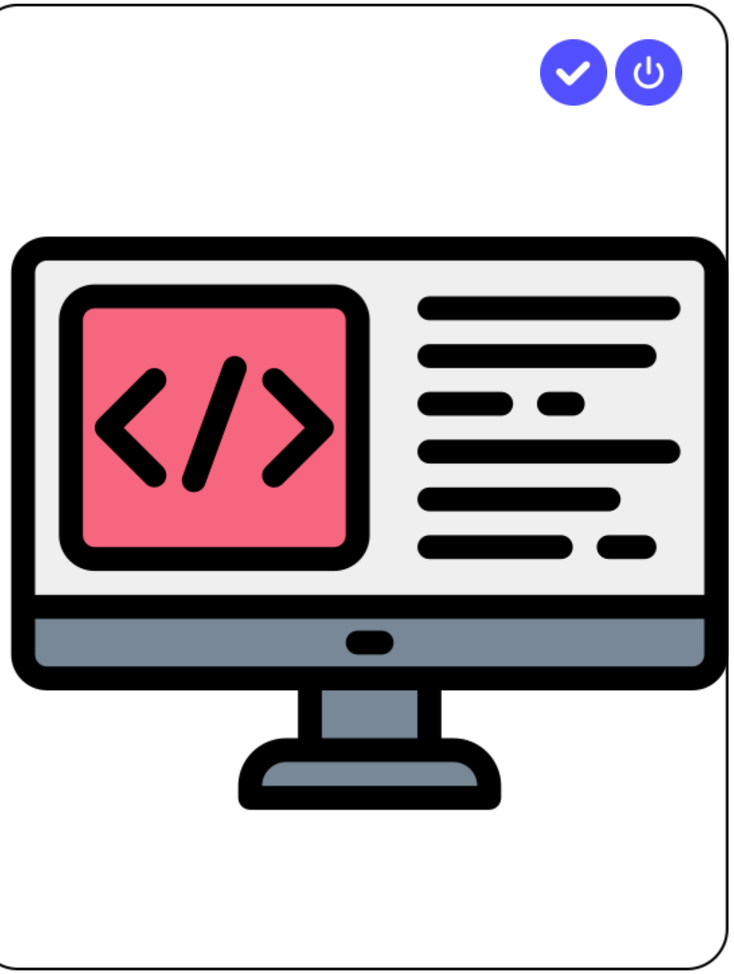
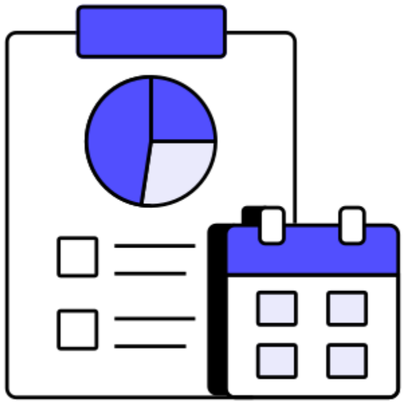
유지훈

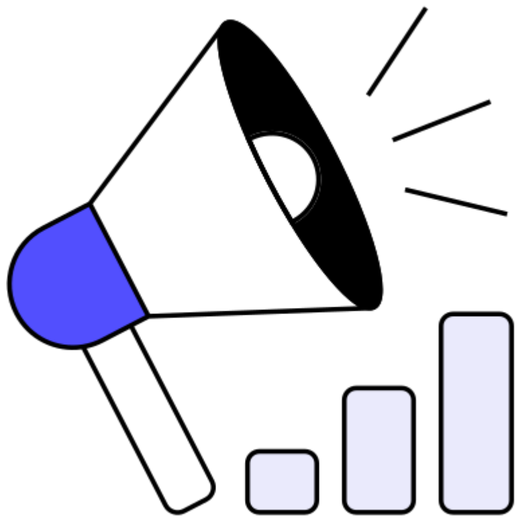
STOPWATCH/ WATCH



목차



01 Introduction



기능 (Function)

사용자의 버튼 입력을 기반으로:
스톱워치 시작/정지
시간 초기화 (Clear)
시간 측정 (ns, s, min, hour 단위로)측정
4자리 FND로 시간 표시 (모드 전환 가능)

목표

디지털 시계 및 스톱워치 모듈구현 FPGA 보드를 통한 구현 모듈 검증

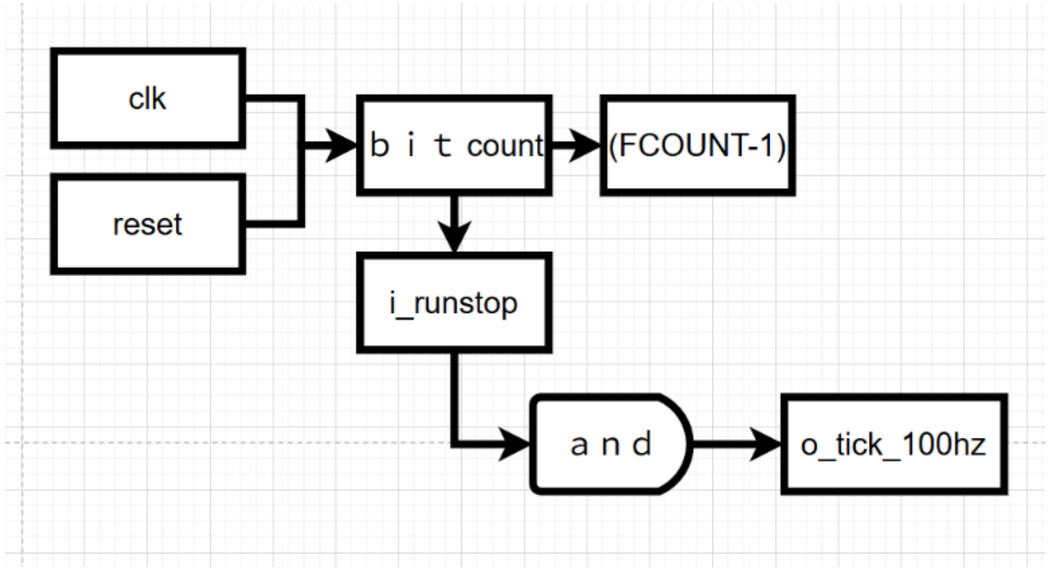
요약

Verilog로 구현된 디지털 스톱워치 시스템이다.
FSM 기반의 제어 유닛, 시간 카운터(Data Path),
버튼 디바운싱, FND 표시
Basys 3로 구현 및 시뮬레이션을 통해 검증

핵심기능

- STAR, STOP: Btn_R 눌러서 스톱워치 실행/정지 제어
- CLEAR: Btn_L 눌러 시간 초기화
- FND 표시: sw[0]으로 ns / s/ min/ hour 표시

02 BLOCK Diagram



tick_gen_100Hz

비트 카운터가 클럭마다 1씩 증가.

