UART FIFO & SENSOR & STOPWATCH & WATCH

Final Project

대한상공회의소 서울기술교육센터
AI 시스템 반도체 설계 (2기)

7조 서윤철, 권희식, 박승헌, 오정일

nextchip

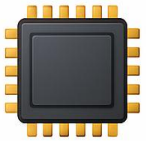
KFIA 한국팹리스산업협회 Korea Fabless Industry Association

대한상공회의소 서울기술교육센터

INDEX

초음파 센서 모듈

온도/습도 센서 모듈



전체 시스템 모듈

개요

■ 프로젝트 목표

UART 통신 이해와
검증 및 구현

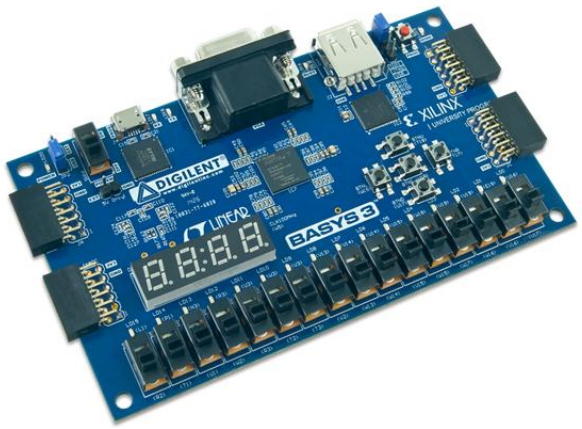
FIFO 메모리 구조
이해와 구현

Sensor의 이해 및
동작 응용

Verilog를 이용한 설계,
디버깅을 통한 error 수정

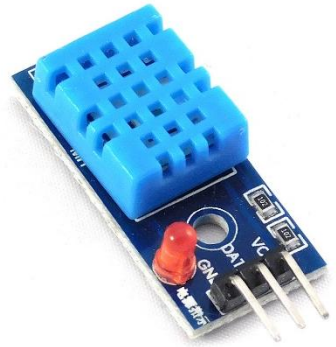
개요

■ 개발 환경



Basys 3 board

HC-SR04 초음파 센서



DHT11 온도/습도 센서

VIVADO HLx Editions

Vivado 2020.2

Visual Studio Code

VS code

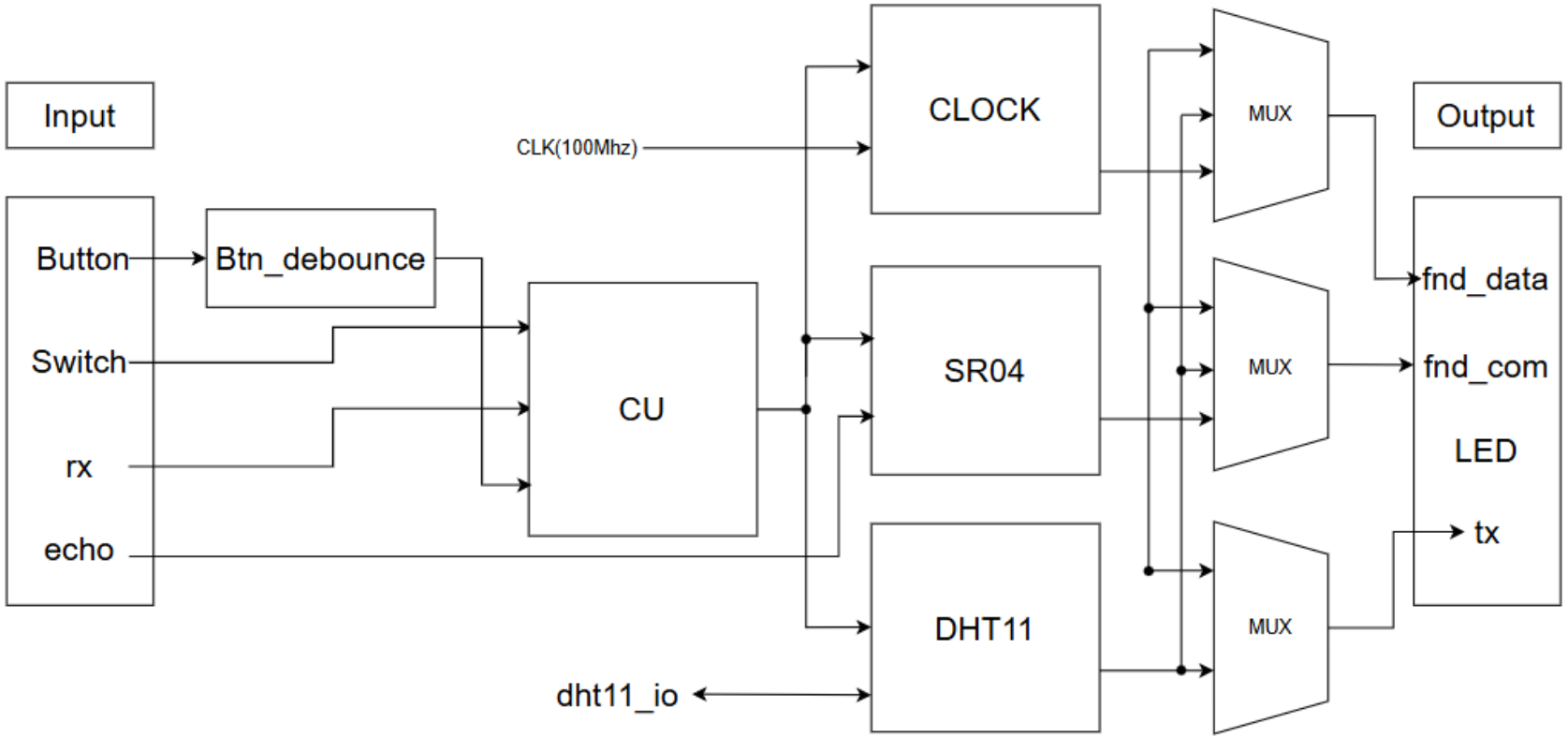
nextchip

KFIA 한국팹리스산업협회 Korea Fabless Industry Association

대한상공회의소 서울기술교육센터

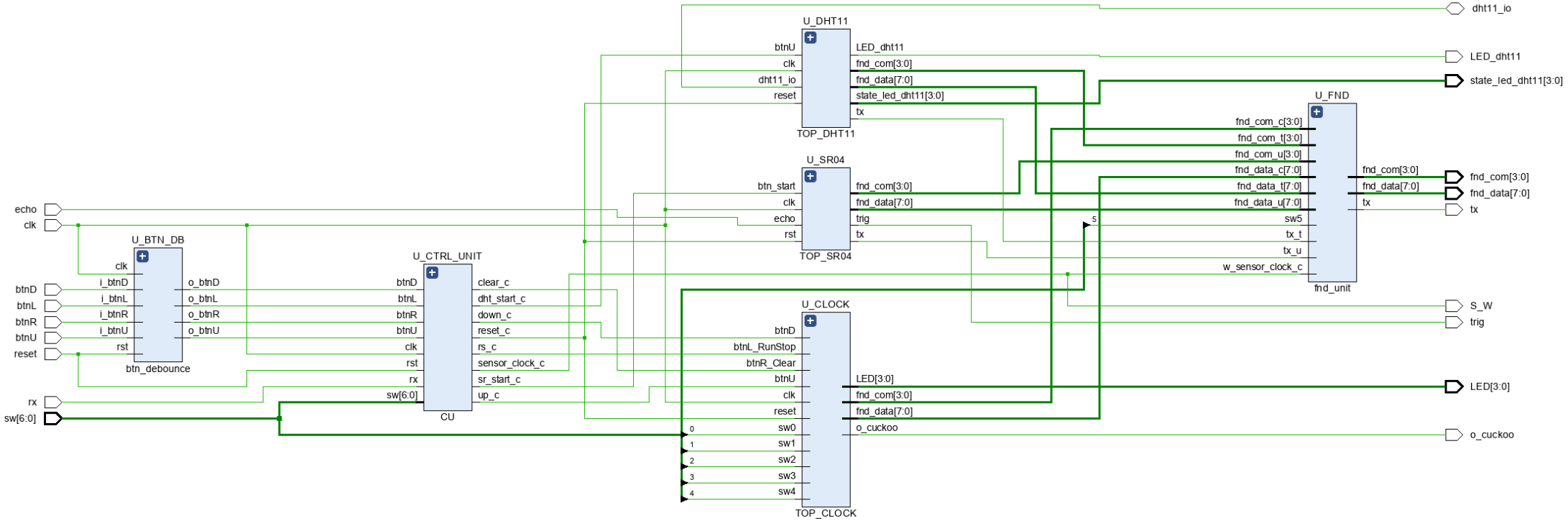
설계 과정

Entire Module Block Diagram



설계 과정

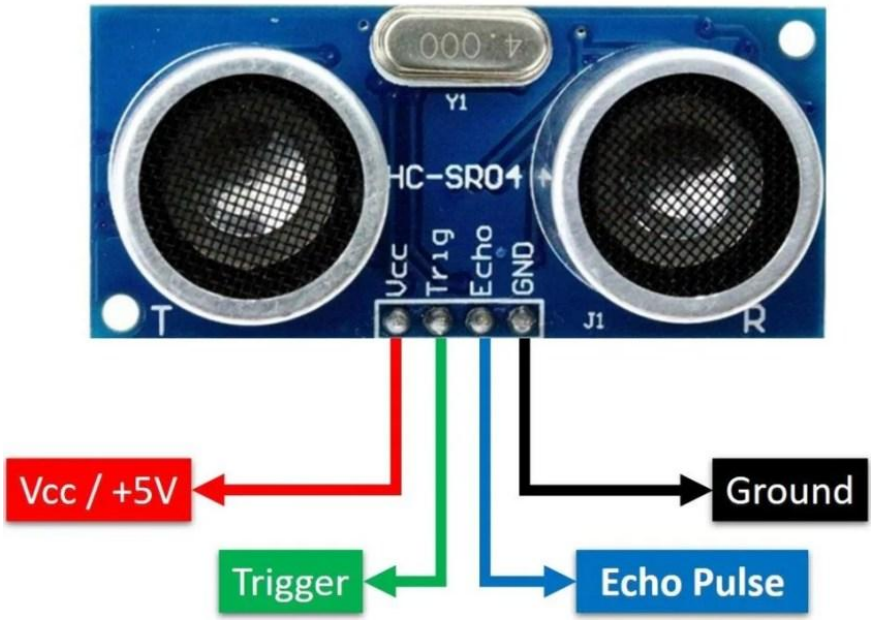
Entire Module Schematic



설계 과정

- **Ultrasonic sensor + UART**

  - 센서 스펙



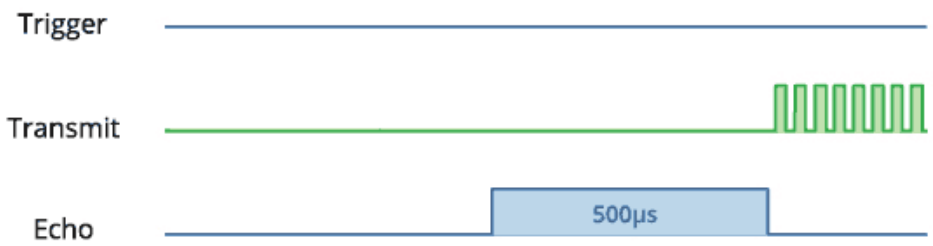
HC-SR04 초음파 센서	
기능	거리 측정
측정 방식	초음파를 이용하여 거리를 측정

구분	스펙
동작 전압	DC 5V
동작 전류	15mA
동작 주파수	40KHz
최대 측정 거리	4m
최소 측정 거리	2cm
정확도	±3mm
측정 각도	15 °
트리거 입력 신호	10us TTL 펄스
크기	40 X 20 X 15 mm

