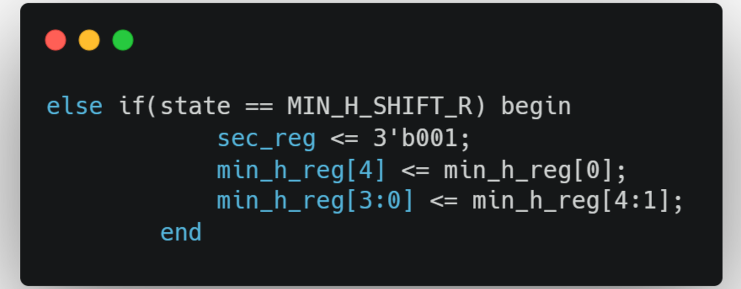




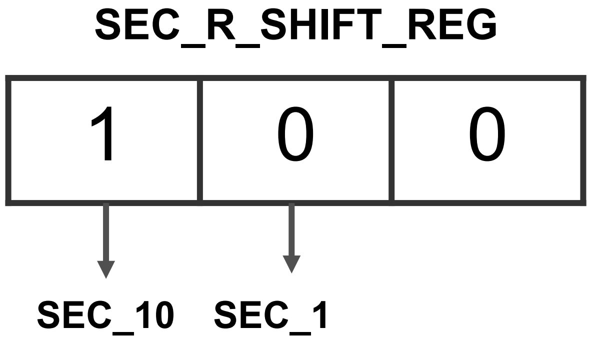
# Watch CU: Output Logic 4

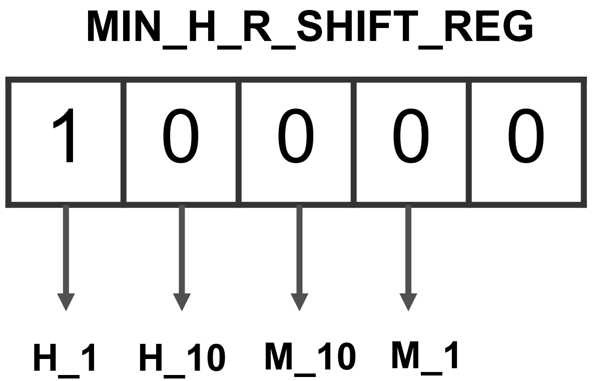


```
else if(state == SEC_SHIFT_R) begin
            sec_reg[2] <= sec_reg[0];
            sec_reg[1:0] <= sec_reg[2:1];
            min_h_reg <= 5'b00001;
        end
```

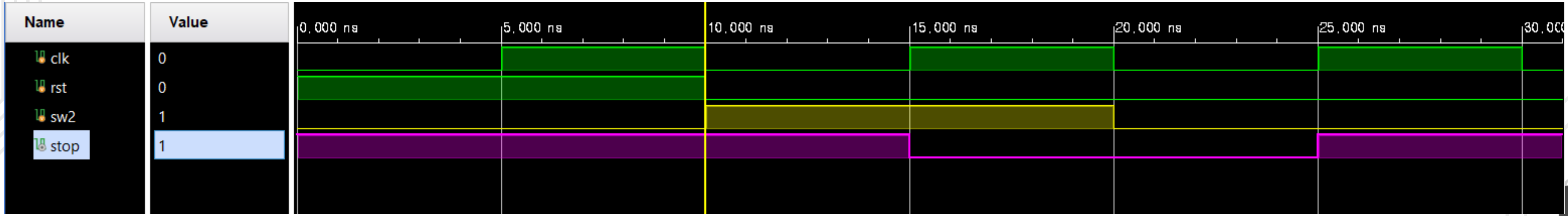


```
else if(state == MIN_H_SHIFT_R) begin
            sec_reg <= 3'b001;
            min_h_reg[4] <= min_h_reg[0];
            min_h_reg[3:0] <= min_h_reg[4:1];
        end
```



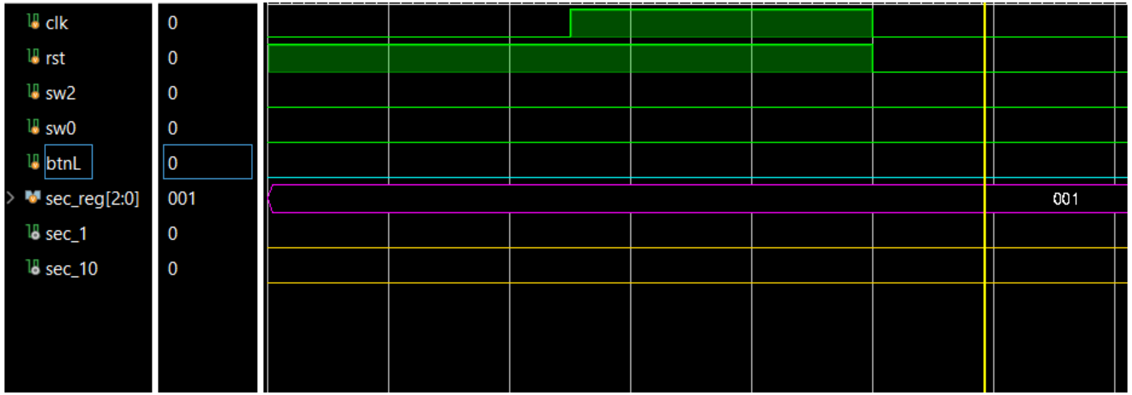


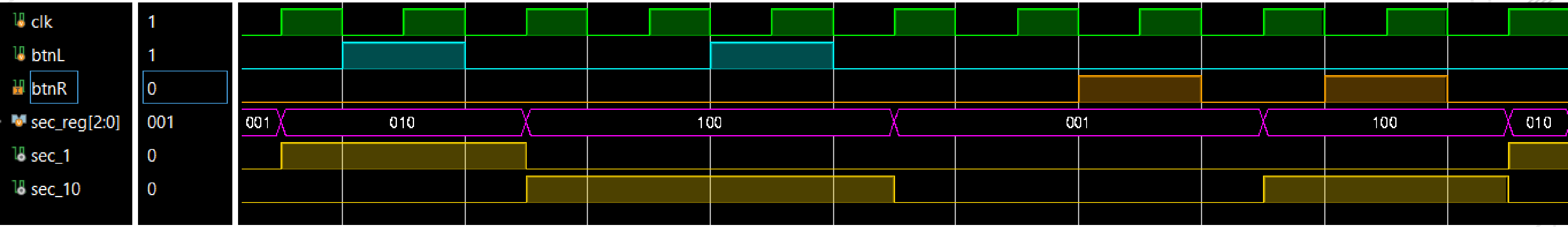
# Watch_CU: Simulation(STOP)



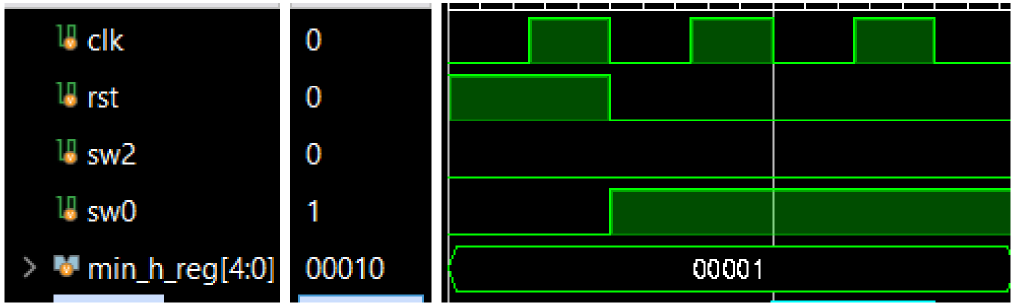
| Name | Value |
|---|---|
| clk | 0 |
| rst | 0 |
| sw2 | 1 |
| stop | 1 |

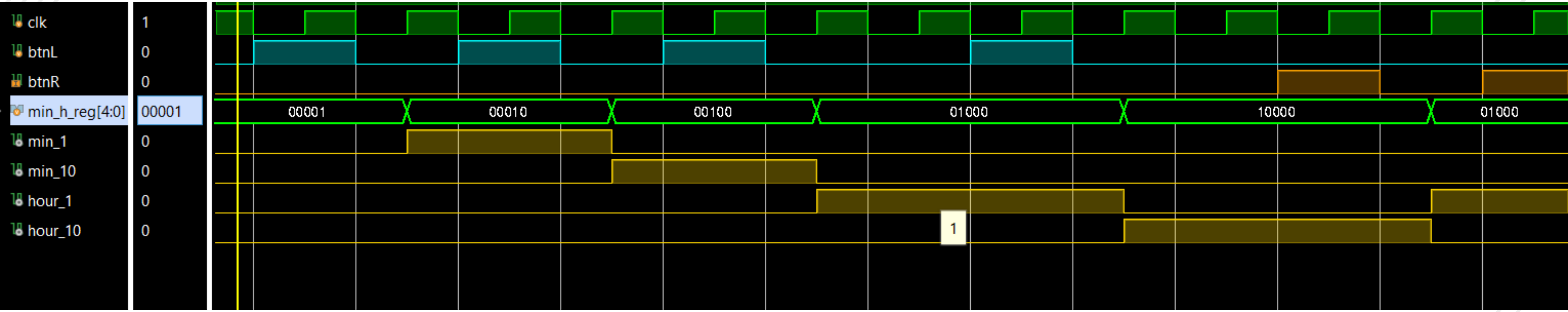
# Watch_CU: Simulation(Shift: Sec)



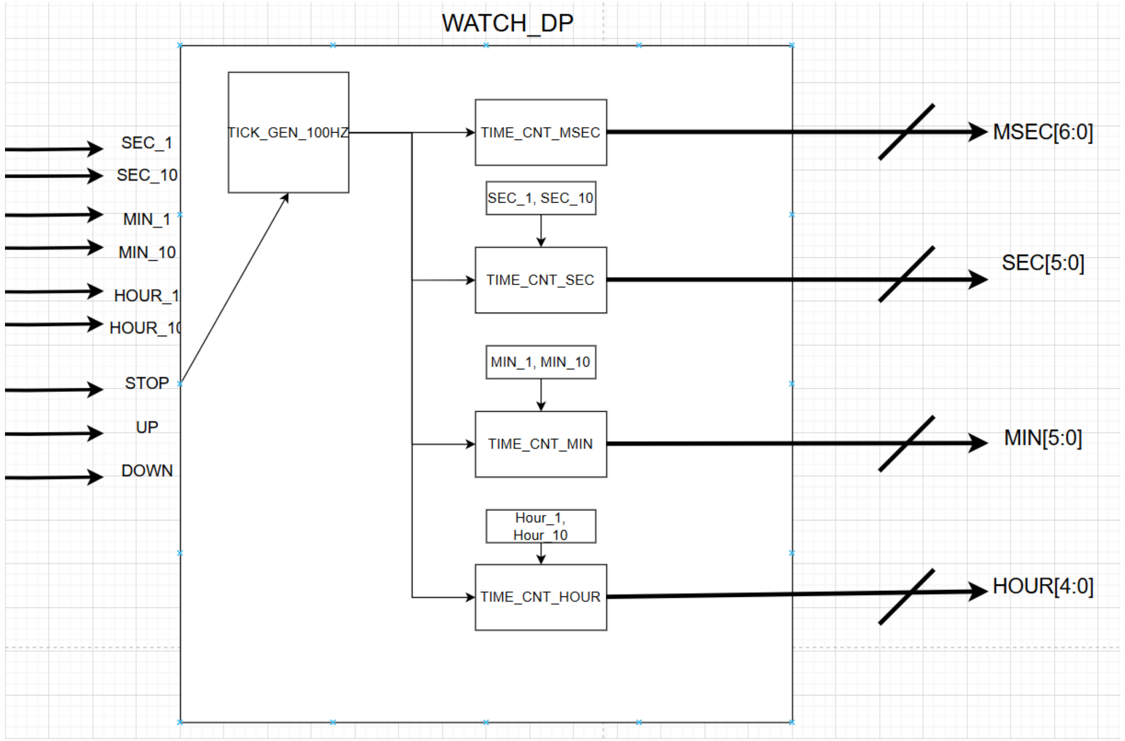


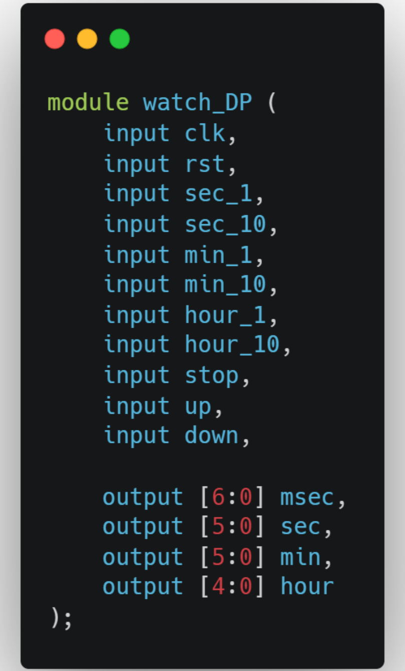
# Watch_CU: Simulation(Shift: Min_H)