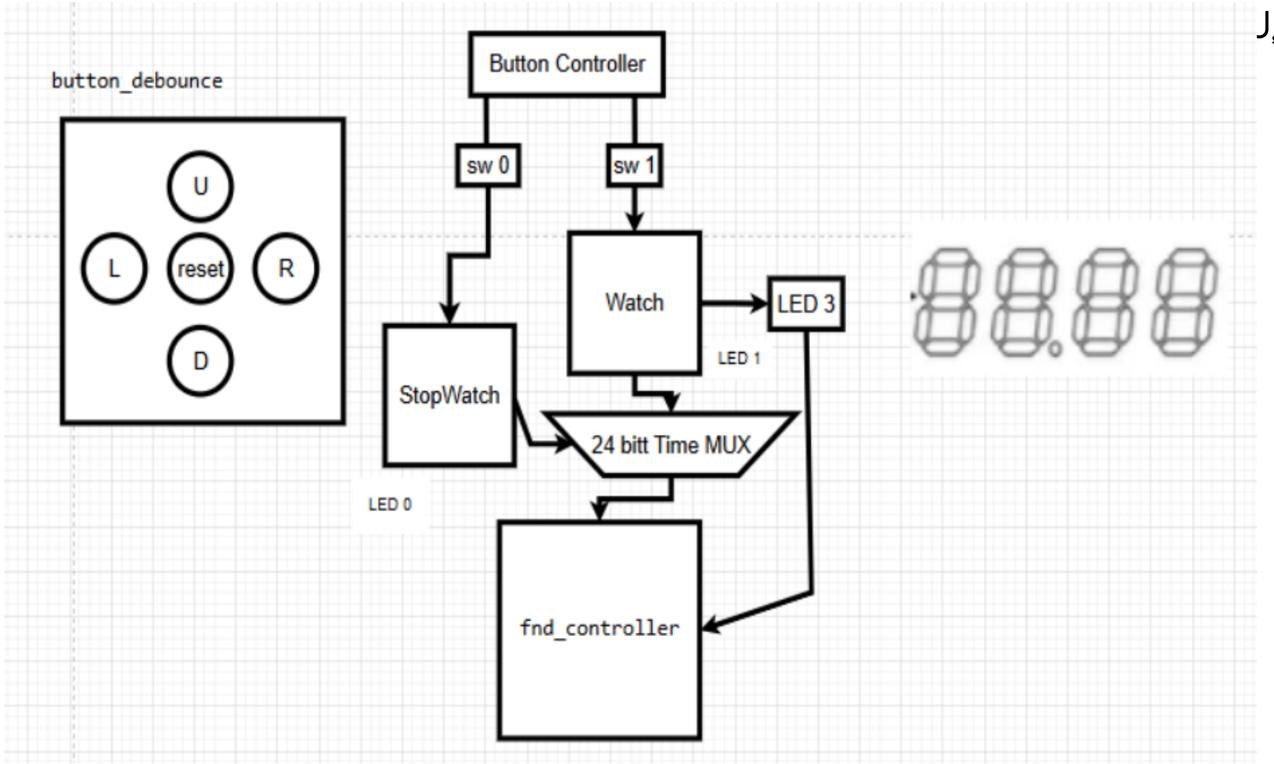
지정된 값(FCOUNT-1)에 도달하면 1클럭짜리 펄스 발생 후 0으로 리셋.

i_runstop이 1일 때만 카운터 동작하도록 AND 게이트로 제어.
N비트 카운터가 클럭마다 1씩 증가.

지정된 값(FCOUNT-1)에 도달하면 1클럭짜리 펄스 발생 후 0으로 리셋.

i_runstop이 1일 때만 카운터 동작하도록 AND 게이트로 제어.

02 BLOCK Diagram



button_debounce

J, D, L, R, reset 버튼 각각에 적용 → 깨끗한 펄스 신호 생성

Button Controller

SW[0], SW[1] 스위치 상태를 읽어 현재 모드 결정

Watch 또는 Stopwatch로 제어 신호 분기

Watch (watch_cu + watch_dp)

시계 모드 시간 유지 및 수정

LED1: 현재 시계 모드임을 표시

StopWatch (stopwatch_cu + stopwatch_dp)