설계 과정

Ultrasonic sensor + UART

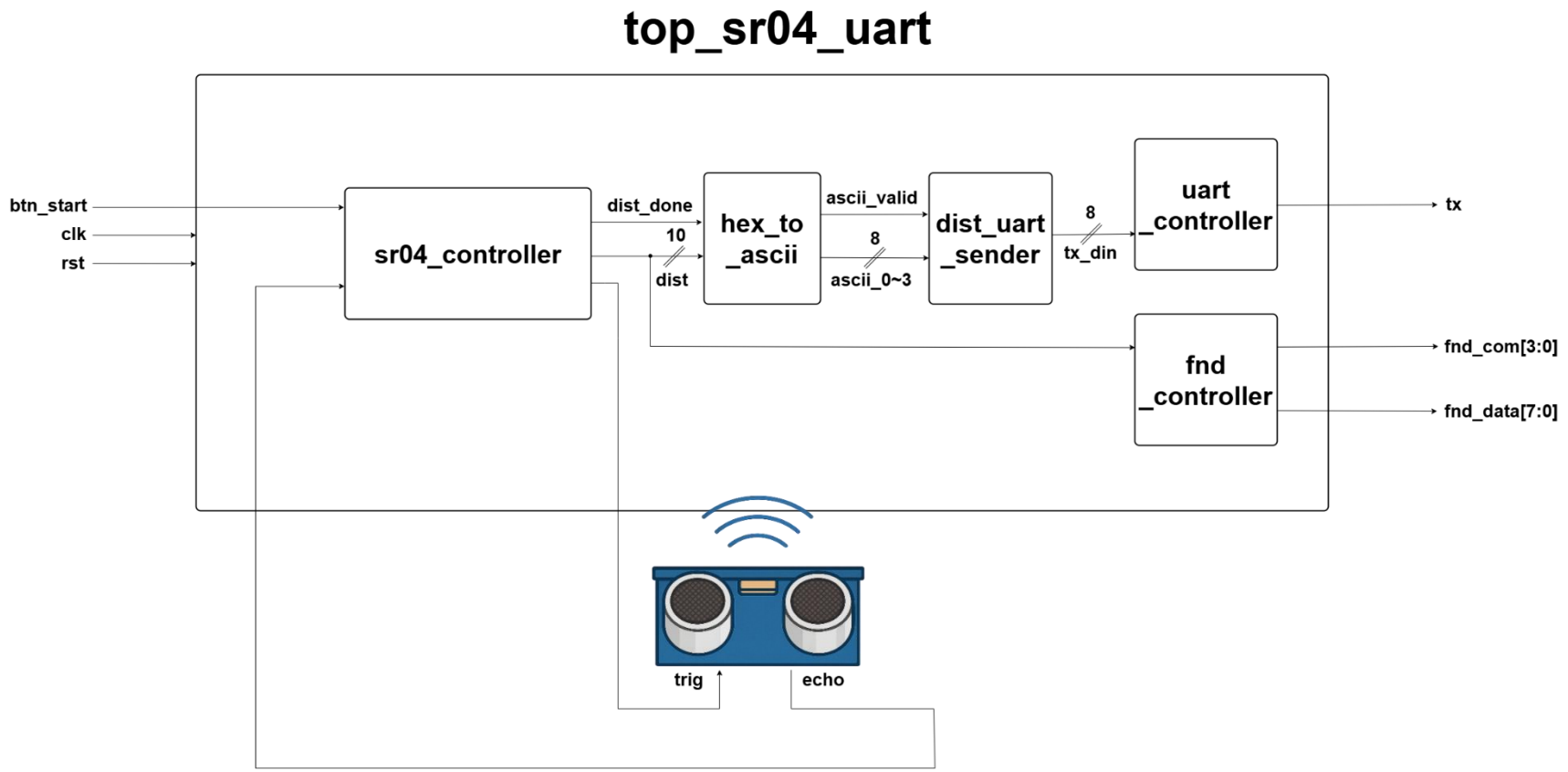
- 센서 동작 과정



- Trigger 핀에 10µs 이상의 HIGH 신호 입력
- Trigger 신호를 감지한 센서는 8번의 40kHz 초음파를 발사
- 초음파가 수신기에 감지되는 순간까지 Echo 핀이 High 상태 유지

설계 과정

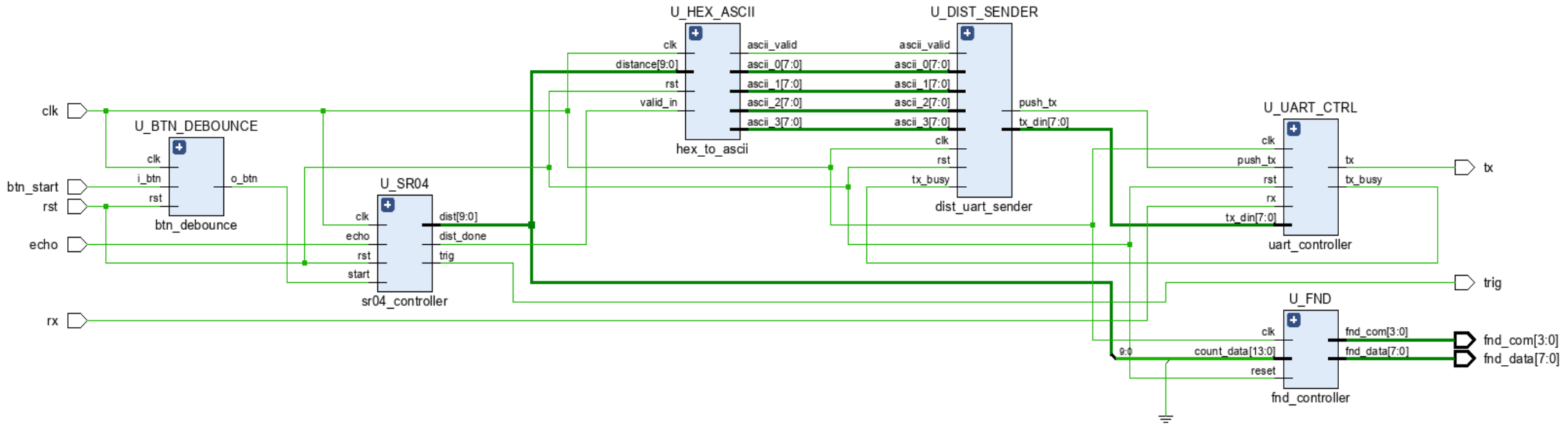
- **Ultrasonic sensor + UART**
  - **Block Diagram (Top Module)**



설계 과정

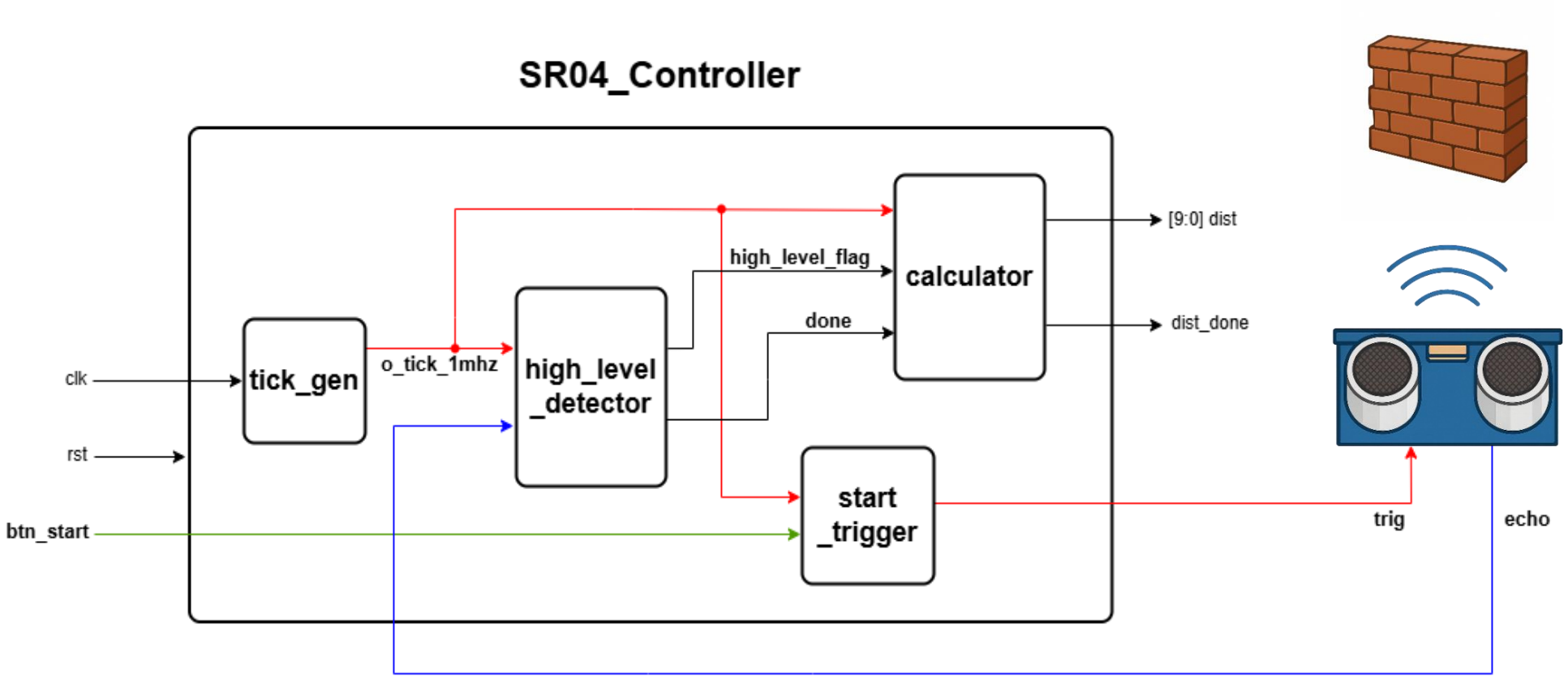
- **Ultrasonic sensor + UART**

  - **Schematic (Top Module)**



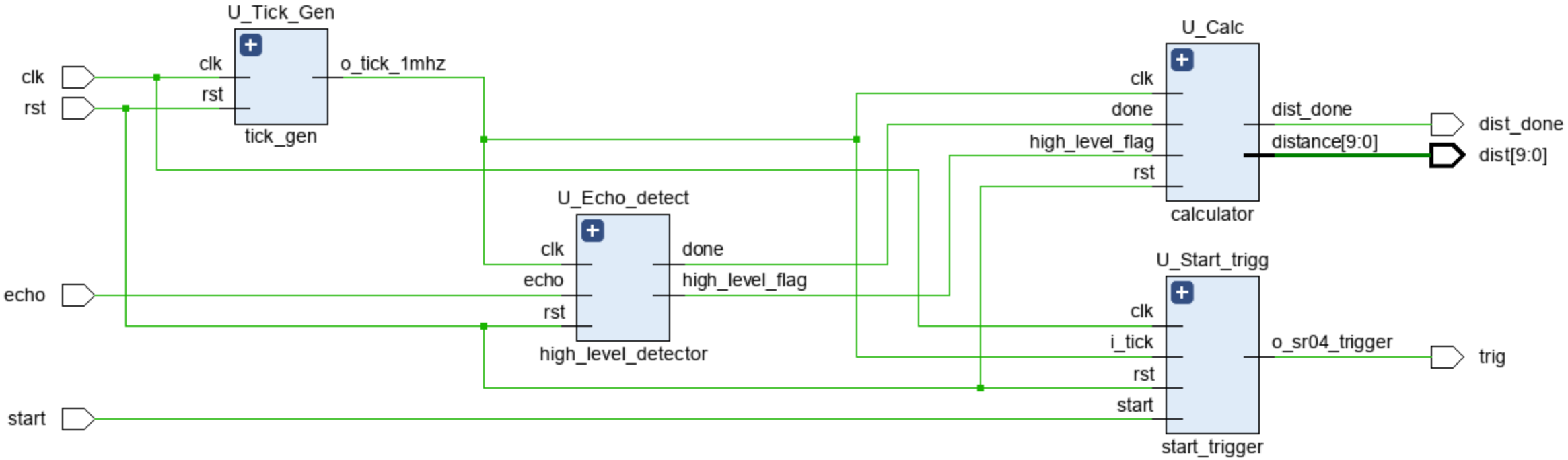
설계 과정

- **Ultrasonic sensor + UART**
  - **Block Diagram (SR04 Controller)**



설계 과정

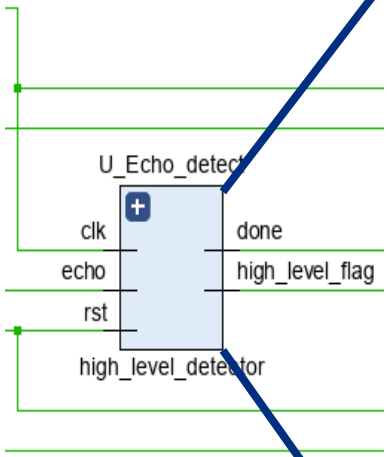
- **Ultrasonic sensor + UART**
  - **Schematic (SR04 Controller)**