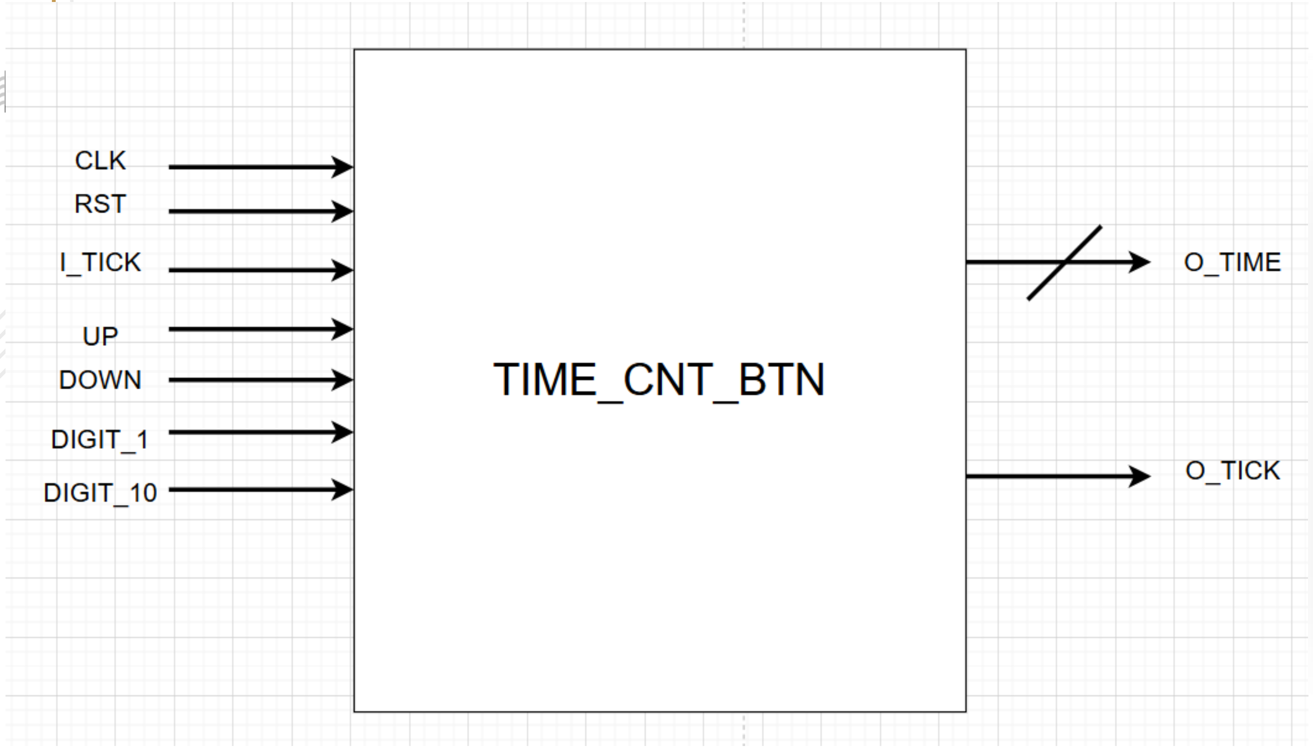
# Watch_DP: Block Diagram

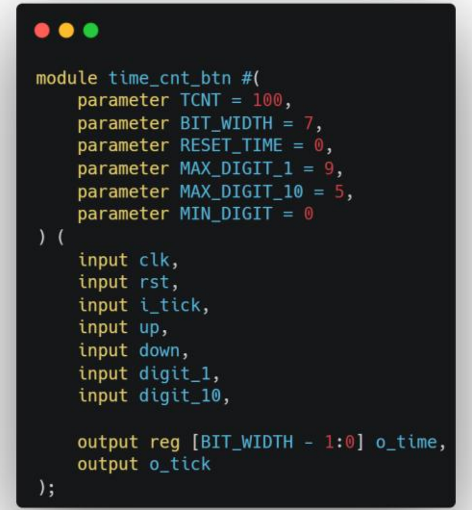




```
module watch_DP (
    input clk,
    input rst,
    input sec_1,
    input sec_10,
    input min_1,
    input min_10,
    input hour_1,
    input hour_10,
    input stop,
    input up,
    input down,

    output [6:0] msec,
    output [5:0] sec,
    output [5:0] min,
    output [4:0] hour
);
```

# TIME_CNT: Block Diagram





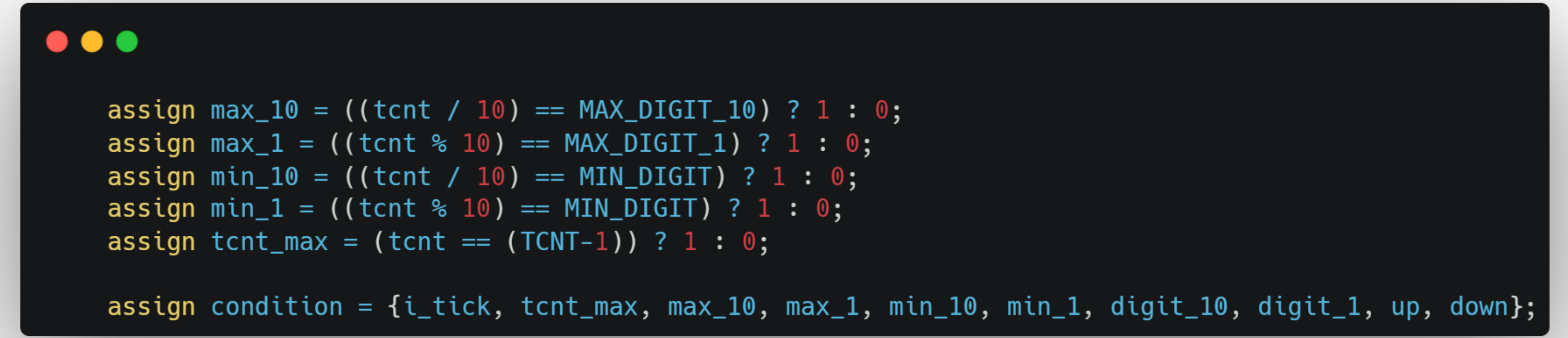
```
module time_cnt_btn #(
    parameter TCNT = 100,
    parameter BIT_WIDTH = 7,
    parameter RESET_TIME = 0,
    parameter MAX_DIGIT_1 = 9,
    parameter MAX_DIGIT_10 = 5,
    parameter MIN_DIGIT = 0
) (
    input clk,
    input rst,
    input i_tick,
    input up,
    input down,
    input digit_1,
    input digit_10,

    output reg [BIT_WIDTH - 1:0] o_time,
    output o_tick
);
```

## TIME_CNT: Problem

| Condition |
|---|
| Digit_10 == MAX |
| Digit_1 == MAX |
| Digit_10 == MIN |
| Digit_1 == MIN |
| TCNT == MAX |
| ETC |

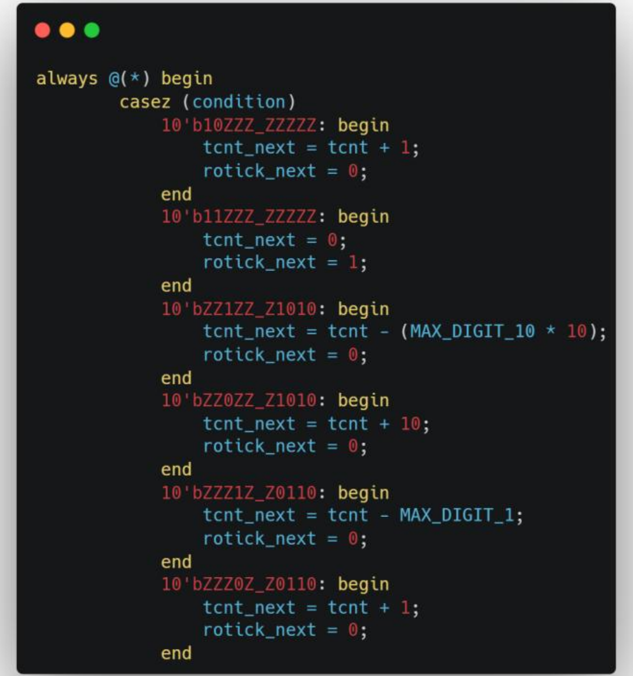
$\Rightarrow$ **Too Many MUX!**

## TIME_CNT: Solution – Flag(CASE)

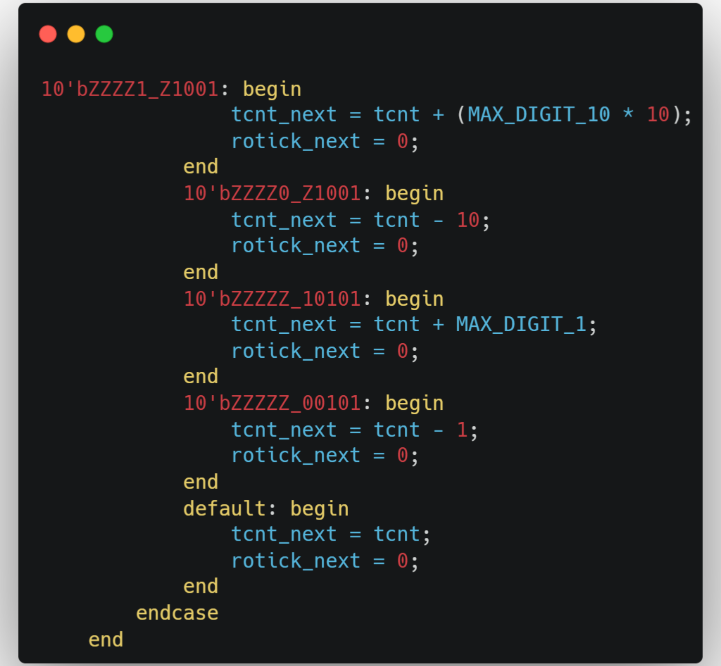


```
assign max_10 = ((tcnt / 10) == MAX_DIGIT_10) ? 1 : 0;
assign max_1 = ((tcnt % 10) == MAX_DIGIT_1) ? 1 : 0;
assign min_10 = ((tcnt / 10) == MIN_DIGIT) ? 1 : 0;
assign min_1 = ((tcnt % 10) == MIN_DIGIT) ? 1 : 0;
assign tcnt_max = (tcnt == (TCNT-1)) ? 1 : 0;

assign condition = {i_tick, tcnt_max, max_10, max_1, min_10, min_1, digit_10, digit_1, up, down};
```