스톱워치 동작(RUN/STOP/CLEAR) 제어

LED0: 스톱워치 RUN 상태 표시

24bit Time MUX

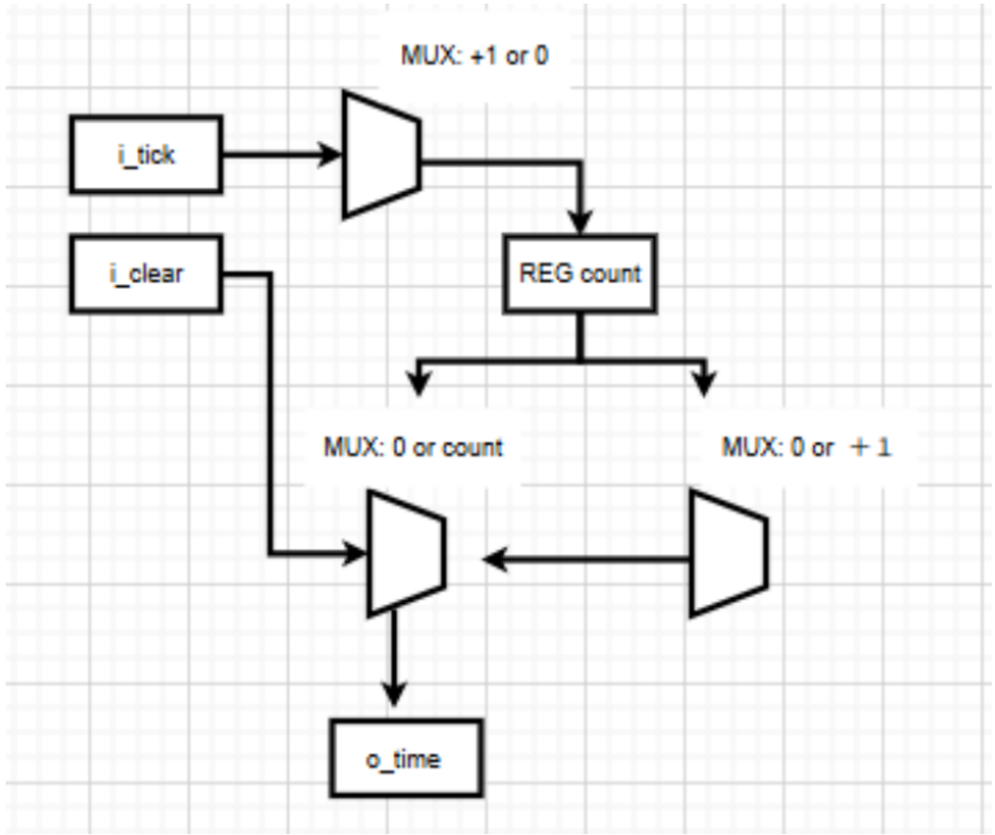
SW[1]에 따라 Watch 시간 또는 Stopwatch 시간 선택

fnd_controller

선택된 시간 데이터를 7세그먼트로 표시

LED 숫자 모양 출력

02 BLOCK Diagram



time_counter

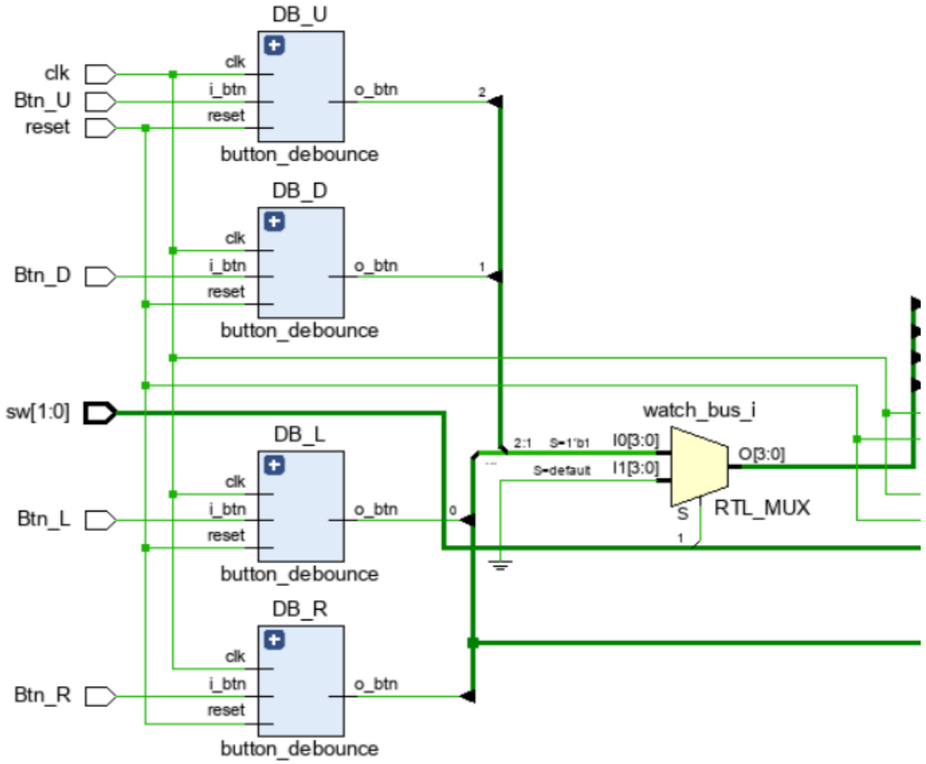
i_tick=1일 때 카운터+1.

i_clear=1이면 언제든 카운터 0으로 초기화.

MAX 값에서 +1이면 0으로 돌아가며
o_tick_up=1을 출력(상위 단위로 carry).

msec→sec→min→hour 단위 증가 신호 전파.

03 Schematic



button_debounce

버튼 입력의 노이즈를 제거하고, 1클럭 펄스로 변환하는 회로

구성:

동기화 - 비동기 입력을 클럭에 맞춤

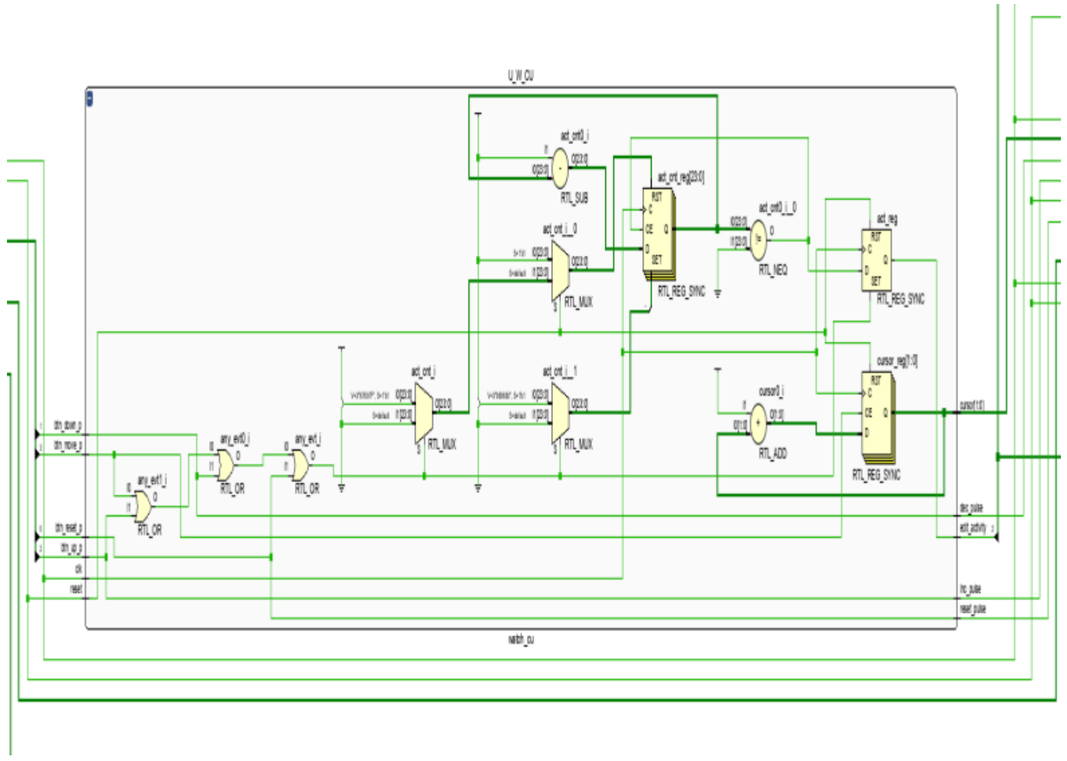
카운터 - 입력이 변하지 않는 시간 측정

비교기 - 안정 상태 판단

엣지 검출기 - 1클럭 폭 신호 출력

결과: 버튼을 한 번 눌러도 여러 번 인식되는 문제 방지

03 Schematic



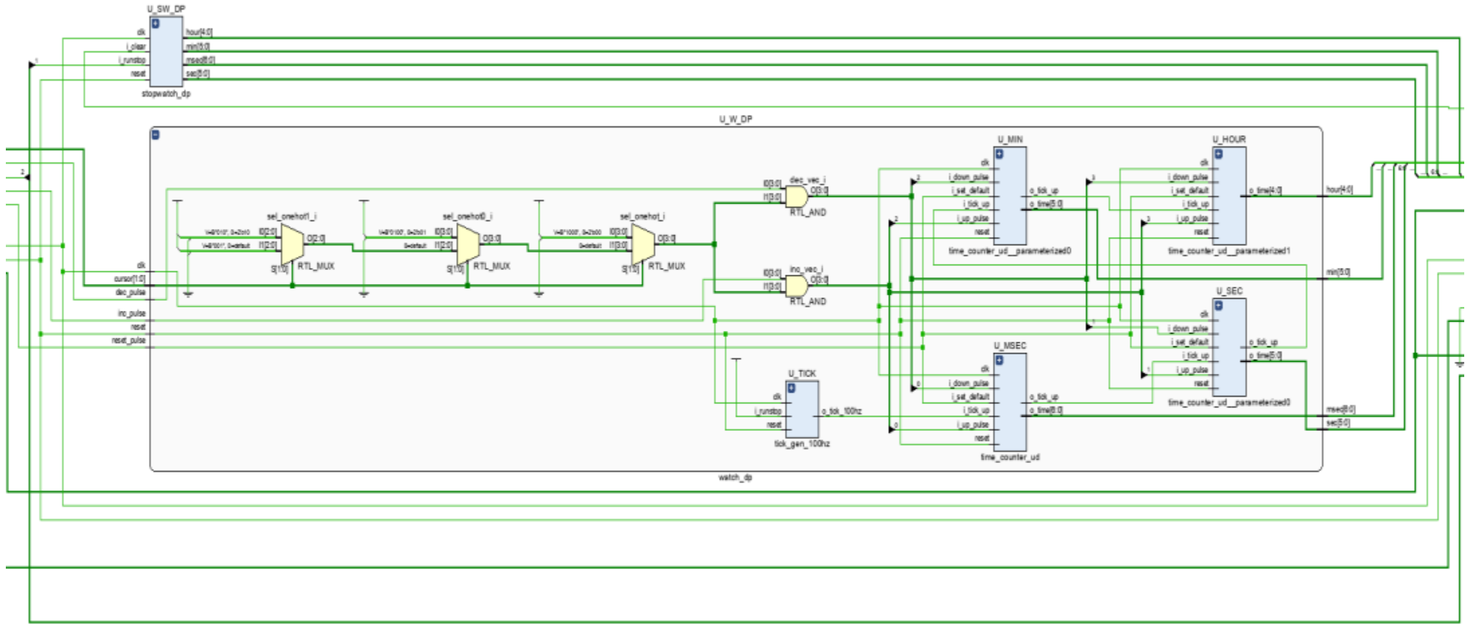
watch_cu

시계 모드에서 커서 이동, 증가, 감소, 리셋 같은 동작 제어
버튼 입력 펄스를 기반으로 내부 제어 신호 생성

원리:
버튼 → move_p, up_p, down_p, reset_p 펄스 입력
move_p → cursor 값(시/분/초/밀리초 선택) 변경
up_p, down_p → 선택된 자리수 값 증가/감소 발생
reset_p → 전체 시간 리셋

출력:
cursor[1:0] (현재 선택 위치)
inc_pulse, dec_pulse, reset_pulse
edit_activity (편집 모드 동작 표시용)

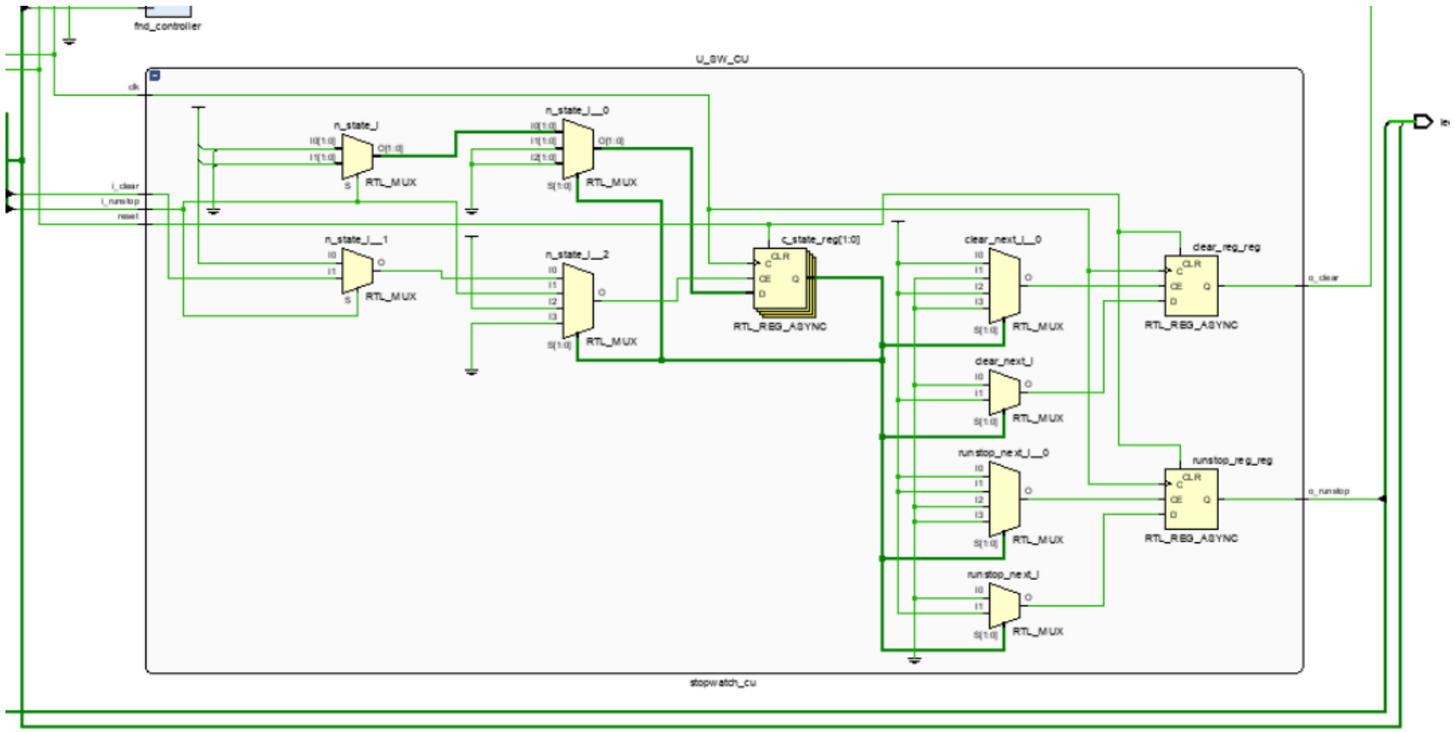
03 Schematic



watch_dp

시계 모드의 데이터 처리 경로
초, 분, 시, 밀리초 카운터 관리
원리:
100Hz tick 입력 → msec, sec, min, hour 카운터 업데이트
inc_pulse, dec_pulse → 선택된 cursor 위치 값 수정
reset_pulse → 전체 카운터 초기화
출력: 현재 시계 값 (hour, min, sec, msec)