설계 과정

■ Ultrasonic sensor + UART

- Main code – high level detector




```
module high_level_detector (
    input clk,
    input rst,
    input echo,
    output reg high_level_flag,
    output reg done
);
    reg echo_d;

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            high_level_flag <= 0;
            done <= 0;
            echo_d <= 0;
        end else begin
            echo_d <= echo;
            if (echo && !echo_d) begin
                high_level_flag <= 1;
                done <= 0;
            end
            else if (!echo && echo_d) begin
                high_level_flag <= 0;
                done <= 1;
            end else begin
                done <= 0;
            end
        end
    end
endmodule
```

1. echo 입력 신호

   → 1클럭 딜레이된 echo_d와 비교

2. Rising Edge 검출

   → echo = 1 & echo_d = 0

   → high_level_flag <= 1

3. Falling Edge 검출

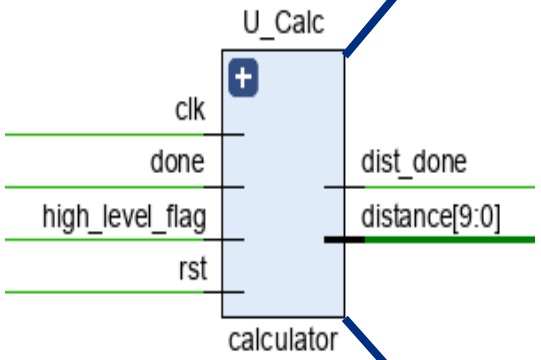
   → echo = 0 & echo_d = 1

   → done <= 1

4. 그 외 구간에서는 done <= 0

설계 과정

- **Ultrasonic sensor + UART**
  - **Main code – calculator**




```
module calculator (
    input clk,
    input rst,
    input high_level_flag,
    input done,
    output reg [9:0] distance,
    output reg dist_done
);
    reg [15:0] count;

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            count <= 0;
            distance <= 0;
            dist_done <= 0;
        end else begin
            if (high_level_flag) begin
                count <= count + 1;
                dist_done <= 0;
            end else if (done) begin
                distance <= count / 58;
                dist_done <= 1;
                count <= 0;
            end else begin
                dist_done <= 0;
            end
        end
    end
endmodule
```

1. high_level_flag = 1 동안 count 증가

2. done 신호 발생 시:

   → 거리 계산: distance <= count / 58

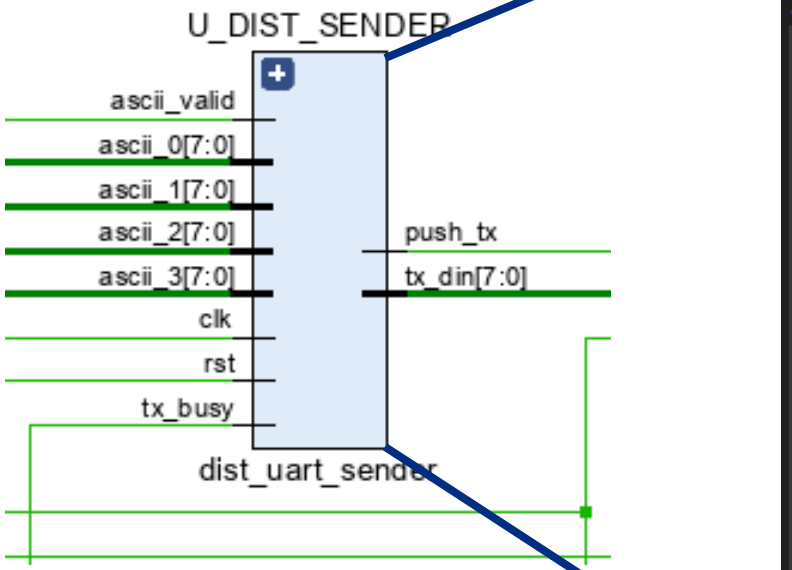
   → dist_done <= 1 (결과 유효 신호)

   → count 초기화

설계 과정

- **Ultrasonic sensor + UART**

  - **Main code – dist uart sender**

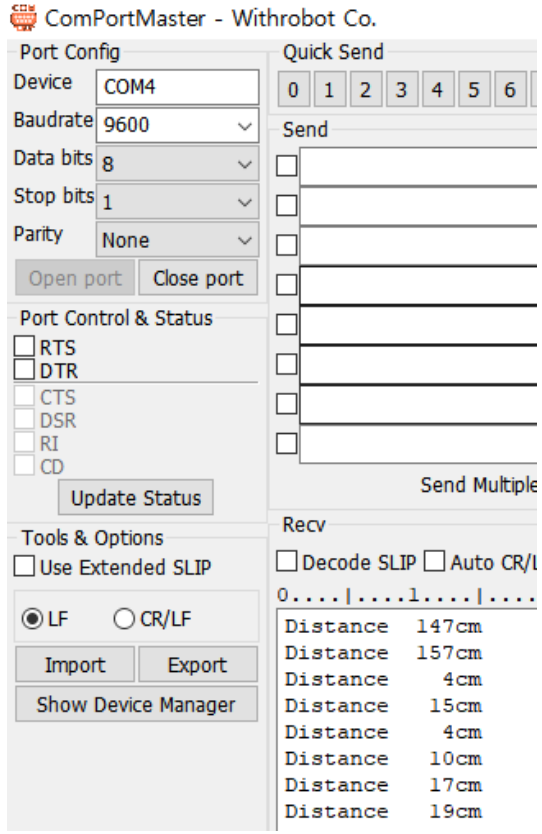



```
reg [7:0] msg[0:16];

initial begin
    msg[0]  = "D";
    msg[1]  = "i";
    msg[2]  = "s";
    msg[3]  = "t";
    msg[4]  = "a";
    msg[5]  = "n";
    msg[6]  = "c";
    msg[7]  = "e";
    msg[8]  = " ";

    msg[13] = "c";
    msg[14] = "m";
    msg[15] = 8'h0D;
    msg[16] = 8'h0A;
end
```

Distance XXXXcm\r\n"

(총 17글자)

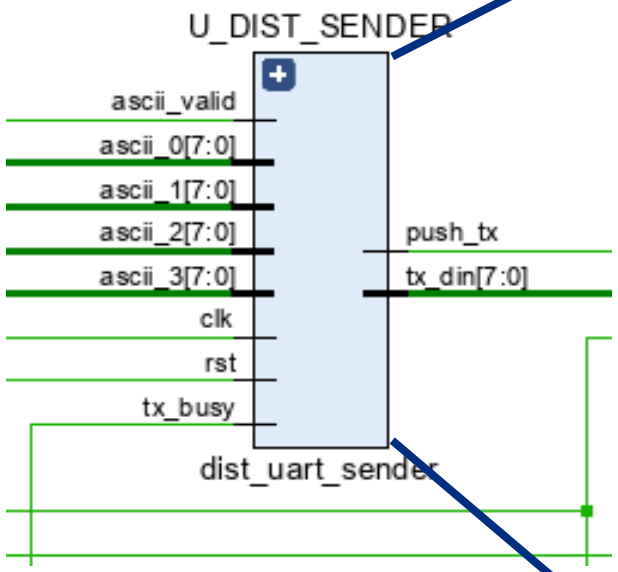
msg[9~12] = ascii_3~0



설계 과정

Ultrasonic sensor + UART

- Main code – dist uart sender




```
reg [4:0] index;
reg sending;

always @(posedge clk or posedge rst) begin
    if (rst) begin
        index   <= 0;
        tx_din  <= 0;
        push_tx <= 0;
        sending <= 0;
    end else begin
        push_tx <= 0;

        if (ascii_valid && !sending) begin
            // 거리 데이터
            sending <= 1;
            index   <= 0;
        end else if (sending && !tx_busy) begin
            tx_din  <= msg[index];
            push_tx <= 1;
            index   <= index + 1;
            if (index == 16) begin  // 마지막 문자까지 전송 완료
                sending <= 0;
            end
        end
    end
end
```

FSM 구조 없이 간단한
상태 레지스터(sending)로
"전송 중/대기" 관리

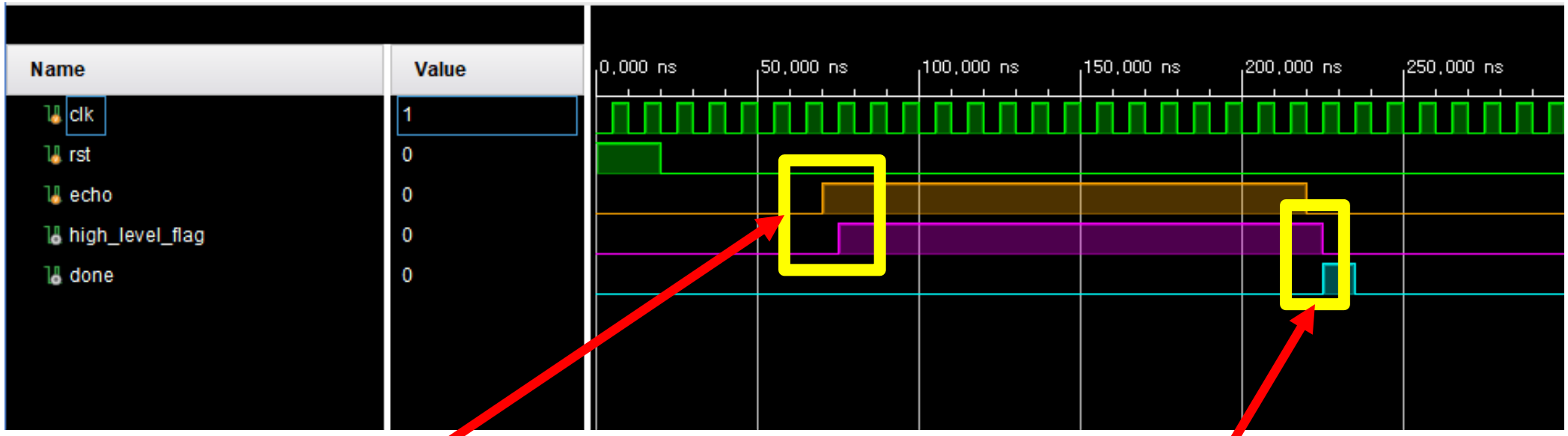
```
if (ascii_3 == 8'h30) begin
    msg[9] = 8'h20;
end else begin
    msg[9] = ascii_3;
end
if (ascii_2 == 8'h30 && ascii_3 == 8'h30) begin
    msg[10] = 8'h20;
end else begin
    msg[10] = ascii_2;
end
if (ascii_1 == 8'h30 && ascii_3 == 8'h30 && ascii_2 == 8'h30) begin
    msg[11] = 8'h20;
end else begin
    msg[11] = ascii_1;
end
msg[12] <= ascii_0;
```

ascii_3,2,1 에서 0인 경우,
공백(스페이스)로 출력
→ 선행 0 제거 (ex: "Distance 45cm")