# TIME_CNT: Solution – Flag(CASE)

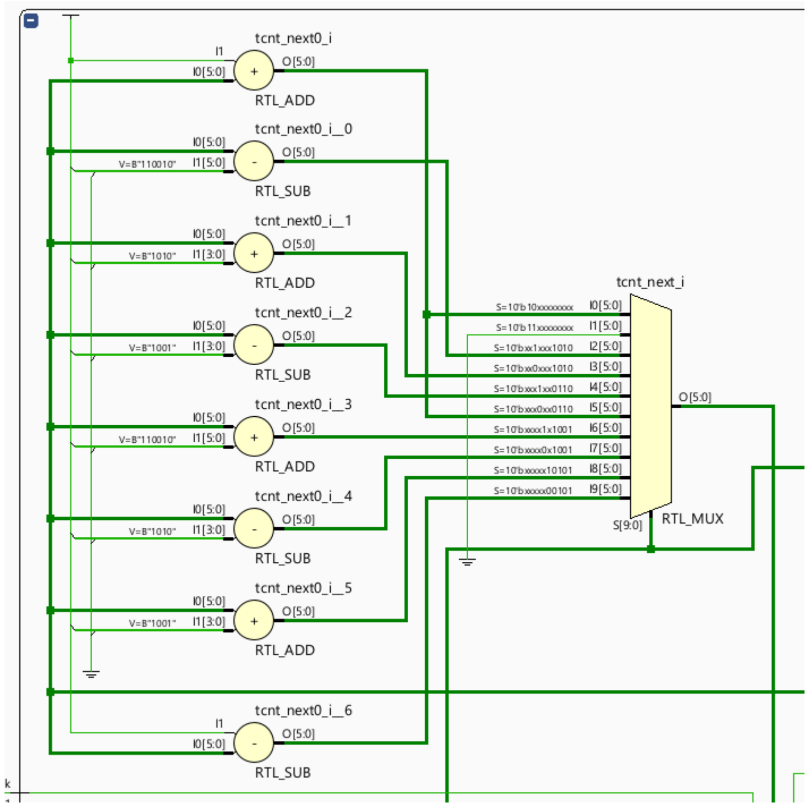


```
always @(*) begin
        casez (condition)
            10'b10ZZZ_ZZZZZ: begin
                tcnt_next = tcnt + 1;
                rotick_next = 0;
            end
            10'b11ZZZ_ZZZZZ: begin
                tcnt_next = 0;
                rotick_next = 1;
            end
            10'bZZ1ZZ_Z1010: begin
                tcnt_next = tcnt - (MAX_DIGIT_10 * 10);
                rotick_next = 0;
            end
            10'bZZ0ZZ_Z1010: begin
                tcnt_next = tcnt + 10;
                rotick_next = 0;
            end
            10'bZZZ1Z_Z0110: begin
                tcnt_next = tcnt - MAX_DIGIT_1;
                rotick_next = 0;
            end
            10'bZZZ0Z_Z0110: begin
                tcnt_next = tcnt + 1;
                rotick_next = 0;
            end
```

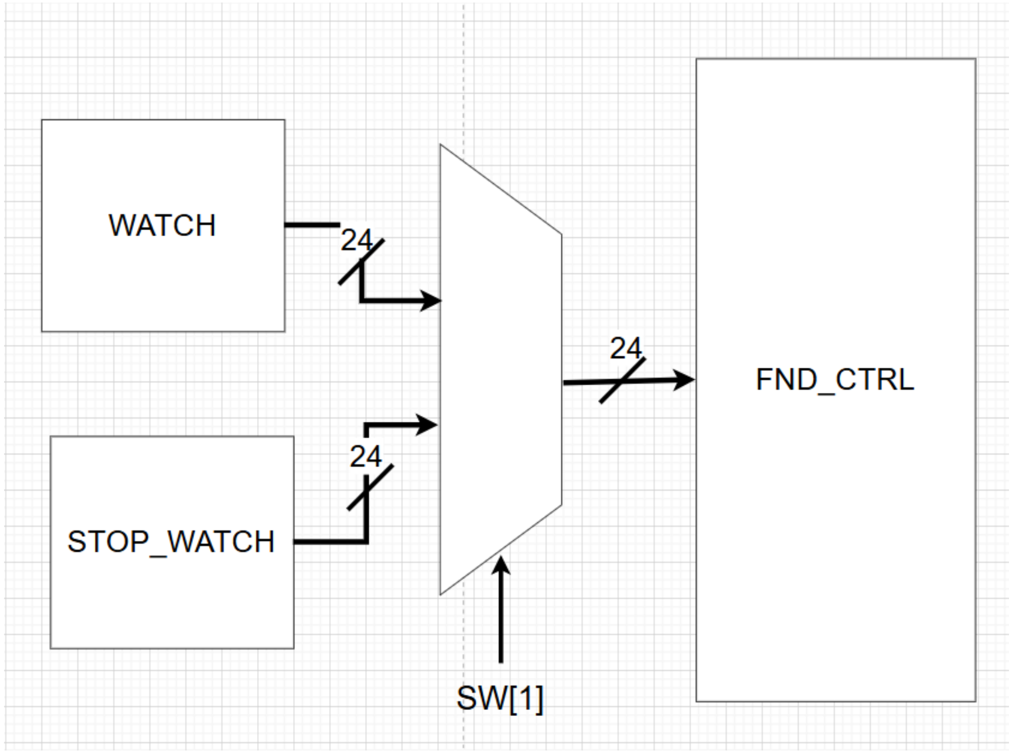


```
            10'bZZZZ1_Z1001: begin
                tcnt_next = tcnt + (MAX_DIGIT_10 * 10);
                rotick_next = 0;
            end
            10'bZZZZ0_Z1001: begin
                tcnt_next = tcnt - 10;
                rotick_next = 0;
            end
            10'bZZZZZ_10101: begin
                tcnt_next = tcnt + MAX_DIGIT_1;
                rotick_next = 0;
            end
            10'bZZZZZ_00101: begin
                tcnt_next = tcnt - 1;
                rotick_next = 0;
            end
            default: begin
                tcnt_next = tcnt;
                rotick_next = 0;
            end
        endcase
    end
```

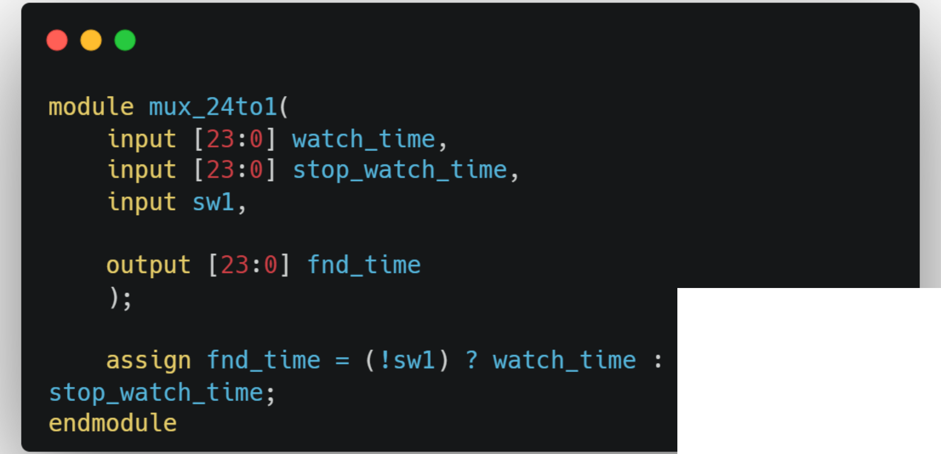
# TIME_CNT: Solution – Schematic



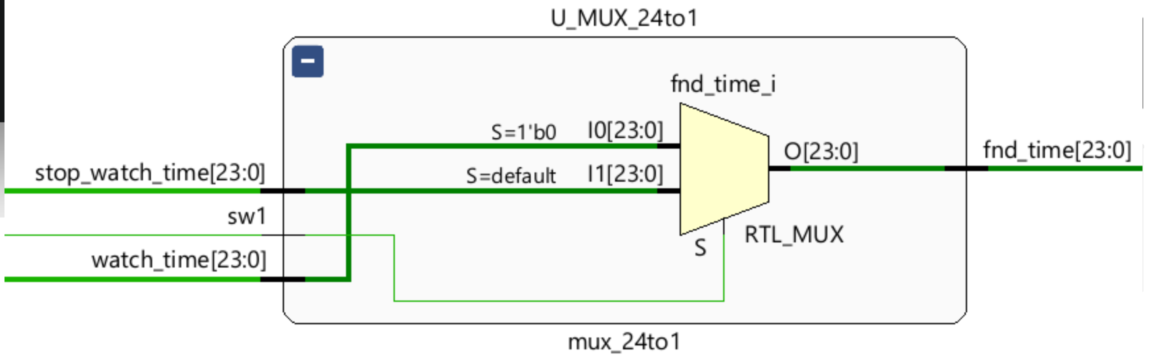
# 2X1 MUX



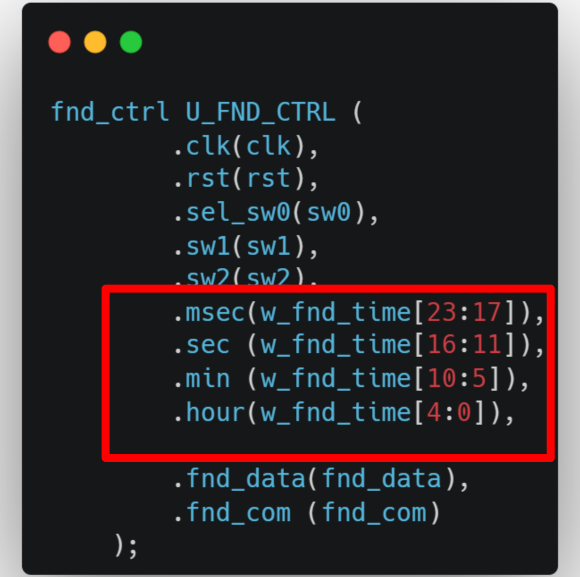
# 2X1 MUX: CODE & Result



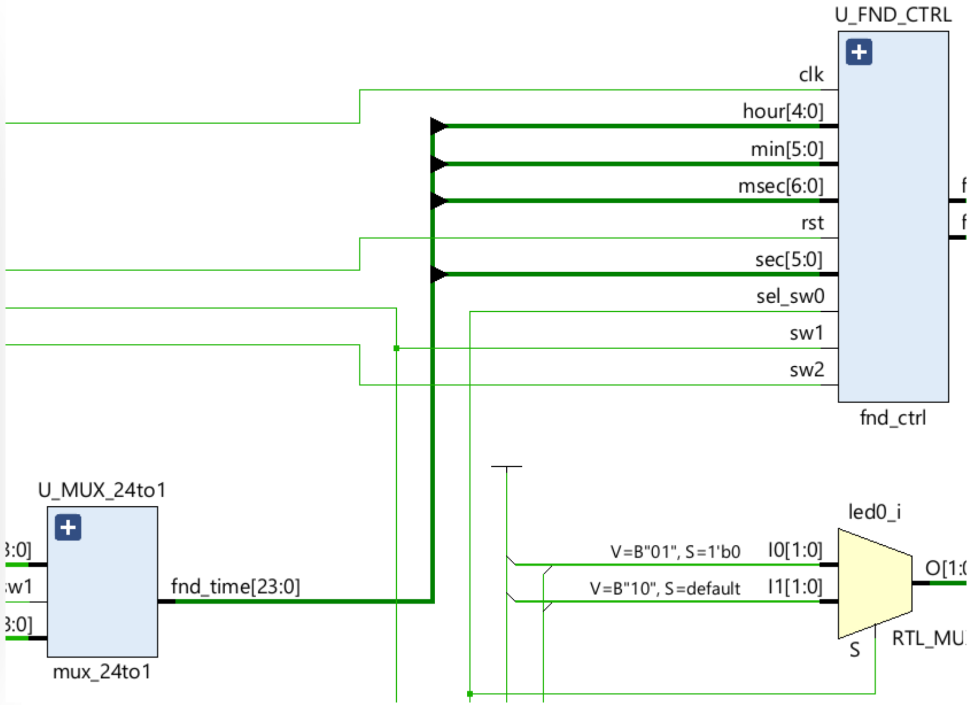
```
module mux_24to1(
    input [23:0] watch_time,
    input [23:0] stop_watch_time,
    input sw1,

    output [23:0] fnd_time
    );

    assign fnd_time = (!sw1) ? watch_time :
stop_watch_time;
endmodule
```