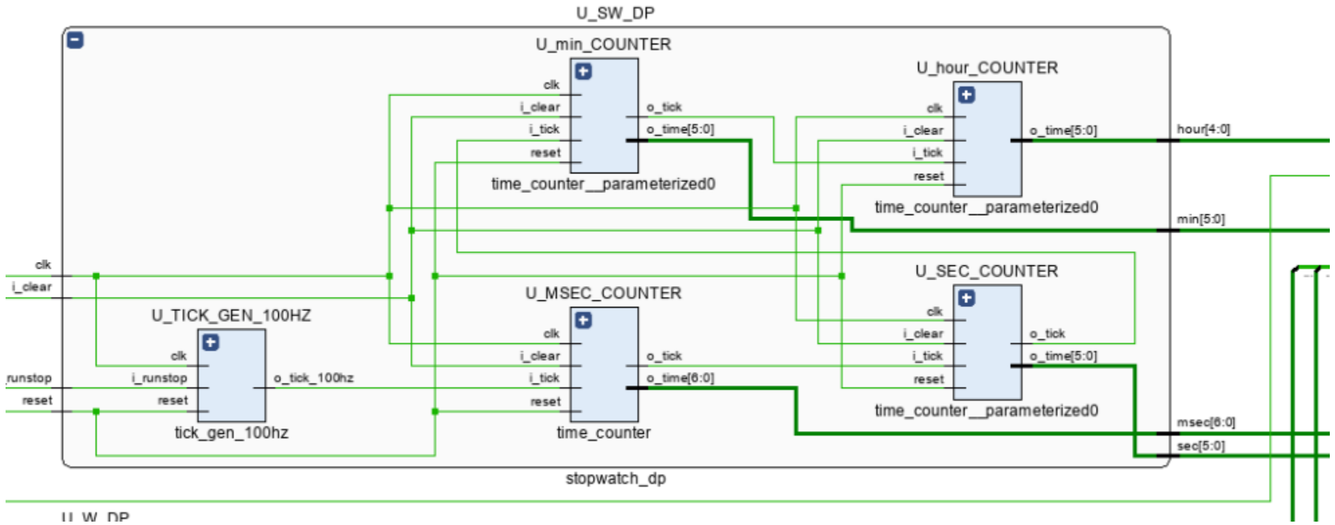
03 Schematic



stopwatch_cu

스톱워치의 동작 상태 제어 (RUN / STOP / CLEAR)
버튼 입력에 따라 상태 전환
원리:
FSM(유한 상태 기계) 기반
RUN 상태 → 카운터 동작, STOP 상태 → 카운터 유지
CLEAR 입력 시 카운터 리셋 후 STOP 상태로 복귀
출력:
o_runstop (동작 여부)
o_clear (리셋 신호)

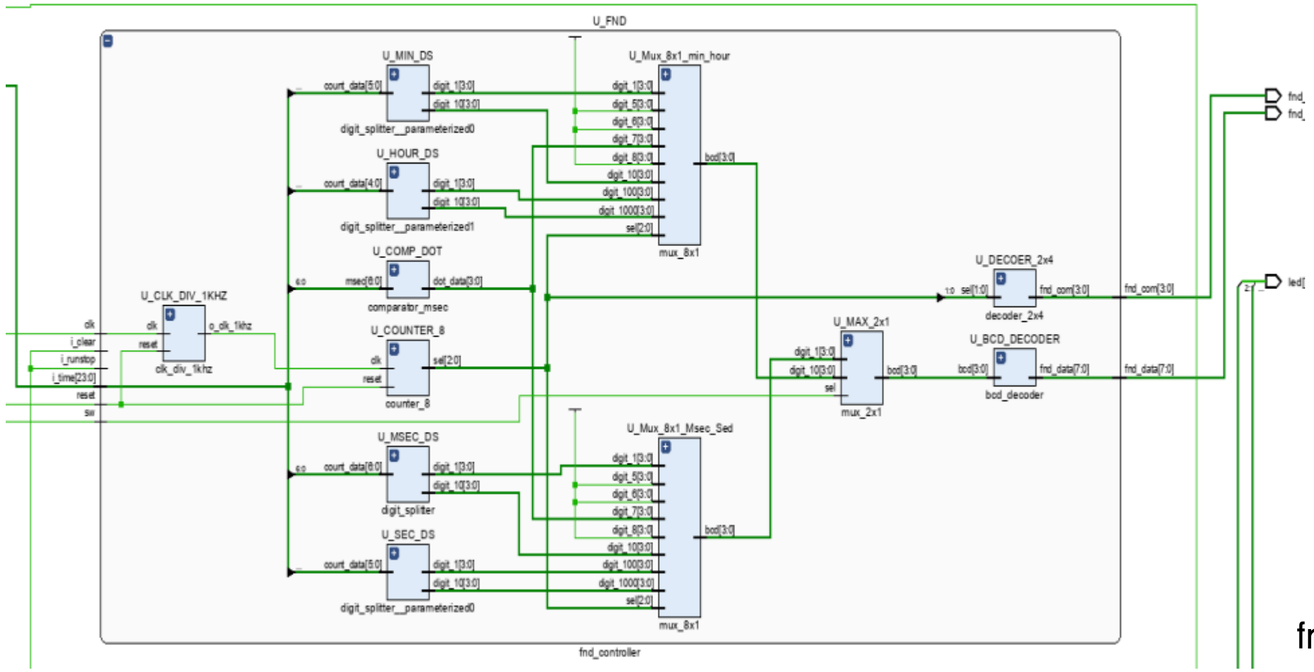
03 Schematic



stopwatch_dp

스톱워치의 시간 카운터 처리
밀리초, 초, 분, 시 단위로 카운트
원리:
tick_gen_100hz 모듈로부터 100Hz tick 입력
o_runstop이 1일 때만 카운트 증가
o_clear 신호 시 전체 카운터 초기화
출력: sw_hour, sw_min, sw_sec, sw_msec

03 Schematic

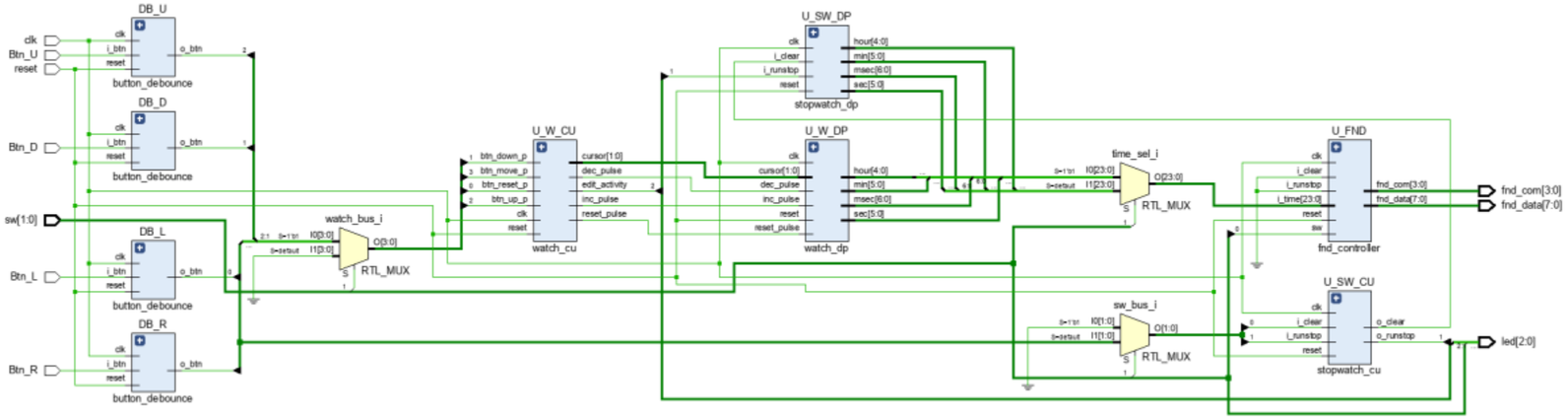


fnd_controller

현재 시간 데이터를 7세그먼트(FND)에 표시
시간 데이터를 BCD 분해 → 디코딩 → 자리별 출력

원리:
Digit Splitter → 24bit 시계/스톱워치 데이터를 각 자리수 BCD로 분리
Multiplexer → 빠르게 자릿수를 순차 선택하여 표시(멀티플렉싱)
BCD to 7-Segment Decoder → BCD → FND 패턴 변환
출력:
fnd_com (자리 선택), fnd_data (세그먼트 LED 점등 데이터)

03 Schematic



FSM(stopwatch_cu)에서 생성한 제어 신호
o_runstop, o_clear를 받아서

o_runstop, o_clear를 받아서,
ns, s, min, hour를 카운트 해 시간을 계산하는 역할을 합니다.