기능 검증 및 설명

- Ultrasonic sensor + UART

  - Simulation – high level detection

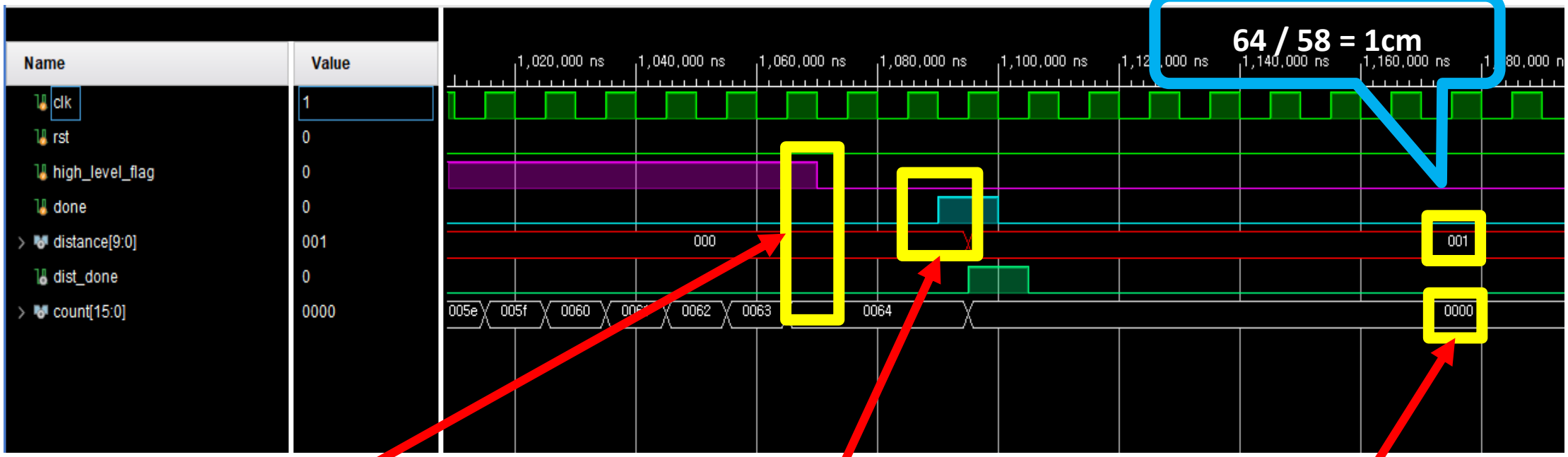


1. echo ↑(비동기)
2. clk이 오면 echo_d 업데이트
3. high_level_flag 업데이트
4. high_level_flag & done 동시 업데이트

기능 검증 및 설명

- Ultrasonic sensor + UART

  - Simulation – calculator



nextchip

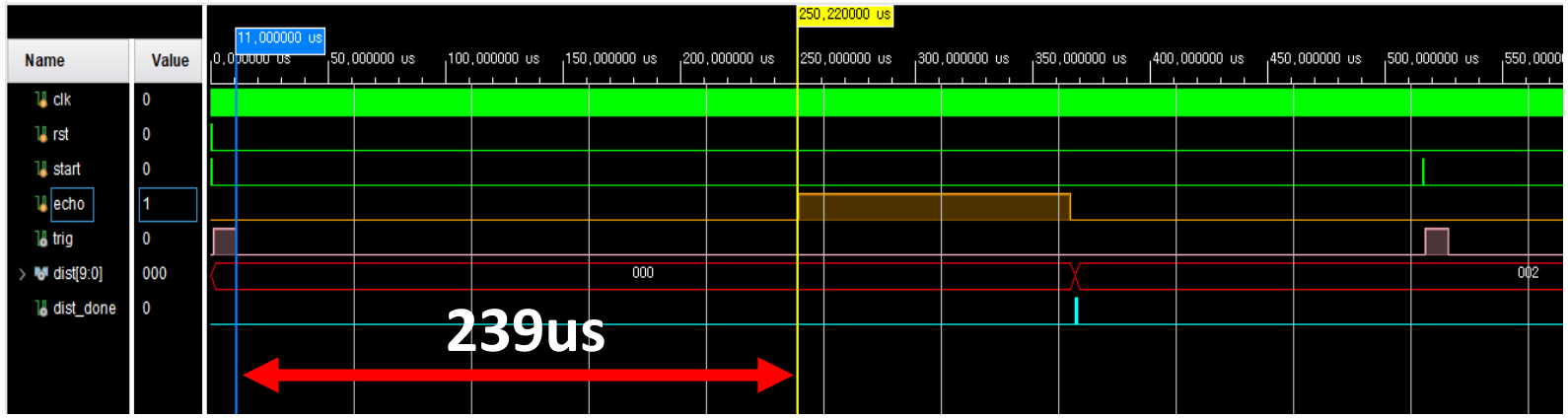
KFIA 한국팹리스산업협회 Korea Fabless Industry Association

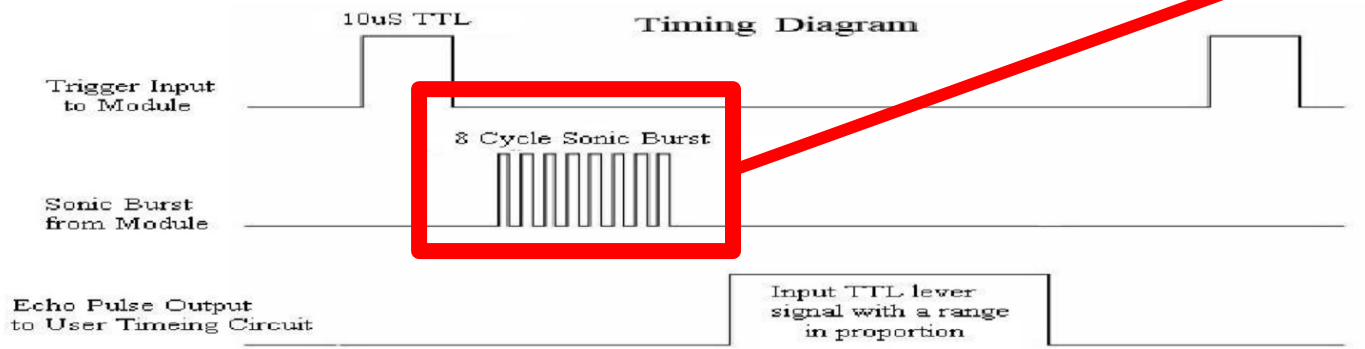
대한상공회의소 서울기술교육센터

기능 검증 및 설명

Ultrasonic sensor + UART

- Simulation – SR04 Controller (1)