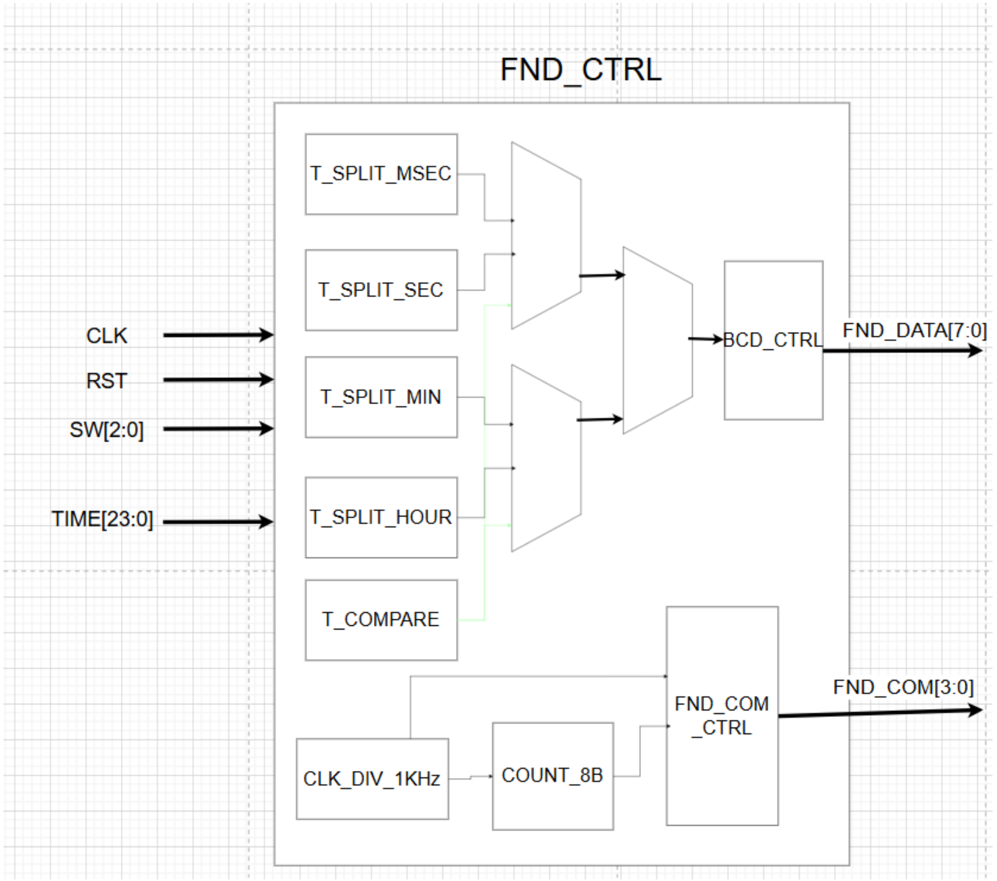
# 2X1 MUX → FND_CTRL



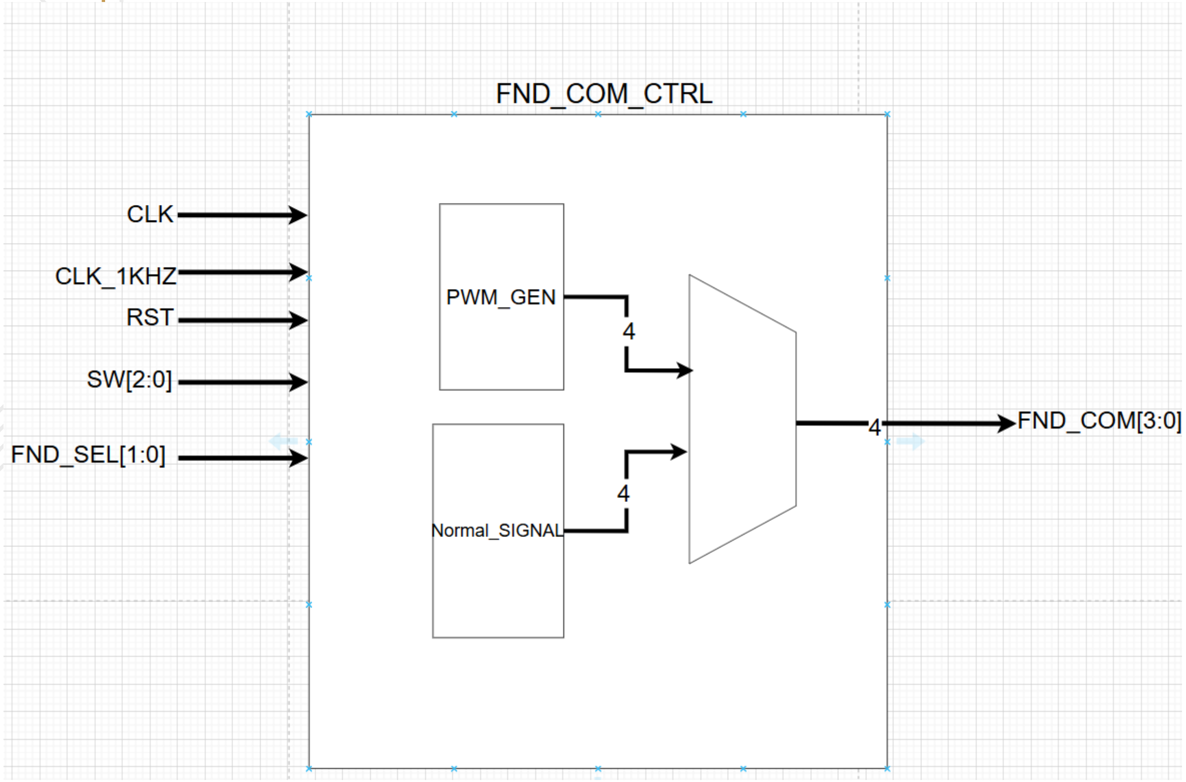
```
fnd_ctrl U_FND_CTRL (
        .clk(clk),
        .rst(rst),
        .sel_sw0(sw0),
        .sw1(sw1),
        .sw2(sw2),
        .msec(w_fnd_time[23:17]),
        .sec (w_fnd_time[16:11]),
        .min (w_fnd_time[10:5]),
        .hour(w_fnd_time[4:0]),

        .fnd_data(fnd_data),
        .fnd_com (fnd_com)
    );
```

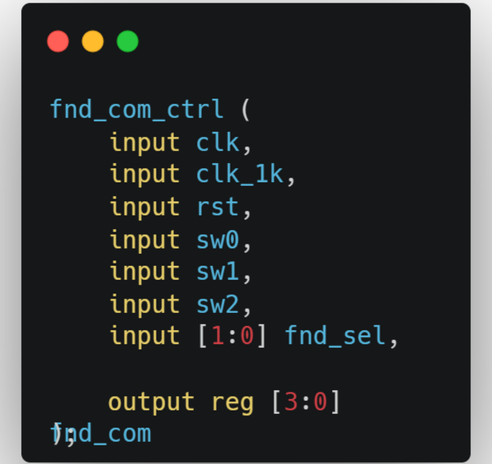


# FND_CTRL


FND_CTRL
T_SPLIT_MSEC
T_SPLIT_SEC
T_SPLIT_MIN
T_SPLIT_HOUR
T_COMPARE
CLK
RST
SW[2:0]
TIME[23:0]
BCD_CTRL
FND_DATA[7:0]
FND_COM
_CTRL
FND_COM[3:0]
CLK_DIV_1KHz
COUNT_8B

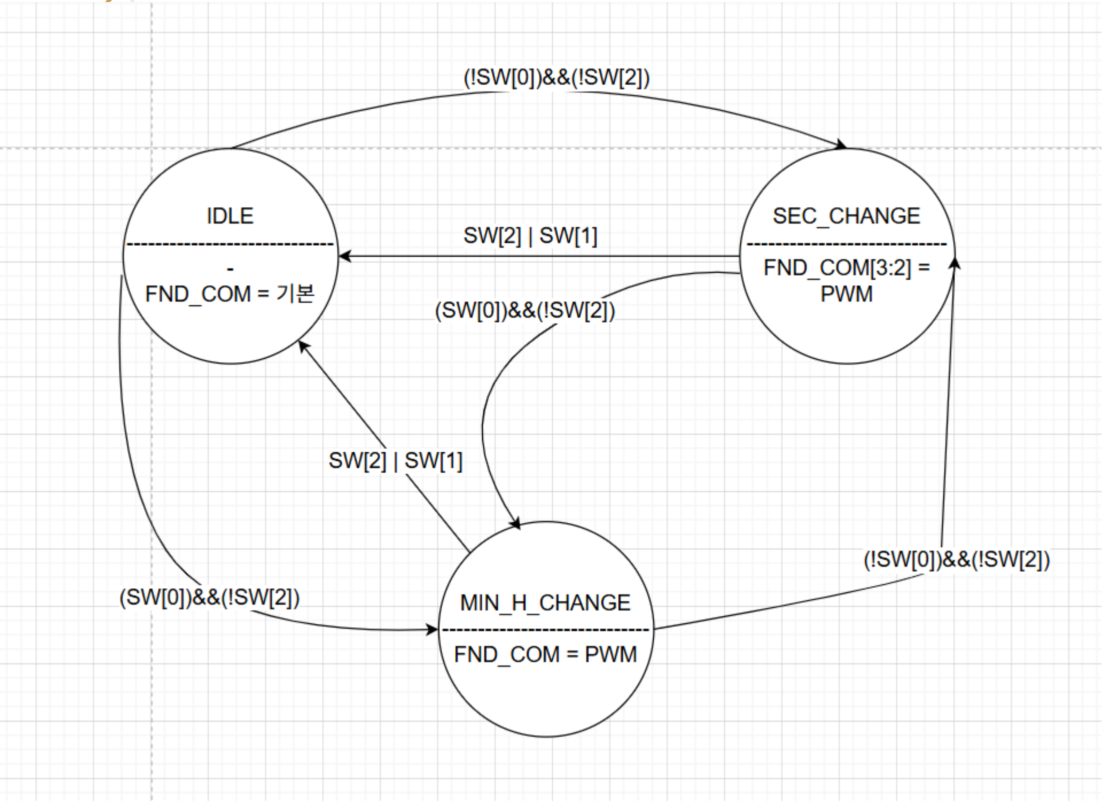
# FND_COM_CTRL





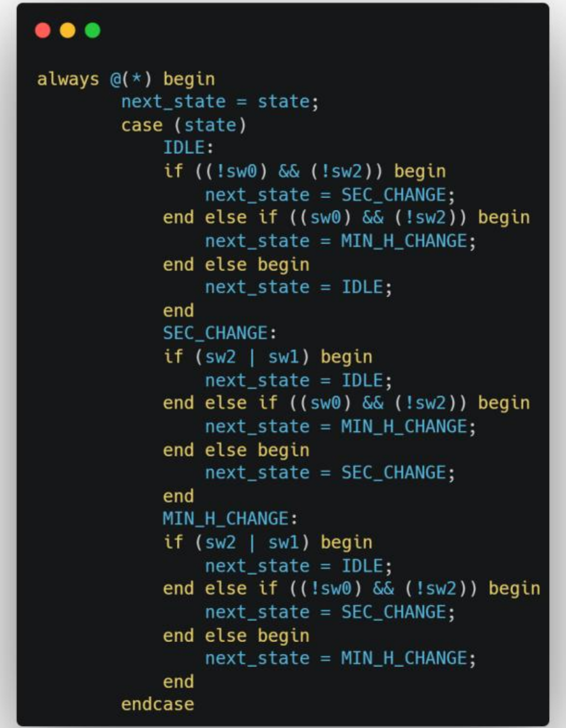
```
fnd_com_ctrl (
    input clk,
    input clk_1k,
    input rst,
    input sw0,
    input sw1,
    input sw2,
    input [1:0] fnd_sel,

    output reg [3:0] fnd_com
);
```