04 시물레이션 결과

sw[1:0]에서 sw[1]=0 → MUX가 time_sel = time_sw(= stopwatch 시간)을 선택.
FND에는 스톱워치의 {hour,min,sec,msec}가 바로 표시됨.

버튼 매핑: run_p=b_r, clr_p=b_l 만 살아있고, mv/up/dn/rst는 0으로 게이팅됨.
→ watch_cu의 inc/dec/reset/cursor는 모두 0(편집 비활성).

U_SW_CU.o_runstop가 토글되면

1일 때: tick_gen_100hz가 동작 → msec(0..99) 증가, 99→0 시 sec+1, sec==59→0 시 min+1… 로 캐리 전달.

0일 때: 모든 카운터 멈춤.

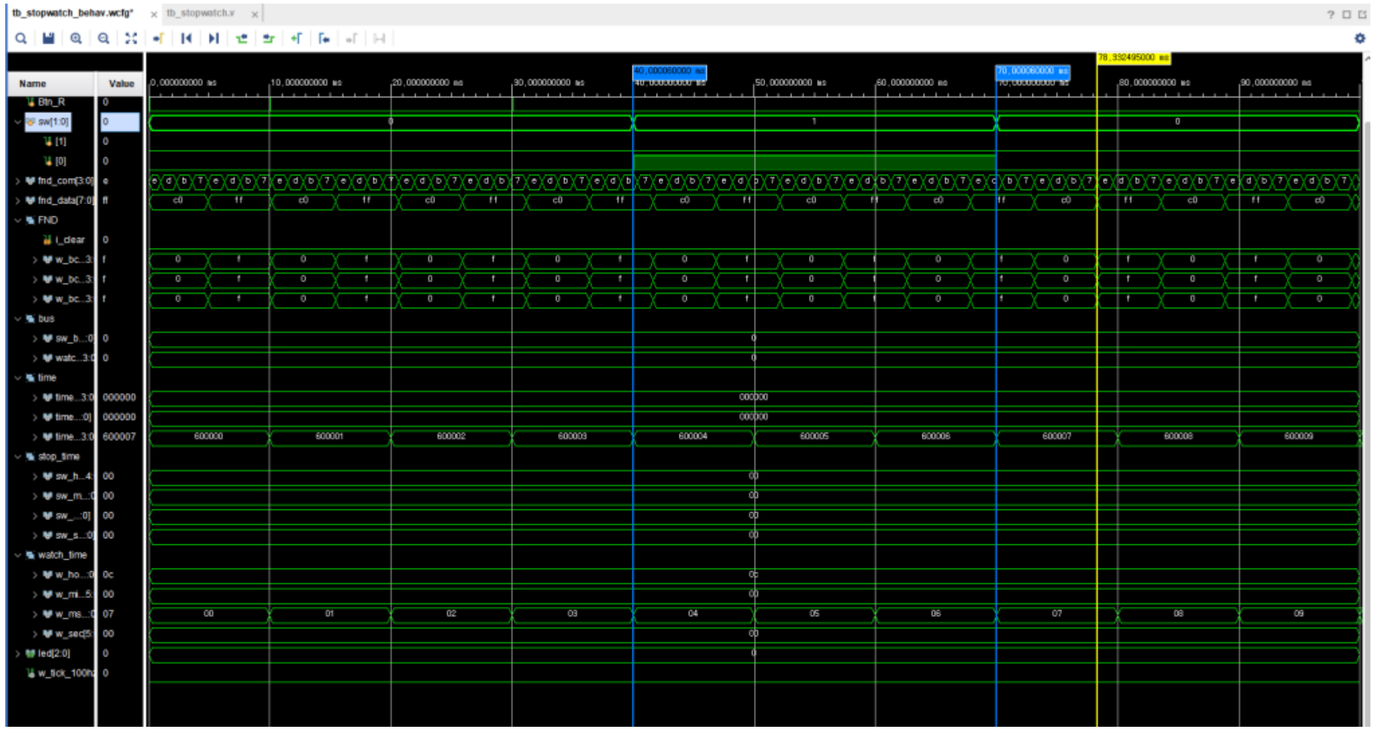
04 시물레이션 결과

RUN 상태에서 clear(Btn_L)는 무시됨(우리 CU 설계: RUN 상태는 clear 무시).
STOP 상태에서 clear면 1클럭 펄스가 나가고 네 자리 모두 0으로 클리어.

LED
led[1]=o_runstop 이므로 RUN 구간에서 High, STOP에서 Low.
led[0]=sw[1]=0 (현재 스톱워치 모드 표시)
led[2]=watch_edit_act=0 (편집 없음)

FND 표시 전환(sw[0])
sw[0]=0: 아래쪽(초:밀리초) 4자리 표시
sw[0]=1: 위쪽(시:분) 4자리 표시
(내부에서 mux_8x1 두 개 + mux_2x1로 구성. dot은 msec<50 구간에만 켜짐)

04 시물레이션 결과



sw[1]=1 → MUX가 time_sel = time_w(= watch_dp 시간) 선택.
FND에는 시계의 {hour,min,sec,msec}가 나타남(초/밀리초는 100 Hz 자동 카운트).

버튼 매핑:
watch_btn_p={b_r,b_u,b_d,b_l} → {mv_p,up_p,dn_p,rst_p}로 통과,
반대로 스톱워치 쪽 run_p/clear_p는 0으로 게이팅 → 스톱워치 CU에는 입력이 안 들어감.