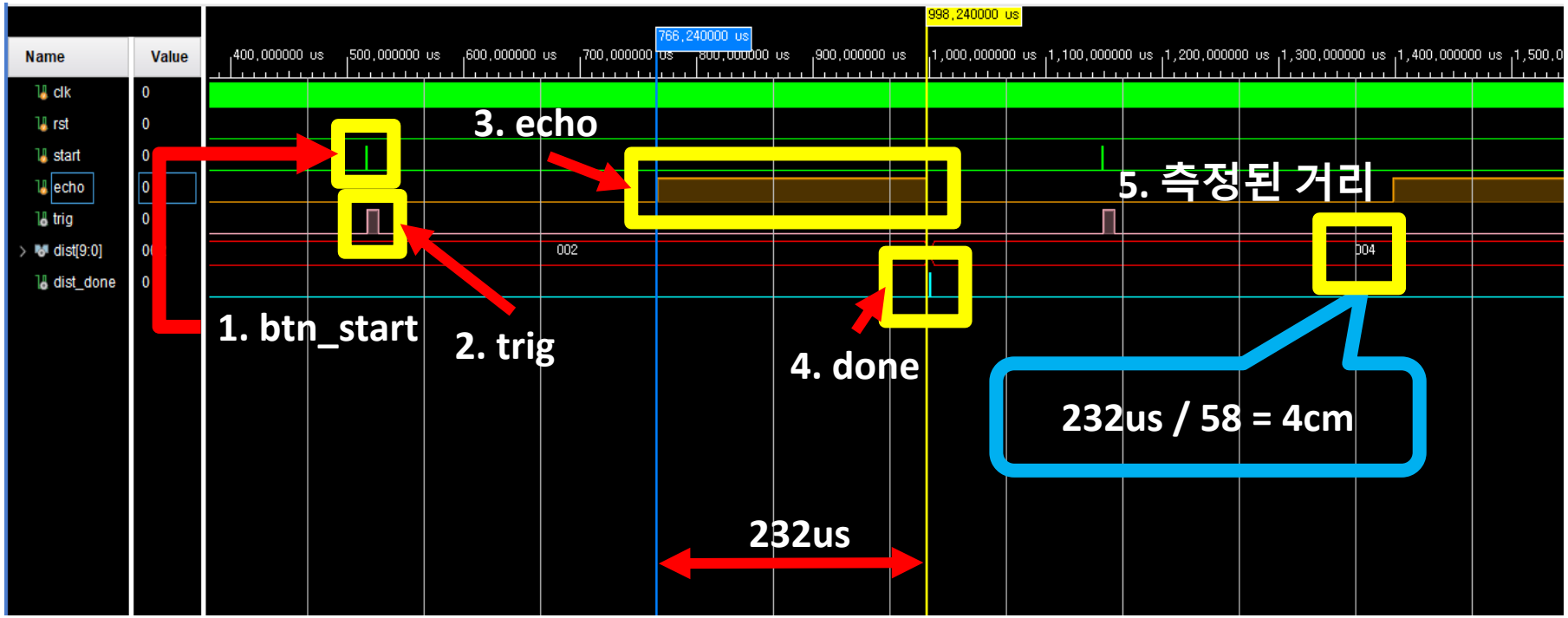




40kHz → 25us period
→ 8cycle
= 8 x 25us
= 200us

기능 검증 및 설명

- **Ultrasonic sensor + UART**
  - **Simulation – SR04 Controller (2)**

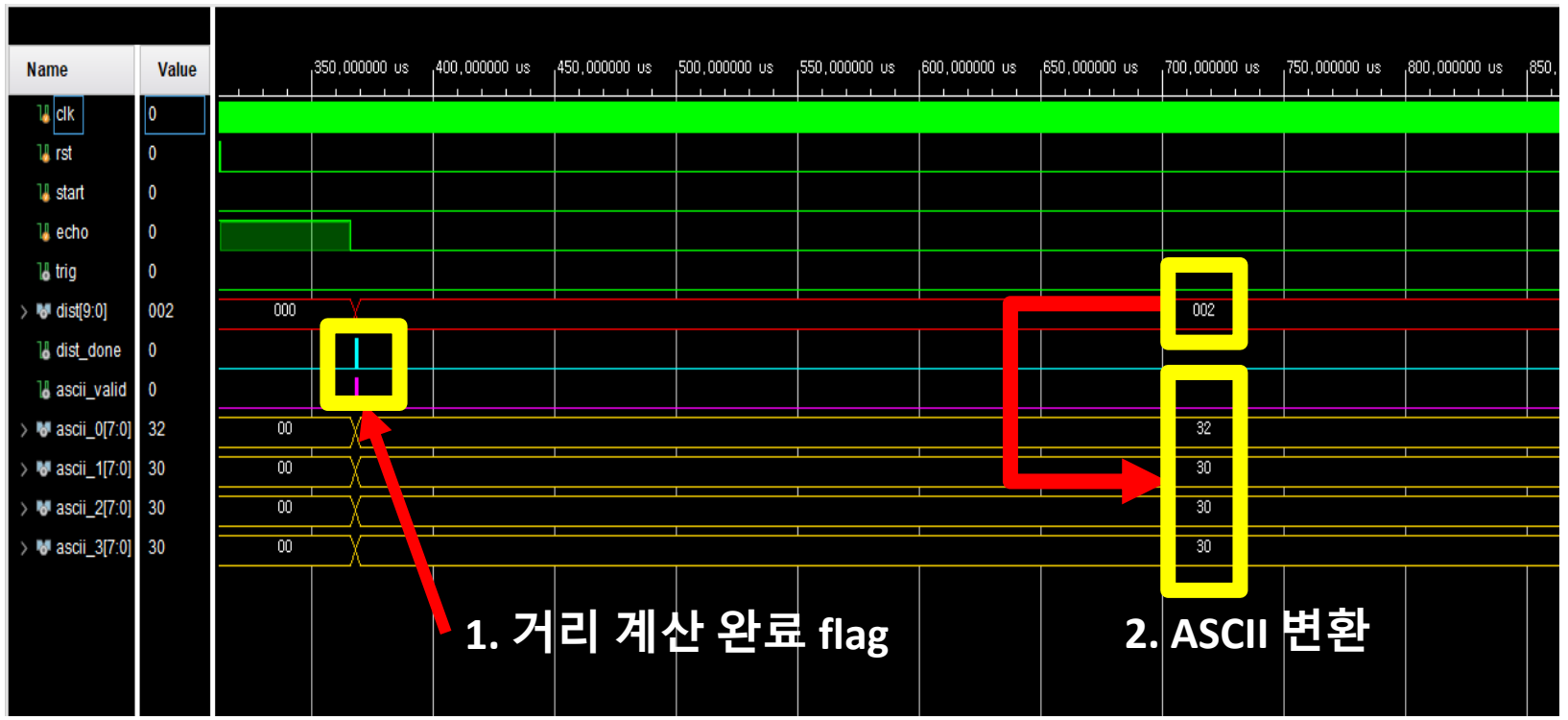


SR04 동작 FLOW

1. btn_start
2. trig
3. echo
4. dist_done
5. 측정된 거리

기능 검증 및 설명

- **Ultrasonic sensor + UART**
  - **Simulation – hex to ascii**

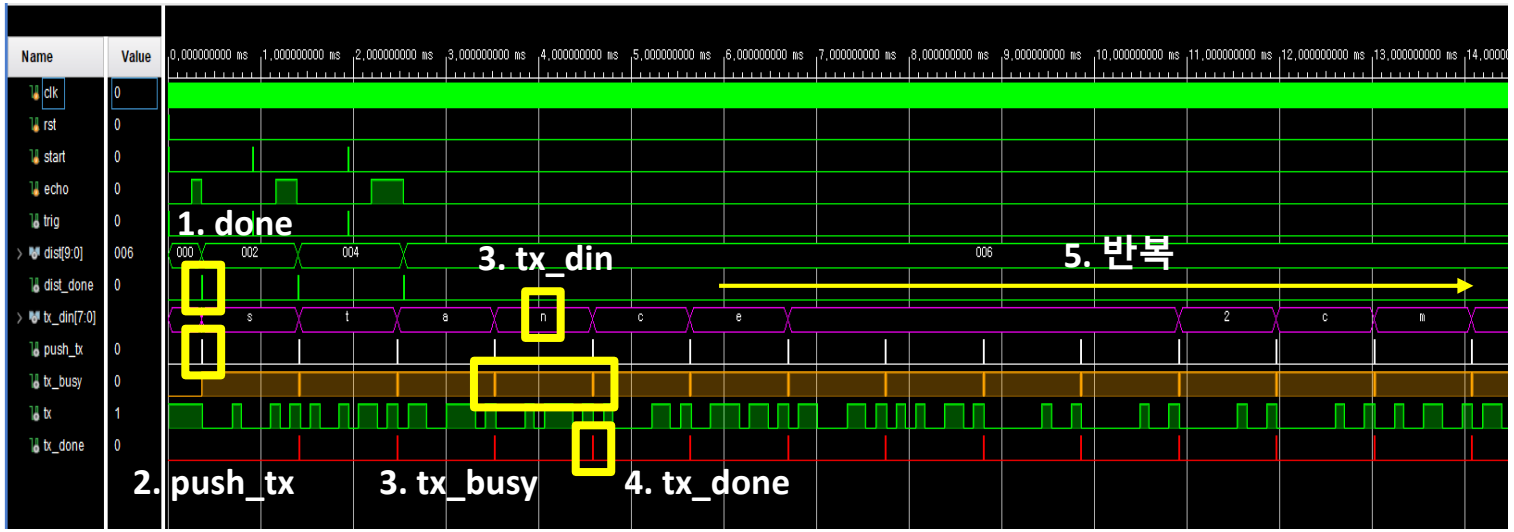


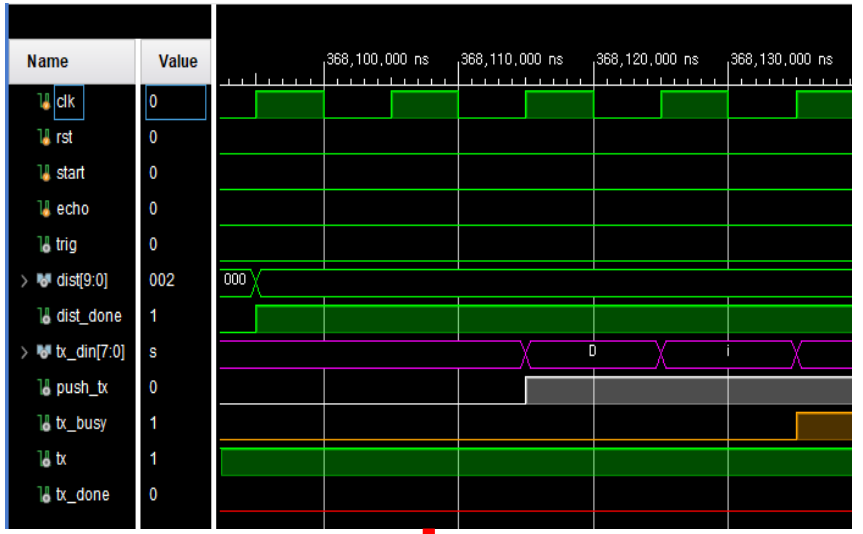
자리	값	ASCII 값
ascii_0	0x32	2
ascii_1	0x30	0
ascii_2	0x30	0
ascii_3	0x30	0

기능 검증 및 설명

- **Ultrasonic sensor + UART**

  - **Simulation – dist uart sender + uart controller**





UART 동작 FLOW

1.done → 2.push_tx → 3.tx_busy, tx_din → 4.tx_done → 5.반복

msg[] 버퍼에 있는 글자들이
UART TX를 통해 출력
→ Distance 2

nextchip

KFIA 한국팹리스산업협회 Korea Fabless Industry Association

대한상공회의소 서울기술교육센터

트러블슈팅과 해결방안

- **Ultrasonic sensor + UART**

  - Trouble Shooting (1)

개선 전 코드

```
module high_level_detector (
    input clk,
    input rst,
    input echo,
    output reg high_level_flag,
    output reg done
);

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            high_level_flag <= 0;
            done <= 0;
        end else begin
            high_level_flag <= echo;
            done <= (echo == 0);
        end
    end

endmodule
```

high_level_flag → echo 값 그대로 복사
echo가 High면 high_level_flag = 1
echo가 Low면 high_level_flag = 0
단순 복사 → 엣지(Edge) 감지가 아님

done → echo Low일 때 계속 1 유지
echo가 떨어진 순간 "1클럭만 done = 1" 되어야 하지만
echo가 Low 상태인 동안 계속 done = 1 유지됨

트러블슈팅과 해결방안

- **Ultrasonic sensor + UART**

  - **Trouble Shooting (1)**

개선 후 코드

```
module high_level_detector (
    input clk,
    input rst,
    input echo,
    output reg high_level_flag,
    output reg done
);
    reg echo_d;

    always @(posedge clk or posedge rst) begin
        if (rst) begin
            high_level_flag <= 0;
            done <= 0;
            echo_d <= 0;
        end else begin
            echo_d <= echo;
            if (echo && !echo_d) begin
                high_level_flag <= 1;
                done <= 0;
            end
            else if (!echo && echo_d) begin
                high_level_flag <= 0;
                done <= 1;
            end else begin
                done <= 0;
            end
        end
    end
endmodule
```

과거 echo 값을 저장 (echo_d) → 엣지 검출 가능

상승 엣지 (echo ↑) → high_level_flag = 1 시작

하강 엣지 (echo ↓) → done = 1 pulse 발생

calculator count 정상 증가 → 정확한 거리 측정 가능

트러블슈팅과 해결방안

- **Ultrasonic sensor + UART**

  - **Trouble Shooting (2)**

개선 전 코드

```
else if (sending) begin
    if (!tx_busy) begin
        tx_din  <= msg[index];
        push_tx <= 1;
        index   <= index + 1;

        if (index == 16) begin
            sending <= 0;
        end
    end else begin
        push_tx <= 1;
    end
end
```

Sending == 1이면 무조건 실행
→ 내부에서 if(!tx_busy) 분기 처리

tx_busy == 0 일 때는 정상
하지만 tx_busy == 1일 때도
else_begin → push_tx <= 1 발생

tx_busy == 1에서도
계속 push_tx 유지되는 문제 발생

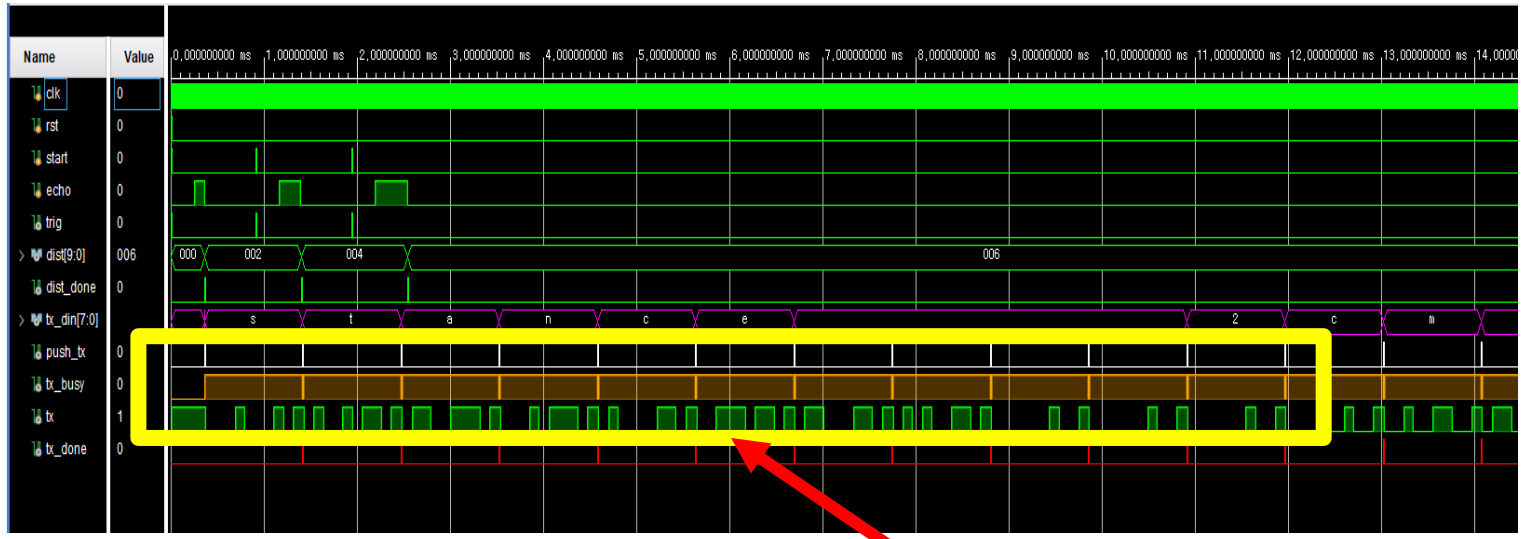
트러블슈팅과 해결방안

- **Ultrasonic sensor + UART**

  - **Trouble Shooting (2)**

개선 후 코드

```
end else if (sending && !tx_busy) begin
    tx_din  <= msg[index];
    push_tx <= 1;
    index   <= index + 1;

    if (index == 16) begin
        sending <= 0;
    end
end
```

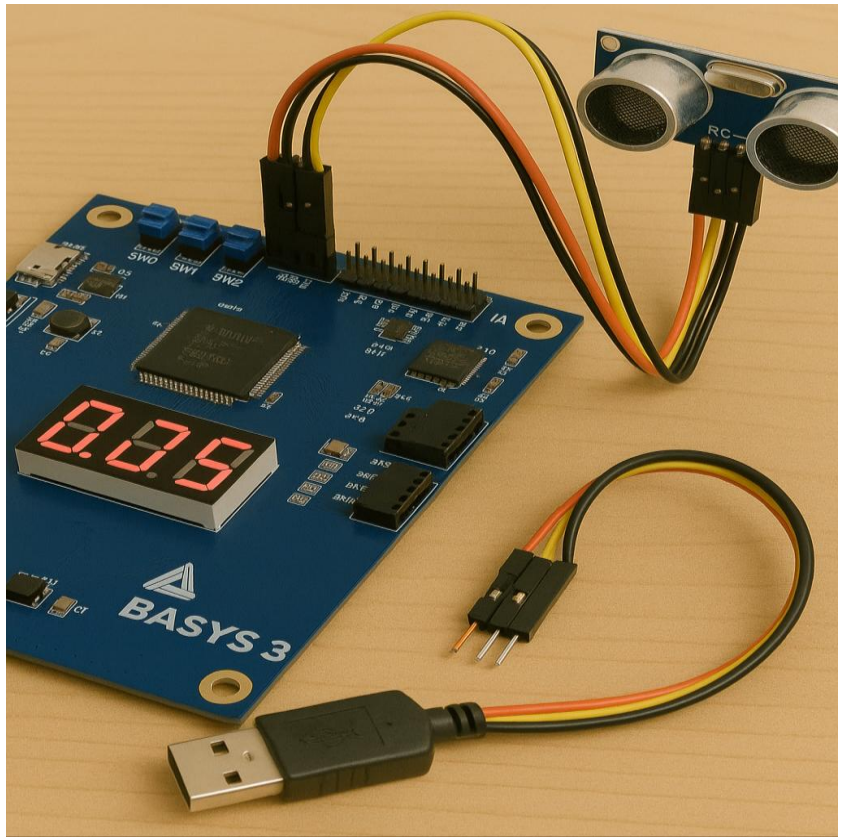


else if (sending && !tx_busy)
조건문 자체에서
push_tx 발생 여부 제어
→ tx_busy == 1일 때 push_tx 발생 x