# FND_COM_CTRL: FSM



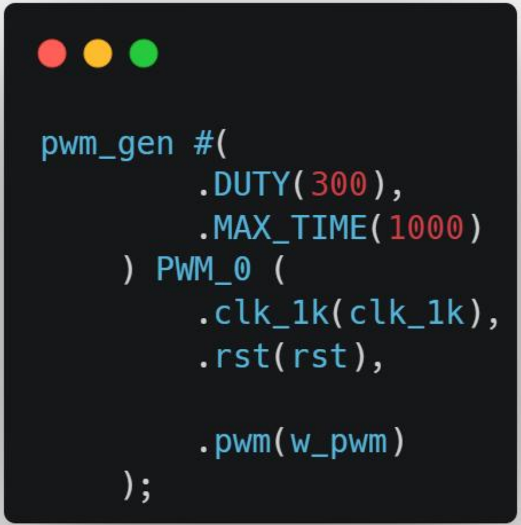
| | |
|---|---|
| STATE 0 | IDLE |
| STATE 1 | SEC_CHANGE |
| STATE_2 | MIN_H_CHANGE |

# FND_COM_CTRL: Next State Logic



```
always @(*) begin
        next_state = state;
        case (state)
            IDLE:
            if ((!sw0) && (!sw2)) begin
                next_state = SEC_CHANGE;
            end else if ((sw0) && (!sw2)) begin
                next_state = MIN_H_CHANGE;
            end else begin
                next_state = IDLE;
            end
            SEC_CHANGE:
            if (sw2 | sw1) begin
                next_state = IDLE;
            end else if ((sw0) && (!sw2)) begin
                next_state = MIN_H_CHANGE;
            end else begin
                next_state = SEC_CHANGE;
            end
            MIN_H_CHANGE:
            if (sw2 | sw1) begin
                next_state = IDLE;
            end else if ((!sw0) && (!sw2)) begin
                next_state = SEC_CHANGE;
            end else begin
                next_state = MIN_H_CHANGE;
            end
        endcase
```

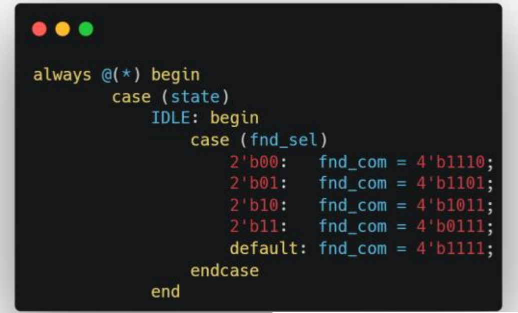
## FND_COM_CTRL: Ouput Logic



```
pwm_gen #(
        .DUTY(300),
        .MAX_TIME(1000)
    ) PWM_0 (
        .clk_1k(clk_1k),
        .rst(rst),

        .pwm(w_pwm)
    );
```

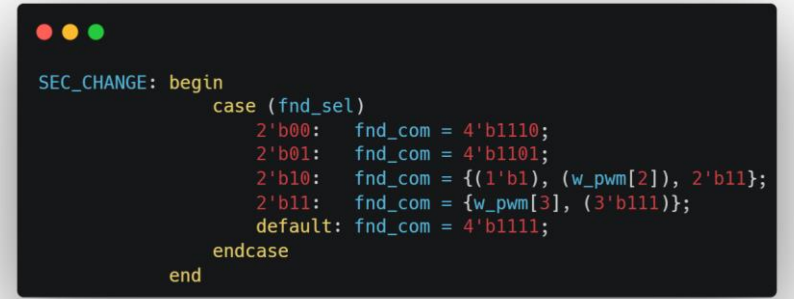
# FND_COM_CTRL: Ouput Logic

## IDLE



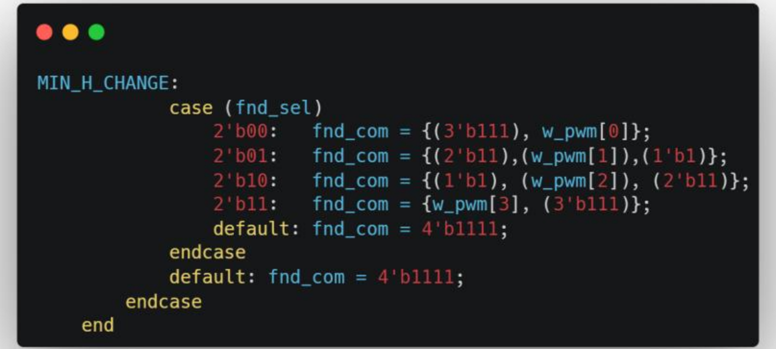
```
always @(*) begin
        case (state)
            IDLE: begin
                case (fnd_sel)
                    2'b00:   fnd_com = 4'b1110;
                    2'b01:   fnd_com = 4'b1101;
                    2'b10:   fnd_com = 4'b1011;
                    2'b11:   fnd_com = 4'b0111;
                    default: fnd_com = 4'b1111;
                endcase
            end
```

## SEC_CHANGE



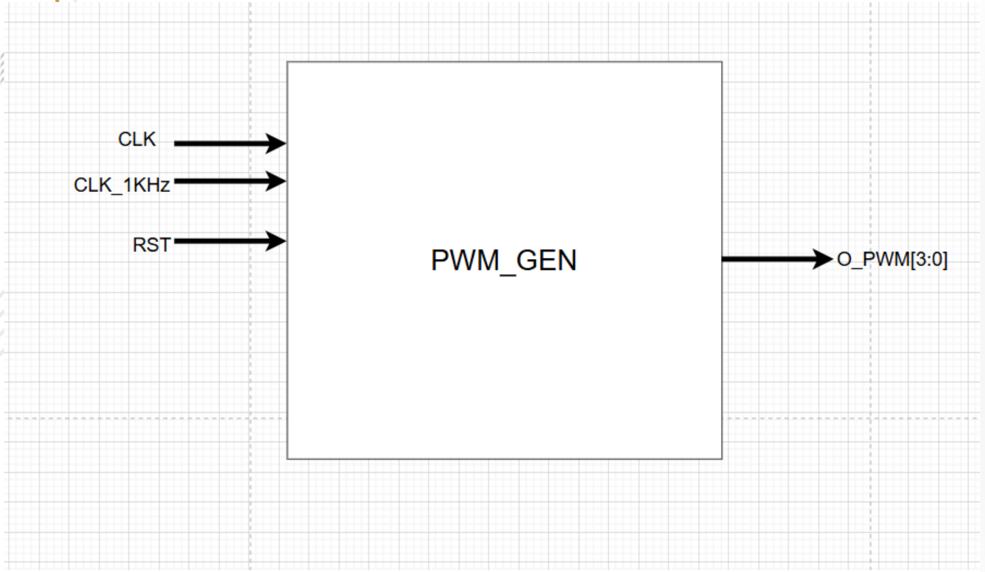
```
SEC_CHANGE: begin
                case (fnd_sel)
                    2'b00:    fnd_com = 4'b1110;
                    2'b01:    fnd_com = 4'b1101;
                    2'b10:    fnd_com = {(1'b1), (w_pwm[2]), 2'b11};
                    2'b11:    fnd_com = {w_pwm[3], (3'b111)};
                    default: fnd_com = 4'b1111;
                endcase
            end
```

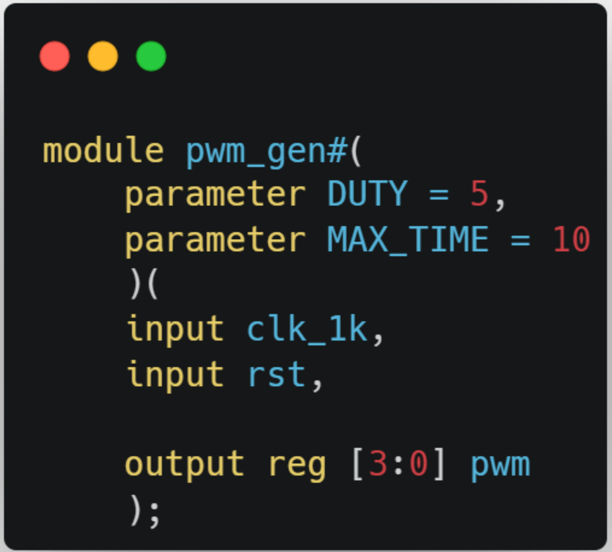
## MIN_H_CHANGE



```
MIN_H_CHANGE:
            case (fnd_sel)
                2'b00:   fnd_com = {(3'b111), w_pwm[0]};
                2'b01:   fnd_com = {(2'b11),(w_pwm[1]),(1'b1)};
                2'b10:   fnd_com = {(1'b1), (w_pwm[2]), (2'b11)};
                2'b11:   fnd_com = {w_pwm[3], (3'b111)};
                default: fnd_com = 4'b1111;
            endcase
            default: fnd_com = 4'b1111;
        endcase
    end
```

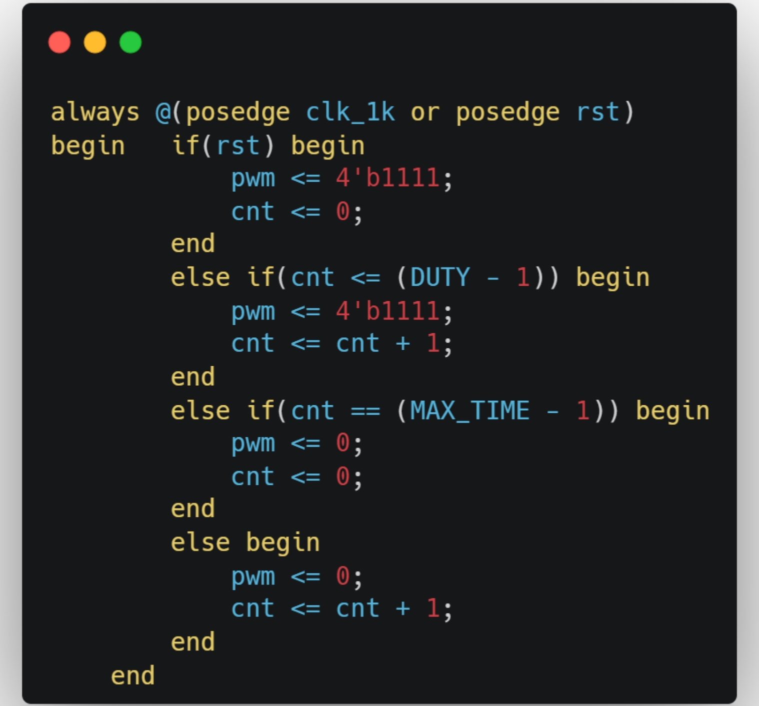
## PWM





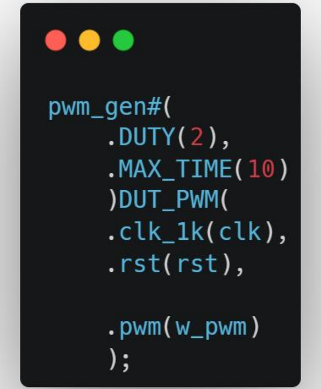
```
module pwm_gen#(
    parameter DUTY = 5,
    parameter MAX_TIME = 10
    )(
    input clk_1k,
    input rst,

    output reg [3:0] pwm
    );
```

# PWM: Code

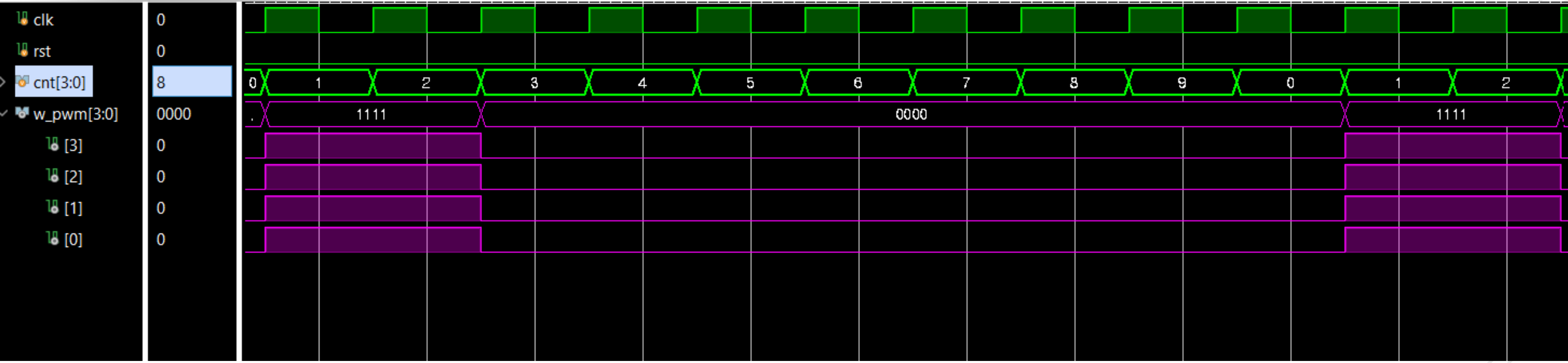


```
always @(posedge clk_1k or posedge rst)
begin   if(rst) begin
            pwm <= 4'b1111;
            cnt <= 0;
        end
        else if(cnt <= (DUTY - 1)) begin
            pwm <= 4'b1111;
            cnt <= cnt + 1;
        end
        else if(cnt == (MAX_TIME - 1)) begin
            pwm <= 0;
            cnt <= 0;
        end
        else begin
            pwm <= 0;
            cnt <= cnt + 1;
        end
    end
```