04 시물레이션 결과

watch_cu

btn_move_p(b_r)를 줄 때마다 cursor가 순환(시→분→초→밀리초)

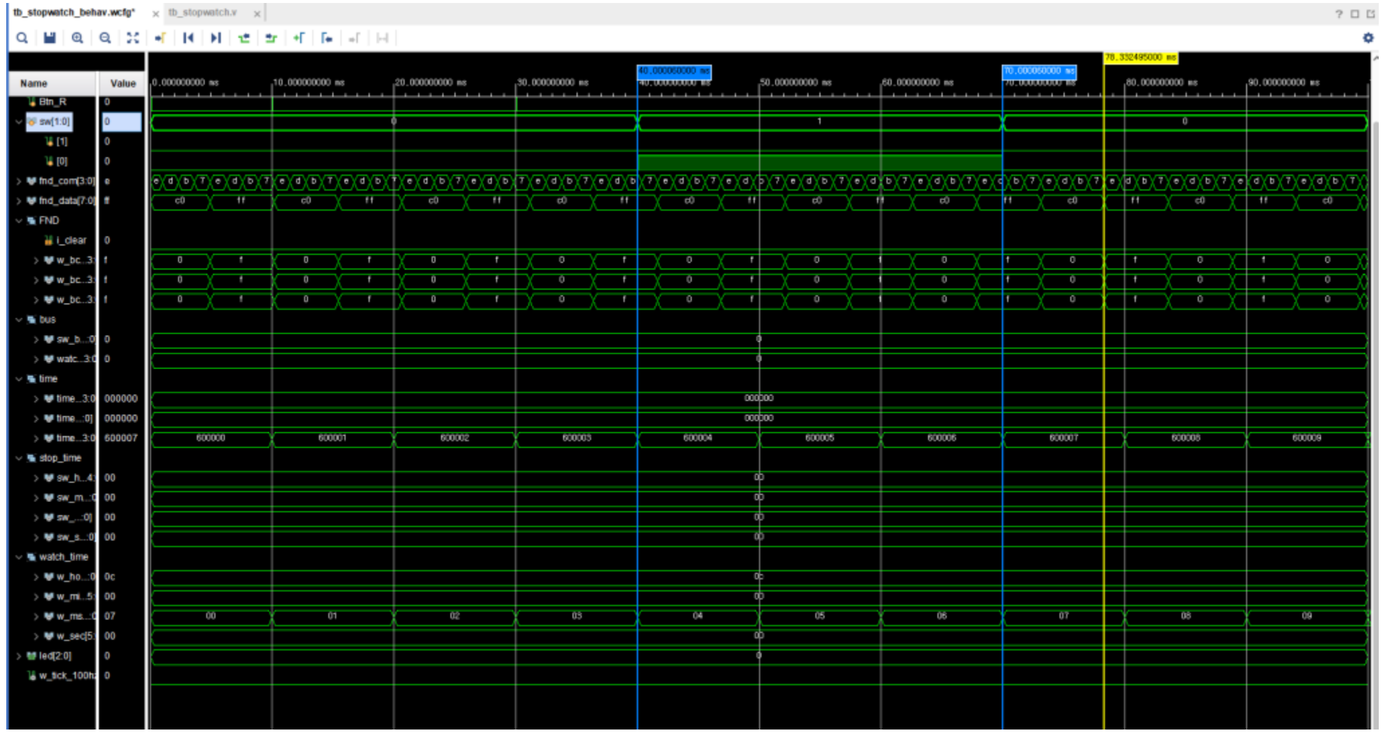
btn_up_p/btn_down_p로 해당 필드만 +1/−1 (범위 wrap: h 0..23, m/s 0..59, ms 0..99)
btn_reset_p(b_l)로 기본값(12:00:00.00) 복귀
이벤트가 들어오면 edit_activity가 약 0.1 s High 유지(LED2로 확인 가능)

“모드만 바꾼 경우”

스톱워치가 RUN 중이었다면 백그라운드에서 계속 돈다.
(sw[1]=1일 때는 단지 입력을 차단했을 뿐 CU의 상태는 유지)

다시 sw[1]=0으로 돌아오면 그 사이 진행된 스톱워치 값이 FND에 나타남.

04 시물레이션 결과



LED

led[0]=1 (워치 모드 표시)

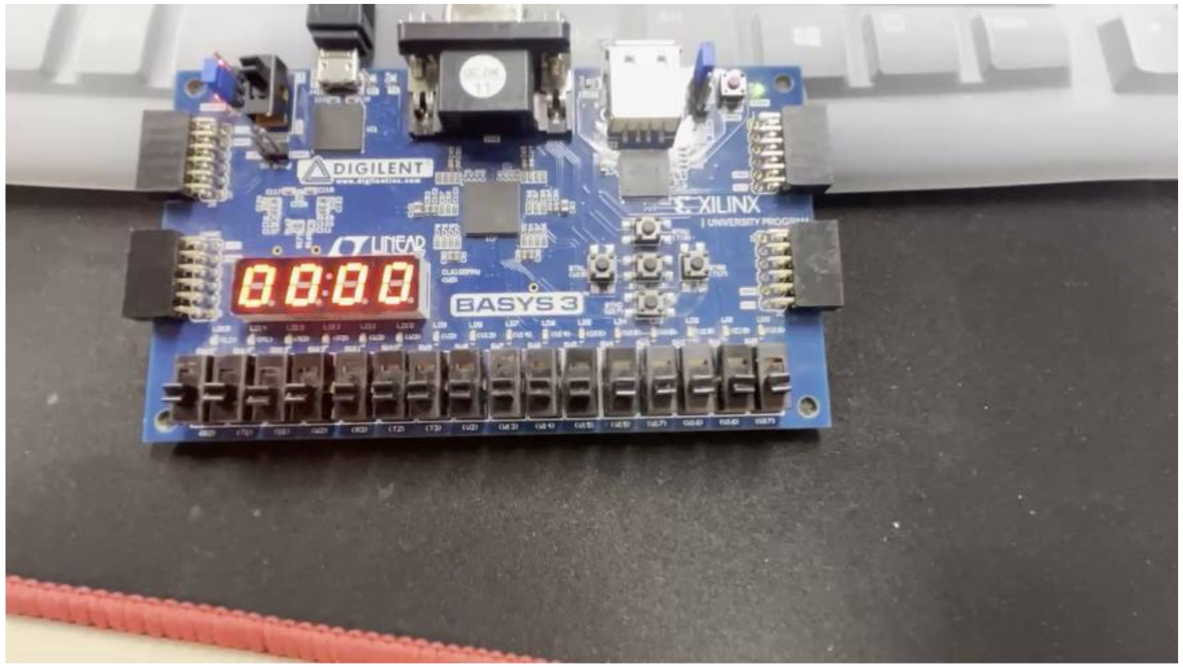
led[1]=o_runstop은 모드와 무관하게 CU 레벨(스톱워치 RUN 여부)을 그대로 반영

led[2]=edit_activity는 편집 이벤트가 있을 때 0.1 s 점등

FND 표시 전환(sw[0])

sw[0]=0이면 “초:밀리초”, sw[0]=1이면 “시:분” 표시.
워치 모드에서도 동일하게 동작.