tx_busy == 0에서 push_tx 발생

고찰

Ultrasonic sensor + UART

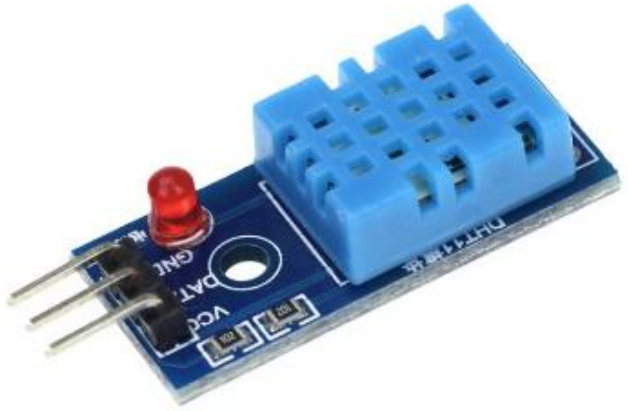


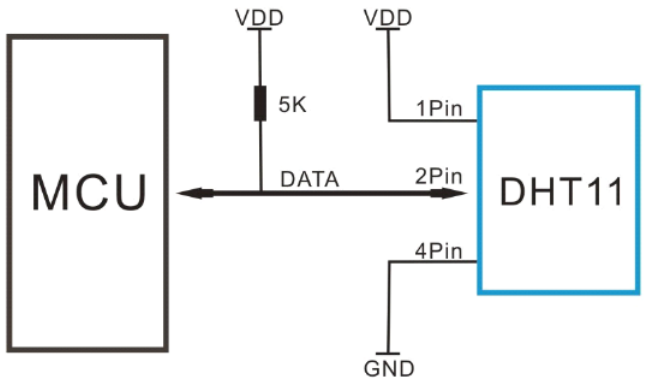
거리 측정 결과를 UART 통신을 통해 외부로 출력하는 과정을 구현하면서 디지털 회로와 시리얼 통신 시스템의 연계를 경험할 수 있었습니다. 특히 Verilog 기반의 UART 송신 회로 설계와 함께, ASCII 변환 후 문자열을 구성하여 전송하는 과정에서 UART 프로토콜의 데이터 처리 방식과 타이밍 동작을 이해하게 되었습니다.

설계 과정

■ DHT11(Hm/T) + UART

- 센서 스펙





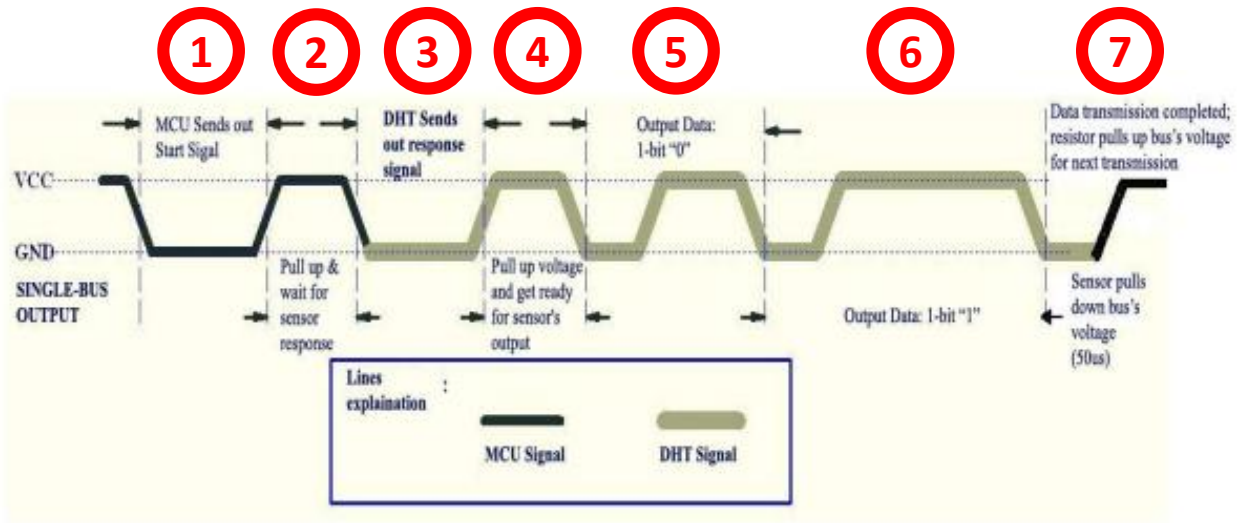
DHT11 온도/습도 센서	
기능	온도, 습도 측정
측정 방식	ADC를 통한 디지털 신호 전송

구분	스펙
측정 범위	온도 : 0 ~ 50°C
	습도 : 20~90%
정확도	온도 : ±2°C
	습도 : ±5%
Resolution	1

설계 과정

■ DHT11(Hm/T) + UART

- DHT11 통신 프로토콜 FLOW



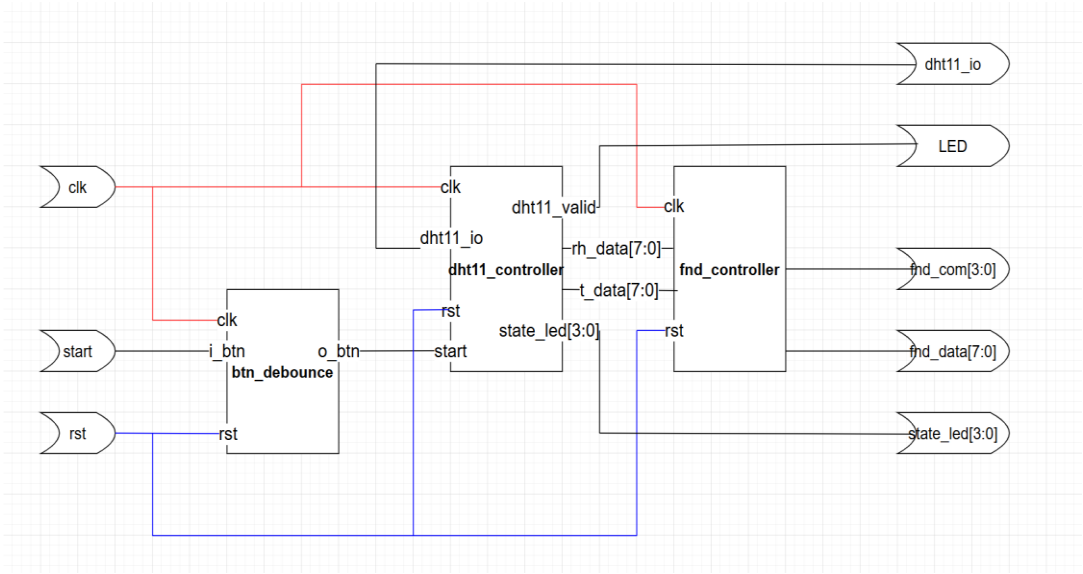
단계	신호 상태
1. Start Signal	MCU가 라인을 LOW로 약 18ms 유지
2. Pull-up 후 대기	MCU가 라인을 HIGH로 전환
3. Response Signal	LOW 80us + HIGH 80us
4. 데이터 전송 시작	각 비트당 패턴 반복
5. 비트 '0' 전송	50us LOW + 26~28us HIGH
6. 비트 '1' 전송	50us LOW + 70us HIGH
7. 종료 처리	마지막에 DHT가 50us LOW 유지 후 라인 해제

데이터 비트	타이밍 구조
'0'	50us LOW + 26~28us HIGH
'1'	50us LOW + 70us HIGH

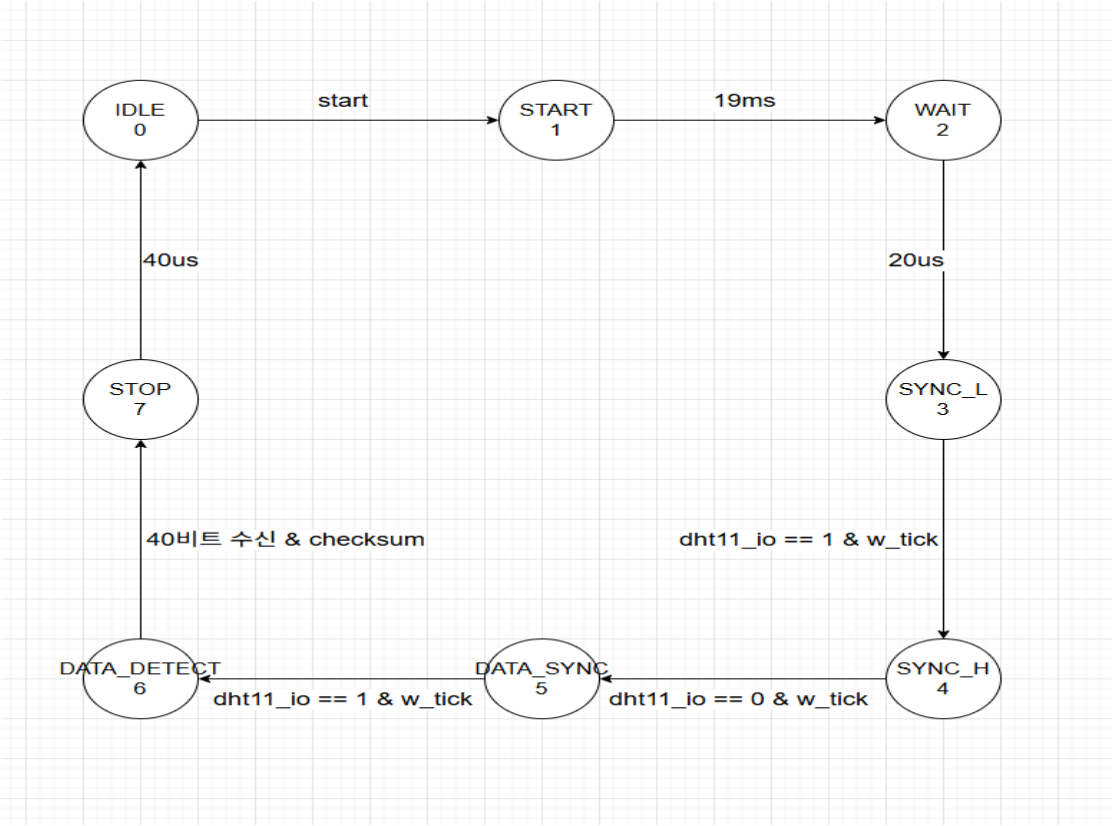
설계 과정

DHT11(Hm/T) + UART

- Block Diagram



- FSM



nextchip

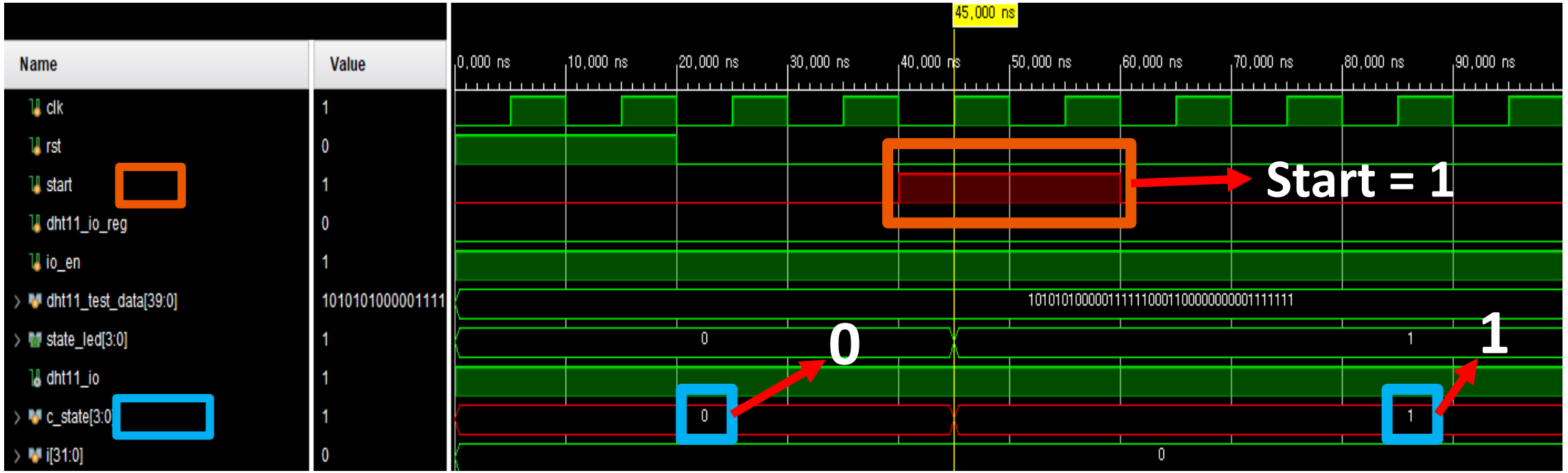
KFIA 한국팹리스산업협회 Korea Fabless Industry Association