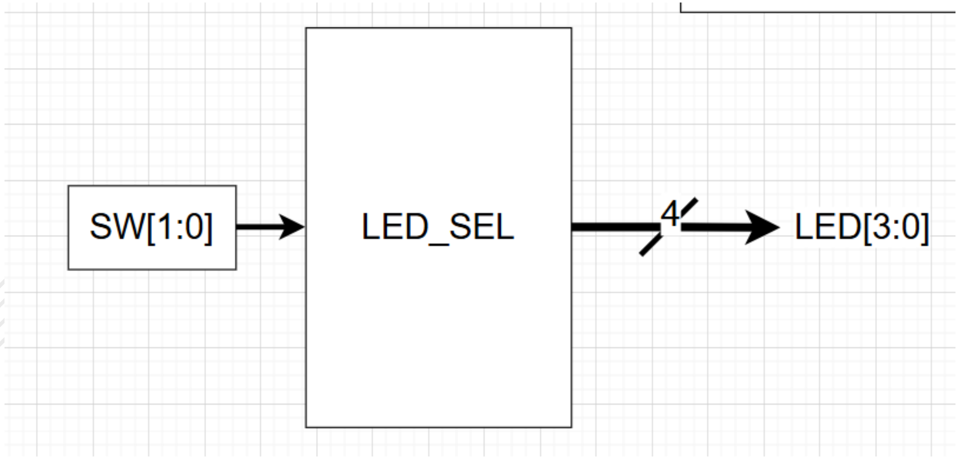
# PWM: Simulation



```
pwm_gen#(
    .DUTY(2),
    .MAX_TIME(10)
    )DUT_PWM(
    .clk_1k(clk),
    .rst(rst),

    .pwm(w_pwm)
    );
```



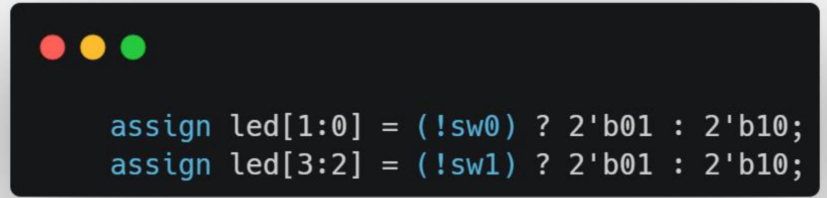
# LED_SEL



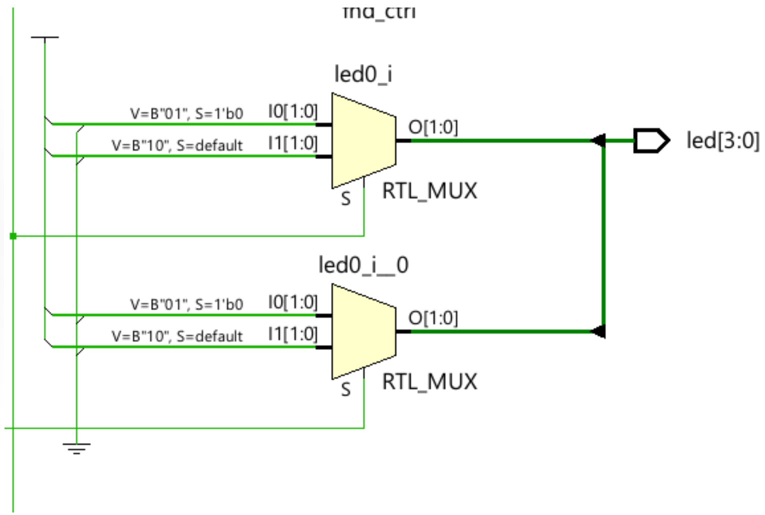
| | LED[1] | LED[0] |
|---|---|---|
| **SW[0] OFF** | | 🟩 |
| **SW[0] ON** | 🟩 | |

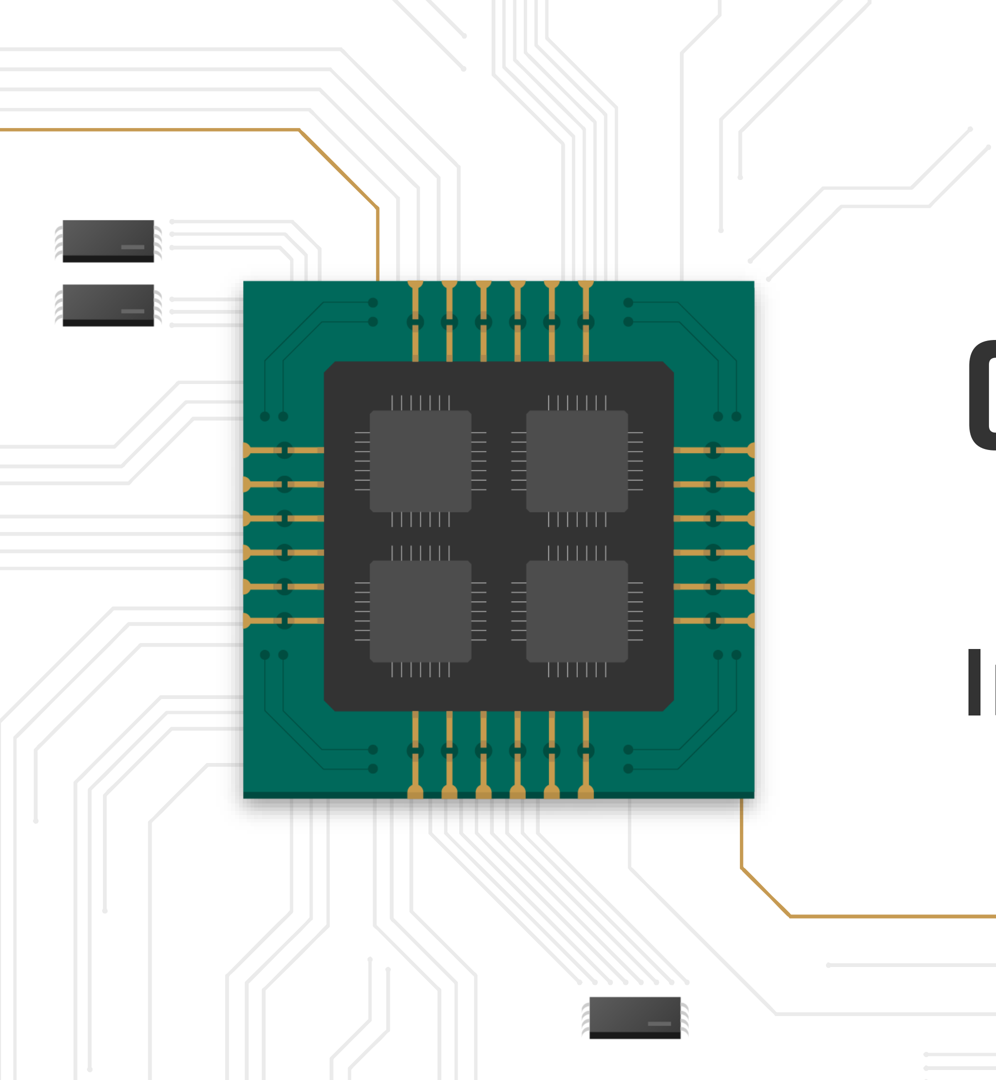
| | LED[3] | LED[2] |
|---|---|---|
| **SW[1] OFF** | | 🟩 |
| **SW[1] ON** | 🟩 | |