05 5.동작영상



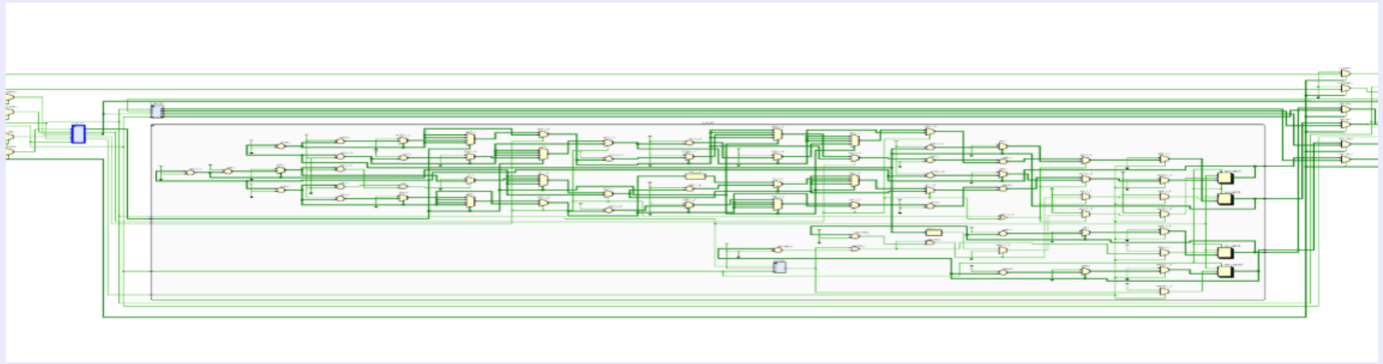
06 트러블슈팅&고찰(버튼/모드 분기 난잡 · 오동작)

Stopwatch/Watch에 버튼이 동시에 먹거나, 스키매틱에 AND/MUX가 난잡해짐

해결: “버스화 + 모드 게이팅 1회”

스키매틱에서 AND/MUX 뭉치 사라지고 RTL_MUX 2개만 남음.

모드 전환 시 반대쪽 블록에 버튼이 절대 전달되지 않음(0으로 게이트).

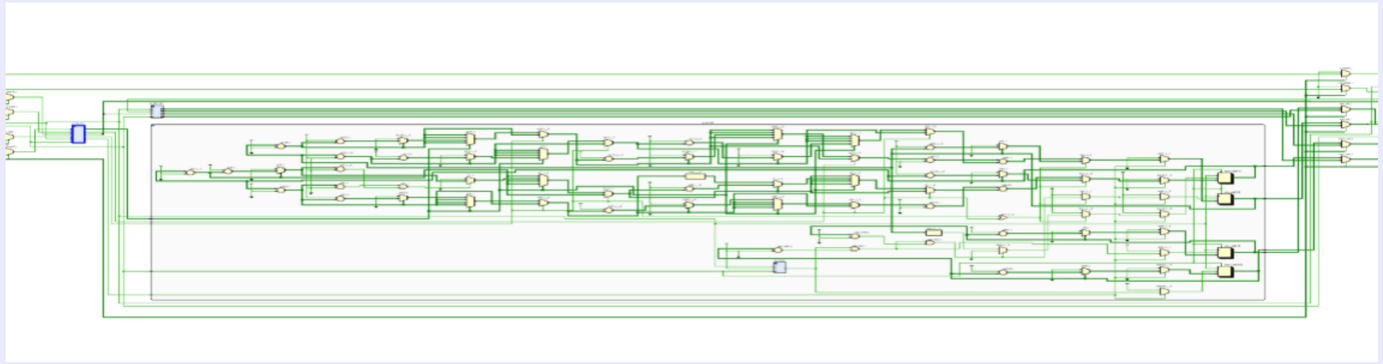


```
wire mode = sw[1];                    // 0: SW, 1: Watch
wire [3:0] watch_btn_p = {b_r,b_u,b_d,b_l};
wire [1:0] sw_btn_p    = {b_r,b_l};

wire [3:0] watch_bus   = mode ? watch_btn_p : 4'b0000;
wire [1:0] sw_bus      = mode ? 2'b00       : sw_btn_p;

assign {mv_p,up_p,dn_p,rst_p} = watch_bus;  // watch_cu 입력
assign {run_p,clr_p}          = sw_bus;     // stopwatch_cu 입력
```

06 트러블슈팅&고찰(시간 선택 MUX 4개 → 1개)

hour/min/sec/msec 각각 ?: 하다 보니 MUX 4개 생성, 배선 복잡.

해결: “24-bit 시간 버스 단일 MUX” 스키매틱에 time_sel_i RTL_MUX 1개만 표시.
FND는 항상 i_time(time_sel)만 받음 → 표시 일관.

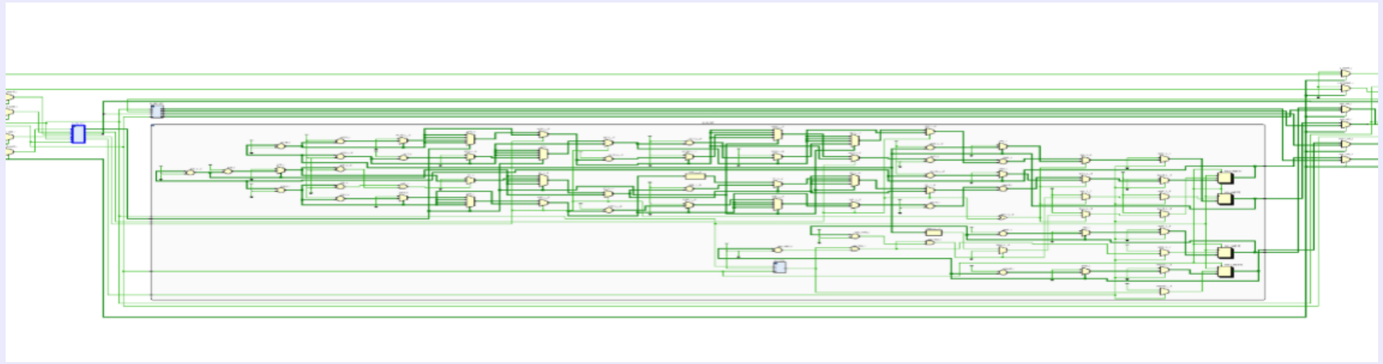


```
wire [23:0] time_sw = {sw_hour, sw_min, sw_sec, sw_msec};
wire [23:0] time_w  = { w_hour,  w_min,  w_sec,  w_msec};
wire [23:0] time_sel = mode ? time_w : time_sw;
```

06 트러블슈팅&고찰(watch_dp)

자리별 inc/dec AND 게이트가 수십 개, 스키매틱 난잡.

스키매틱에서 AND 덩어리 제거 → (msec→sec→min→hour).



```
wire [3:0] sel_1hot = (cursor==2'b00)?4'b1000:
                      (cursor==2'b01)?4'b0100:
                      (cursor==2'b10)?4'b0010:4'b0001;
wire [3:0] inc_vec = {4{inc_pulse}} & sel_1hot;
wire [3:0] dec_vec = {4{dec_pulse}} & sel_1hot;

time_counter_ud #(.BIT_WIDTH(7), .TIME_COUNT(100)) U_MSEC (
  .i_tick_up(tick_100hz), .i_up_pulse(inc_vec[0]), .i_down_pulse(dec_vec[0]),
  .i_set_default(reset_pulse), .o_time(msec), .o_tick_up(sec_tick)
);
```