대한상공회의소 서울기술교육센터

기능 검증 및 설명

■ DHT11(Hm/T) + UART

- Simulation

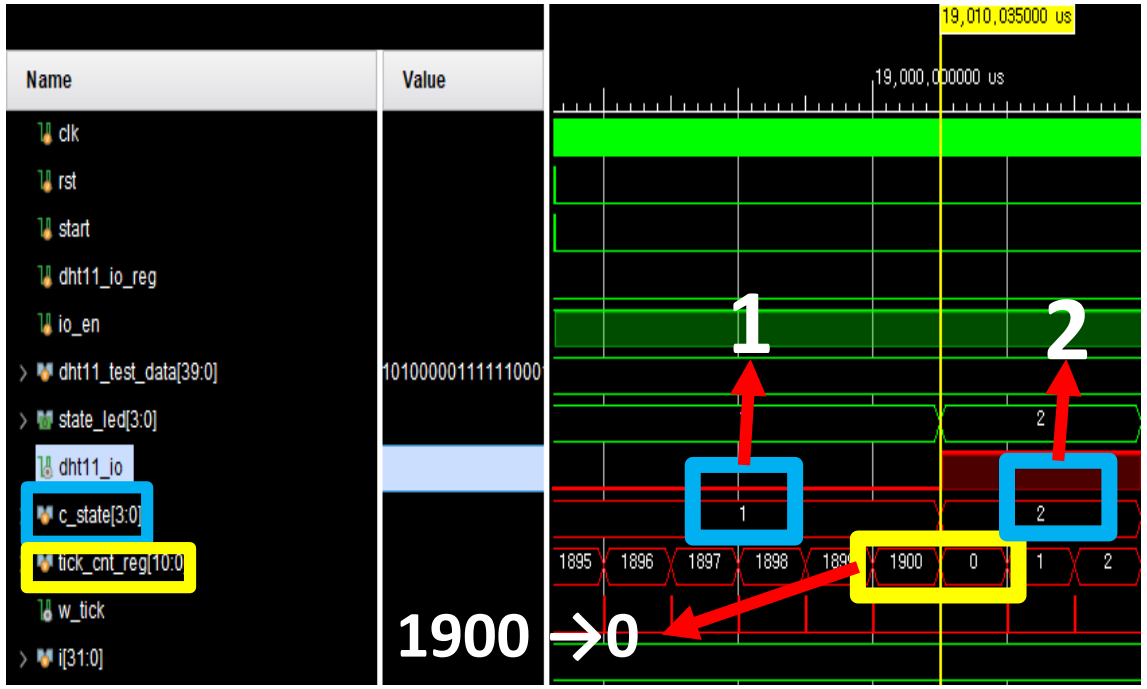


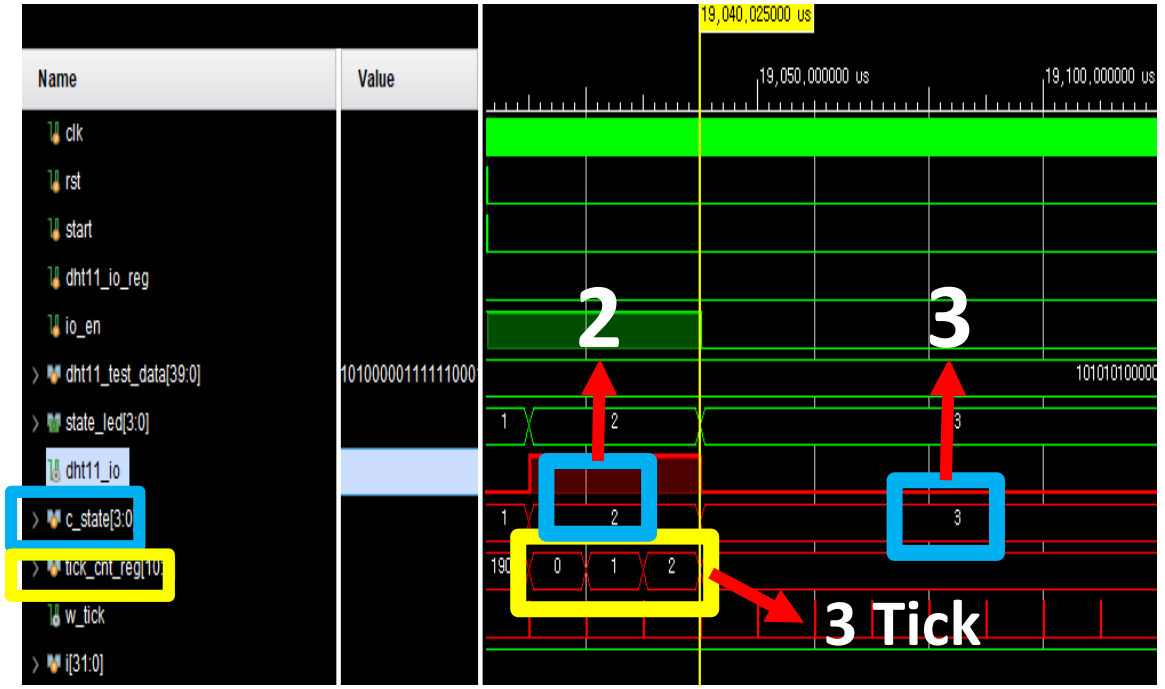
START(0 → 1) 입력될 때, STATE : IDLE (0) → START (1)

기능 검증 및 설명

■ DHT11(Hm/T) + UART

- Simulation





1900 tick(19ms) 들어올 때,

STATE : START (1) → WAIT (2)

3 tick(30µs) 들어올 때,

STATE : WAIT (2) → SYNC_L (3)

nextchip

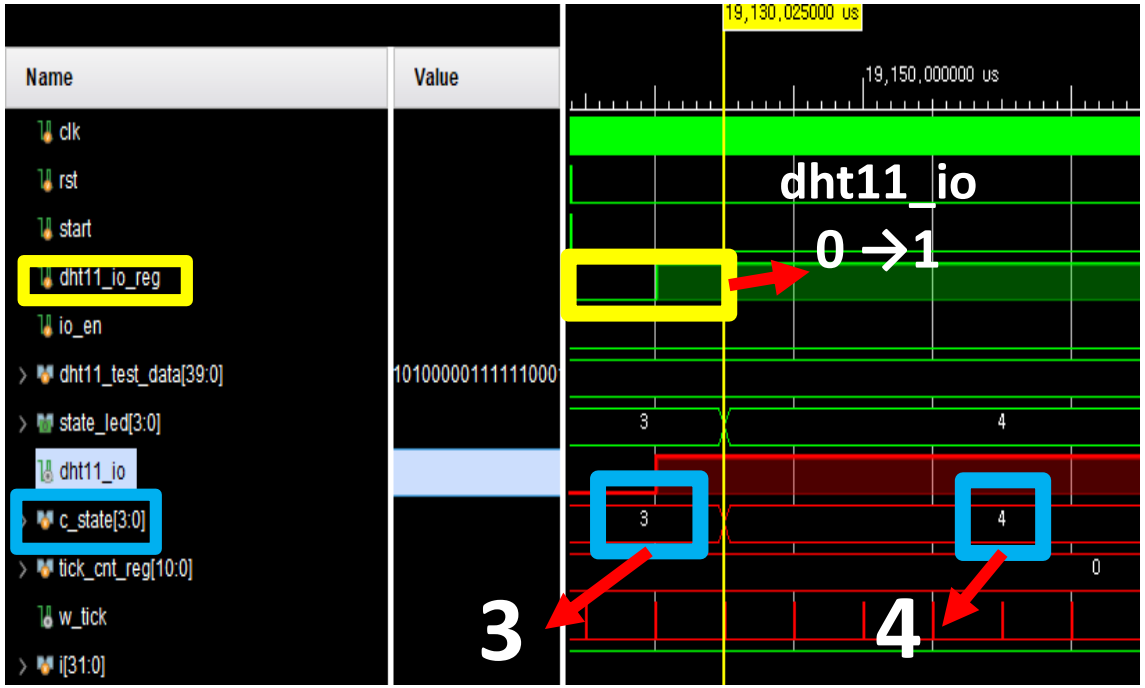
KFIA 한국팹리스산업협회 Korea Fabless Industry Association

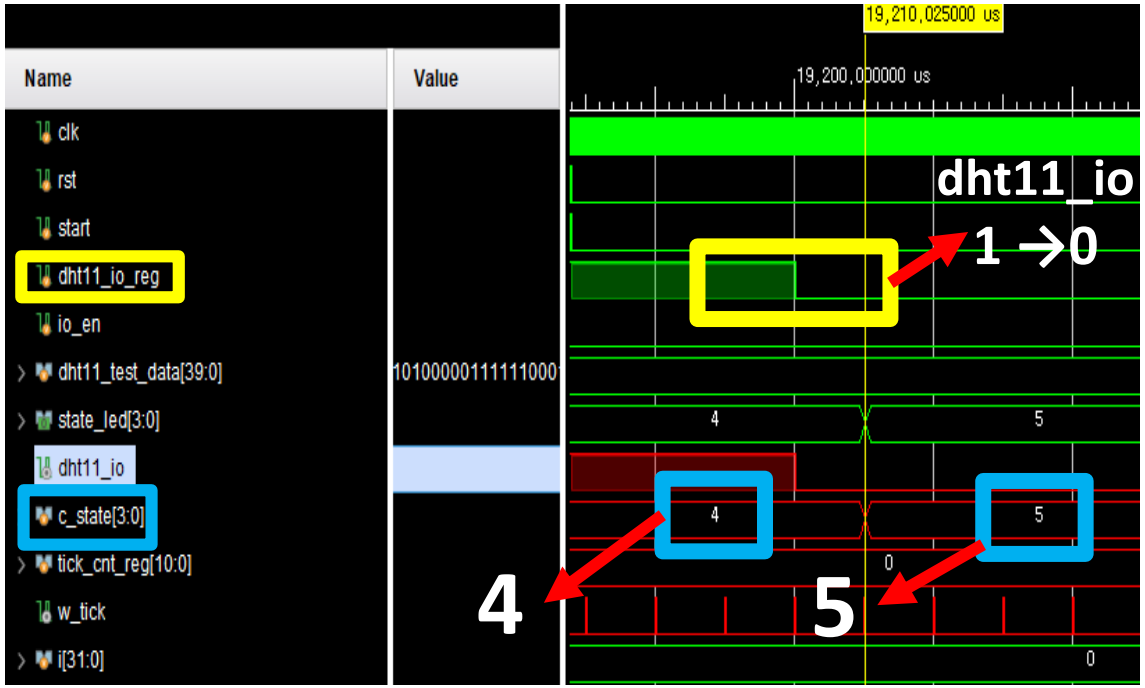
대한상공회의소 서울기술교육센터

기능 검증 및 설명

DHT11(Hm/T) + UART

- Simulation





(dht_io == 1 & tick 발생) In SYNC_L(3),

STATE : SYNC_L (3) → SYNC_H (4)