## LED_SEL: CODE



```
assign led[1:0] = (!sw0) ? 2'b01 : 2'b10;
assign led[3:2] = (!sw1) ? 2'b01 : 2'b10;
```





## 04

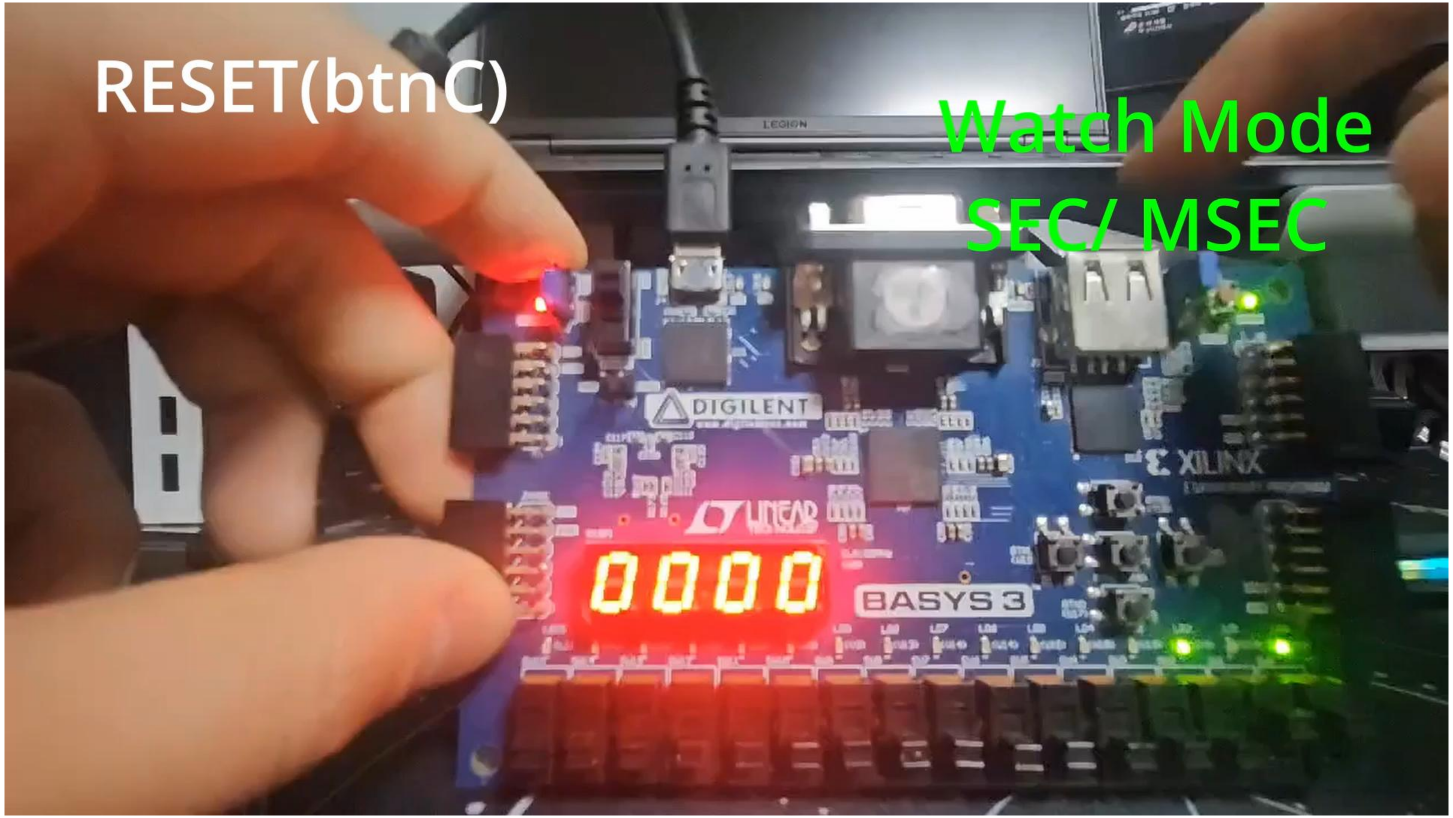
## Implementation

## Notice

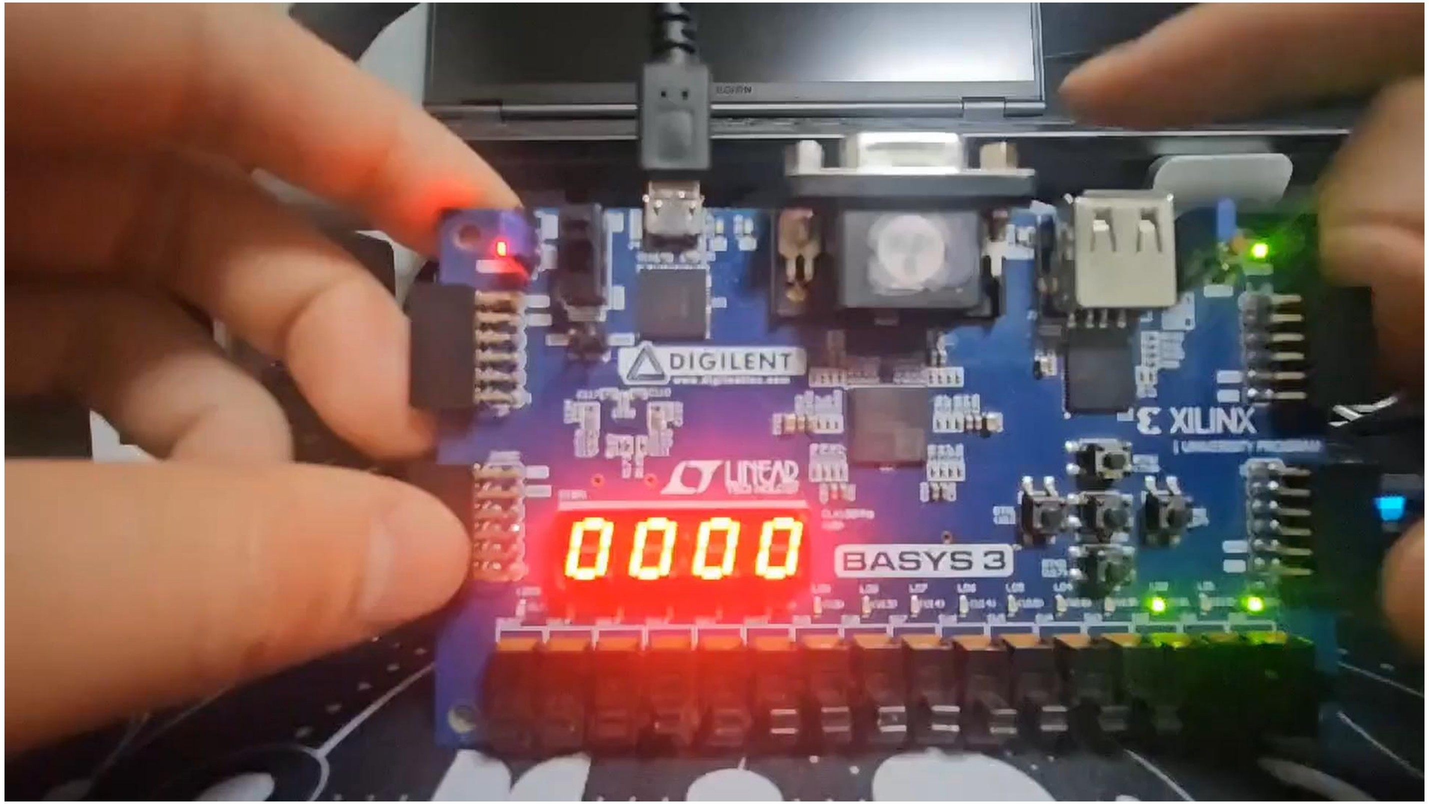
❗ SW[1] Short 문제

| Origin | Change |
|---|---|
| SW[0] | SW[2] |
| SW[1] | SW[3] |
| SW[2] | SW[4] |

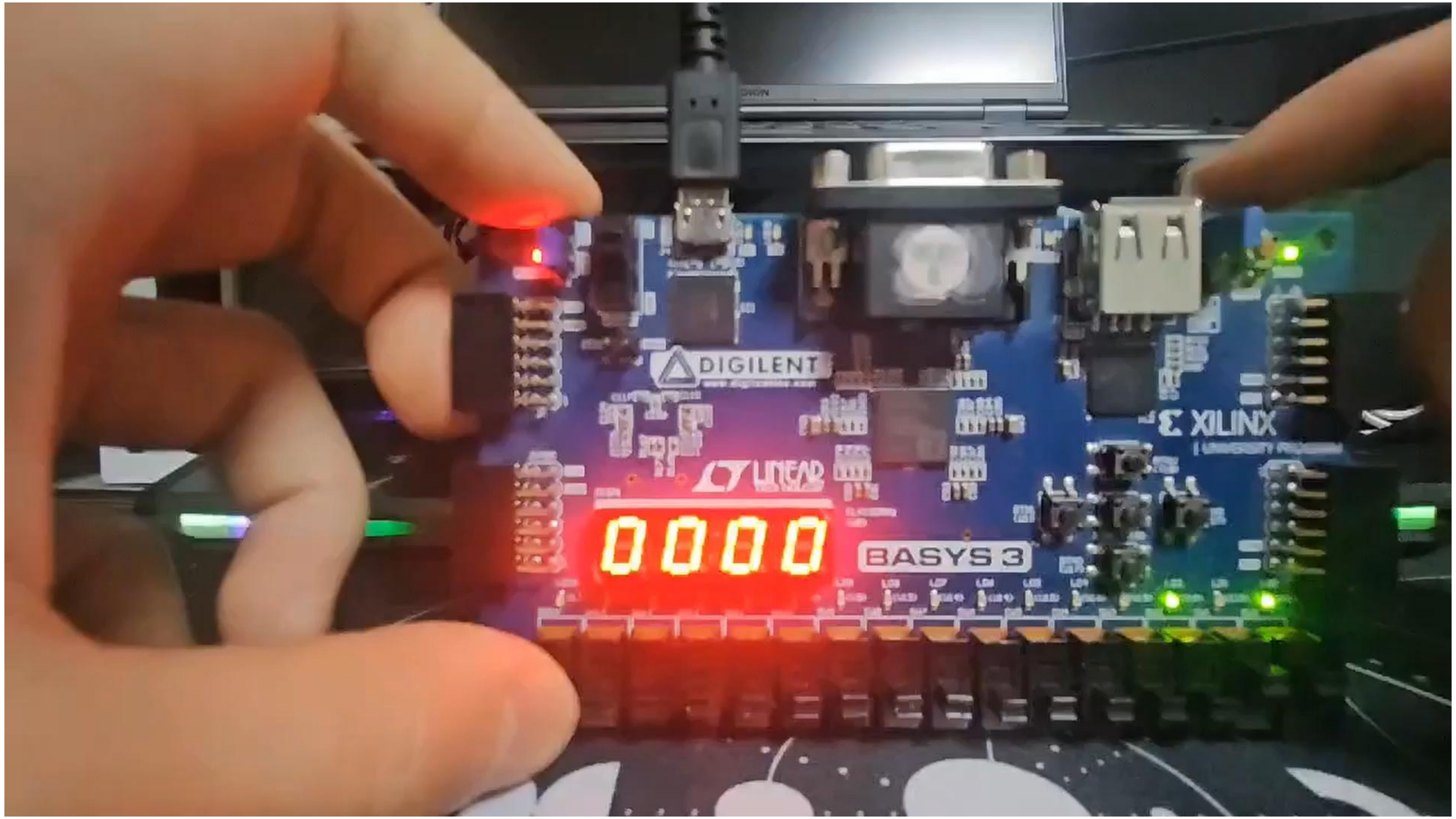
# Time Change-STOP

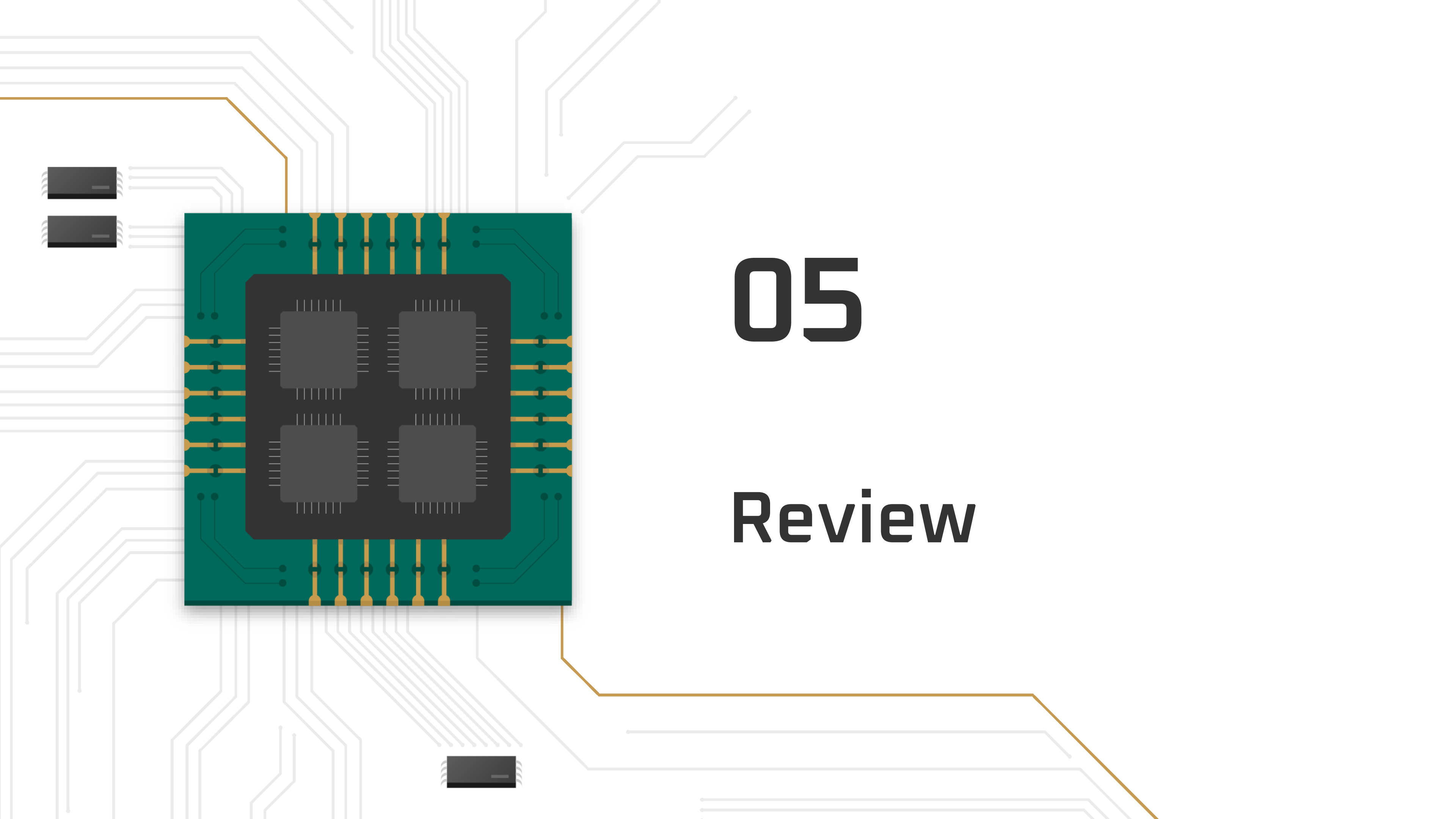


## Time Change-RUN



## Watch - Stop Watch





## 05

## Review

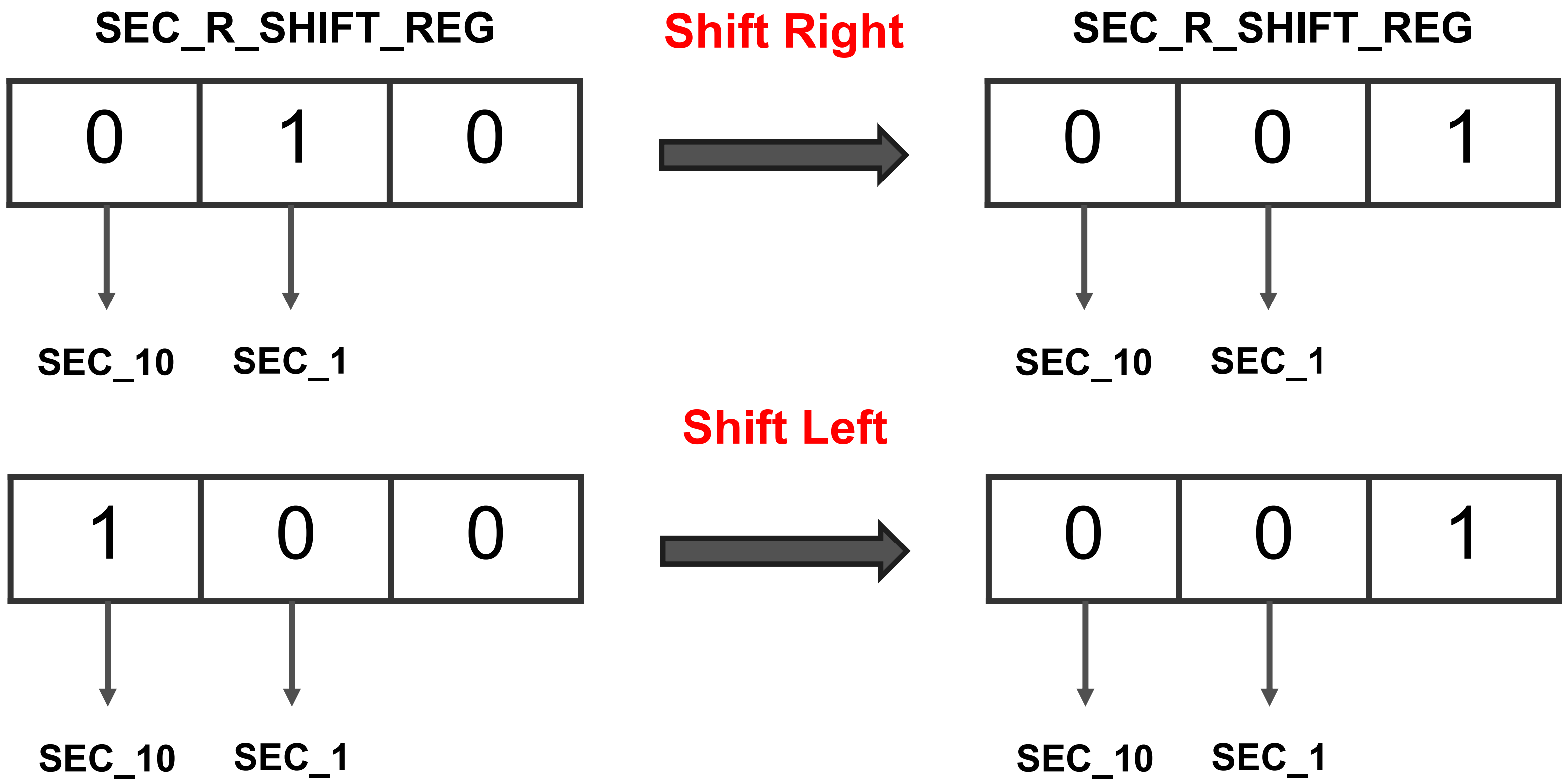
## 어려웠던 점1: Watch_CU

| Problem | Solution |
|---|---|
| FSM의 조건이 많음<br><br>→ 다단 MUX 발생<br><br>→ 타이밍 문제 가능성 | Rotate Shift Register 이용<br><br>→ 제어 신호 출력에 Shift reg 이용<br><br>→ 조건 수 감소<br><br>→ 다단 MUX 감소 |

## 어려웠던 점2: Watch_DP

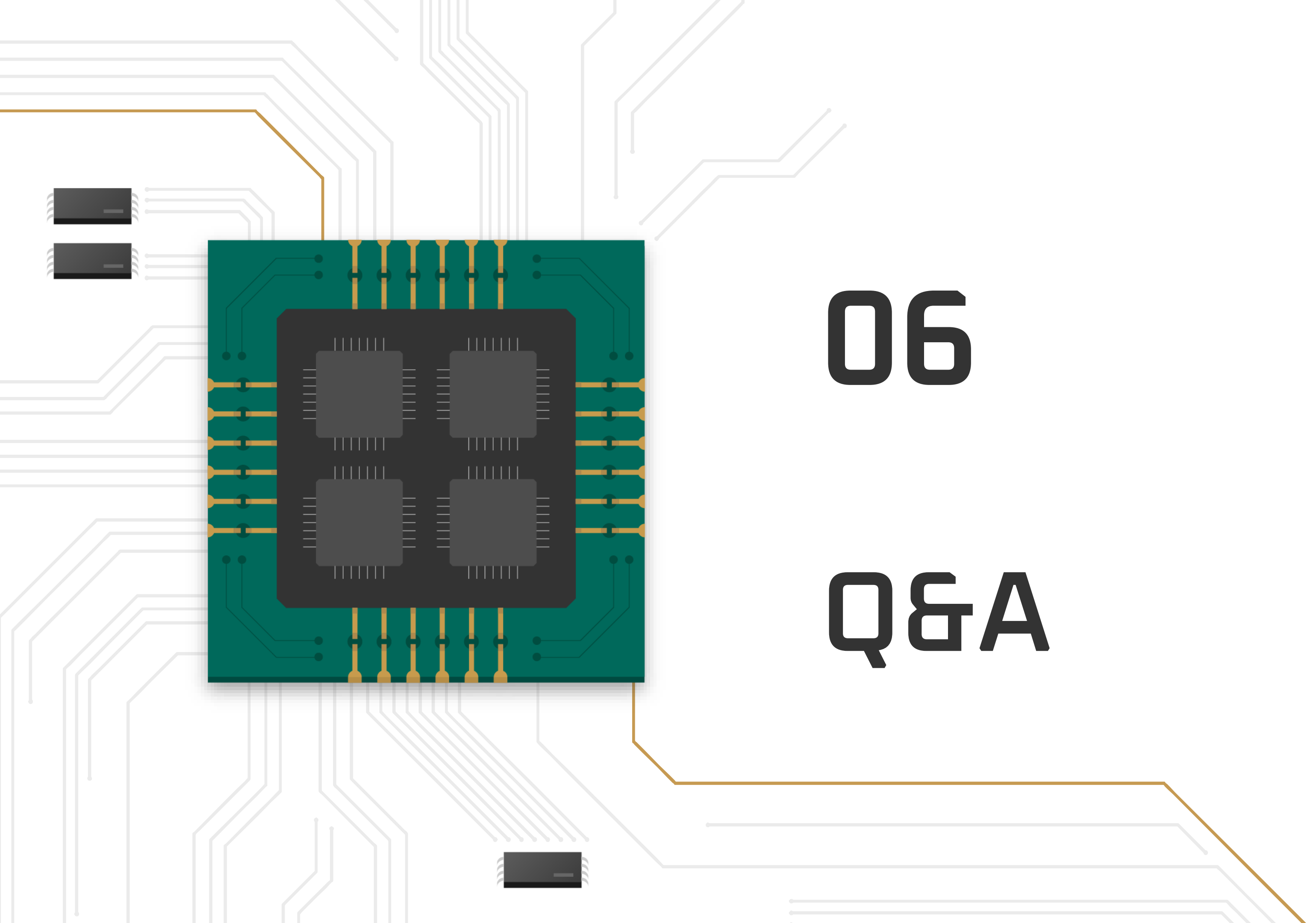
| Problem | Solution |
|---|---|
| 조건식 다수<br><br>→ 다단 MUX 문제<br><br>→ Case문 시도<br><br>→ Case문은 비교 조건 불가 | Flag를 이용한 Case문<br><br>→ 각 조건들에 대해 Flag 세팅<br><br>→ 하나의 Condition으로 묶음<br><br>→ CaseZ를 이용해 다대일 MUX 설계 |

# 아쉬운 점1: Watch_CU



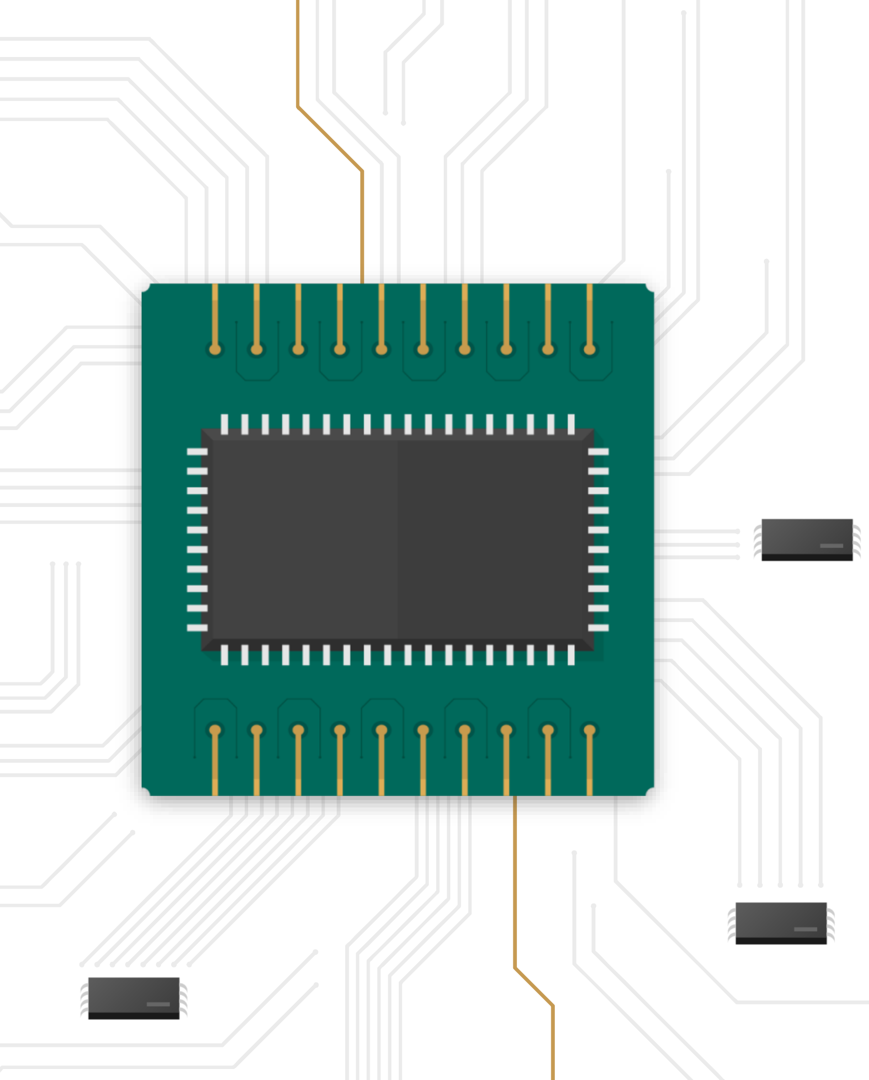
## 아쉬운 점2: CU 조건 이용

| 기존 | Update |
|---|---|
| Watch CU에서 컨트롤 신호 생성 | FND_COM_CTRL FSM 사용X |
| → DP를 제외한 FND CTRL 등에서 사용 X | → Watch CU에서 생성한 제어 신호 사용 |
| → 신호 낭비 | → FSM 필요 X |
| → AREA 증가 | → Area 감소 |
|  | → Performance 향상 |



## 06

## Q&A



# Thanks!