(dht_io == 0 & tick 발생) In SYNC_H(4),

STATE : SYNC_H (4) → DATA_SYNC (5)

nextchip

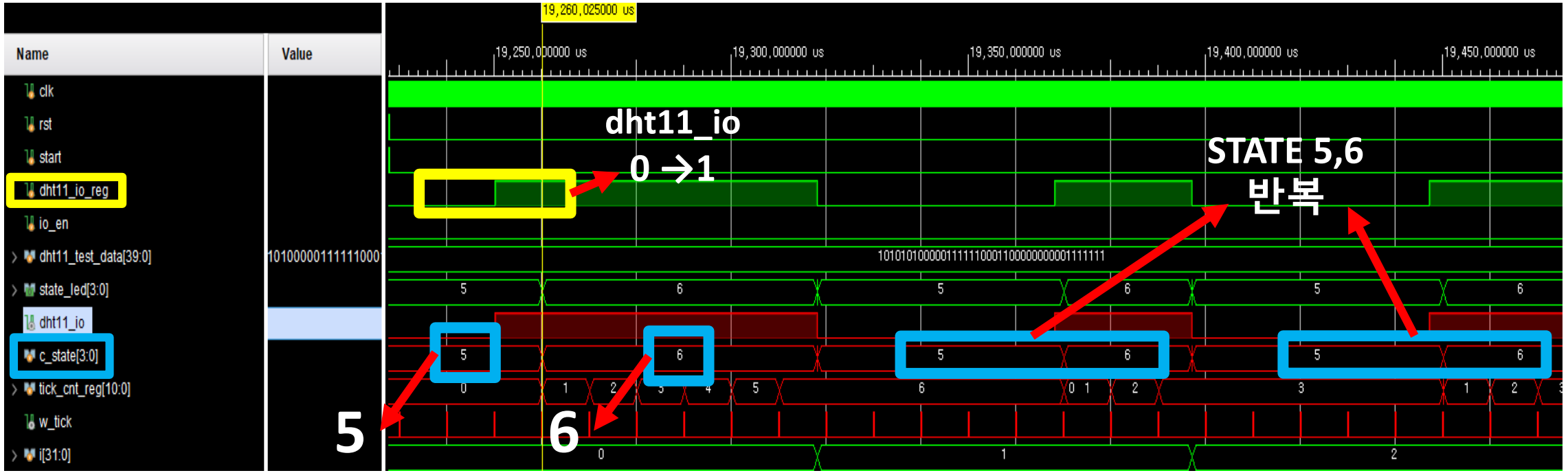
KFIA 한국팹리스산업협회 Korea Fabless Industry Association

대한상공회의소 서울기술교육센터

기능 검증 및 설명

■ DHT11(Hm/T) + UART

- Simulation



(dht_io == 1 & tick 발생) In DATA_SYNC(5),

STATE : DATA_SYNC (5) → DATA_DETECT (6)

Data_cnt_reg == 0 될 때까지,

STATE : DATA_SYNC (5) ↔ DATA_DETECT (6)

nextchip

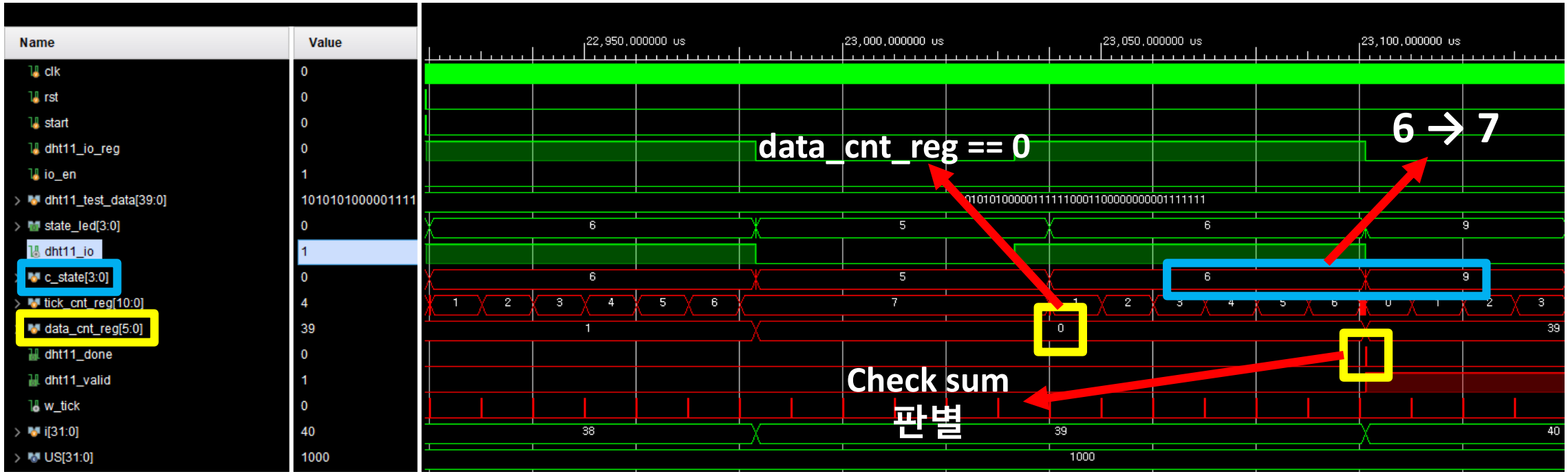
KFIA 한국팹리스산업협회 Korea Fabless Industry Association

대한상공회의소 서울기술교육센터

기능 검증 및 설명

DHT11(Hm/T) + UART

- Simulation



Data_cnt_reg == 0이 되면 Checksum 검사

통과 시, STATE : DATA_DETECT (6) → STOP(7)

nextchip

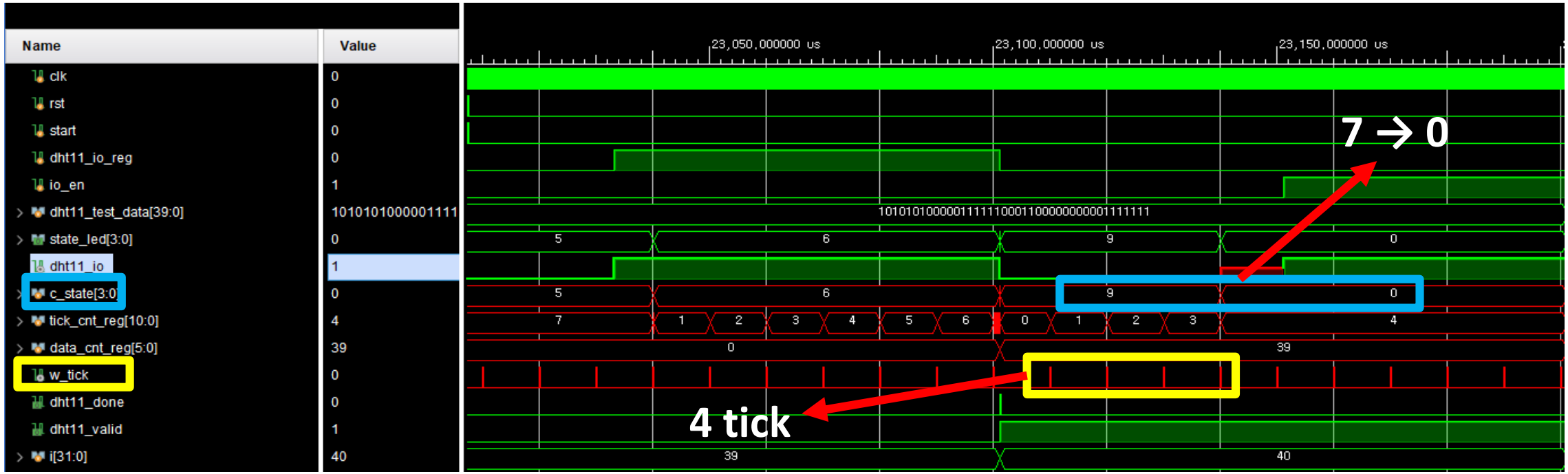
KFIA 한국팹리스산업협회 Korea Fabless Industry Association

대한상공회의소 서울기술교육센터

기능 검증 및 설명

■ DHT11(Hm/T) + UART

- Simulation



4 tick(40us) 들어올 때, STATE : STOP (7) → IDLE (0)

nextchip

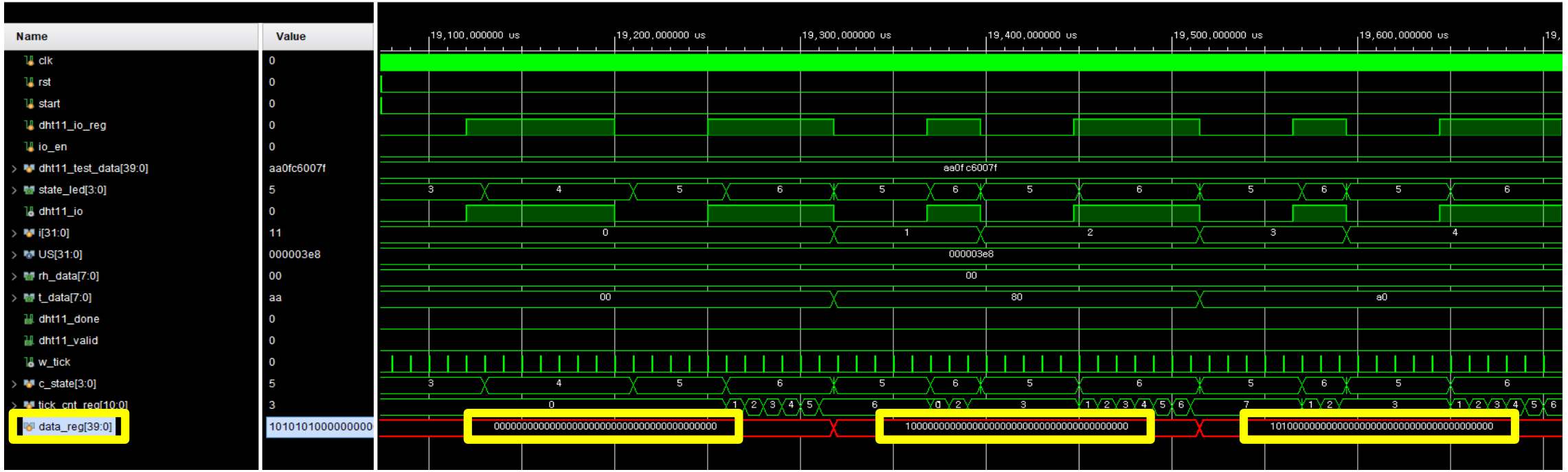
KFIA 한국팹리스산업협회 Korea Fabless Industry Association

대한상공회의소 서울기술교육센터

기능 검증 및 설명

■ DHT11(Hm/T) + UART

- Simulation



data_reg가 점차 0 → 1010…으로 채워지는 것은 센서로부터

들어오는 각 비트를 하나씩 Shift-in하며 저장하고 있는 것을 의미

nextchip

KFIA 한국팹리스산업협회 Korea Fabless Industry Association

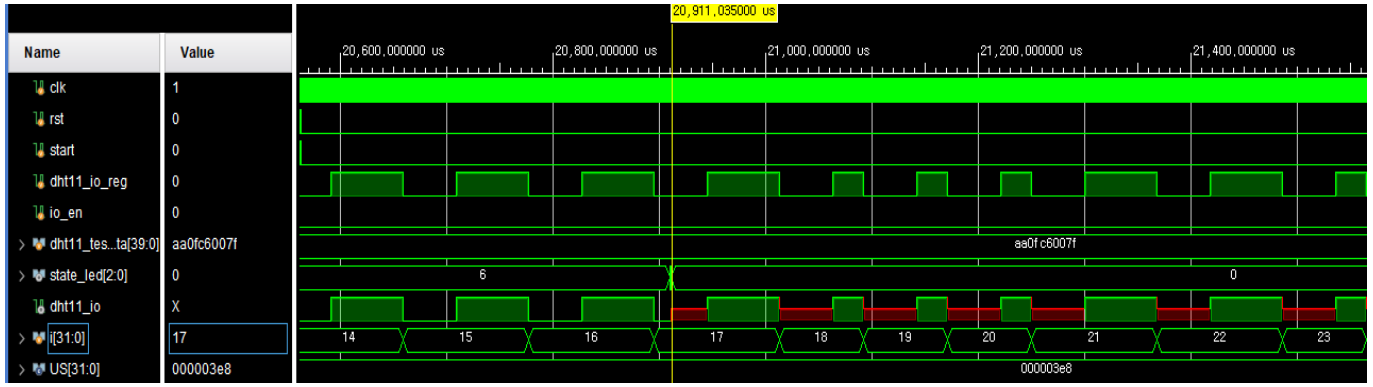
대한상공회의소 서울기술교육센터

기능 검증 및 설명

■ DHT11(Hm/T) + UART

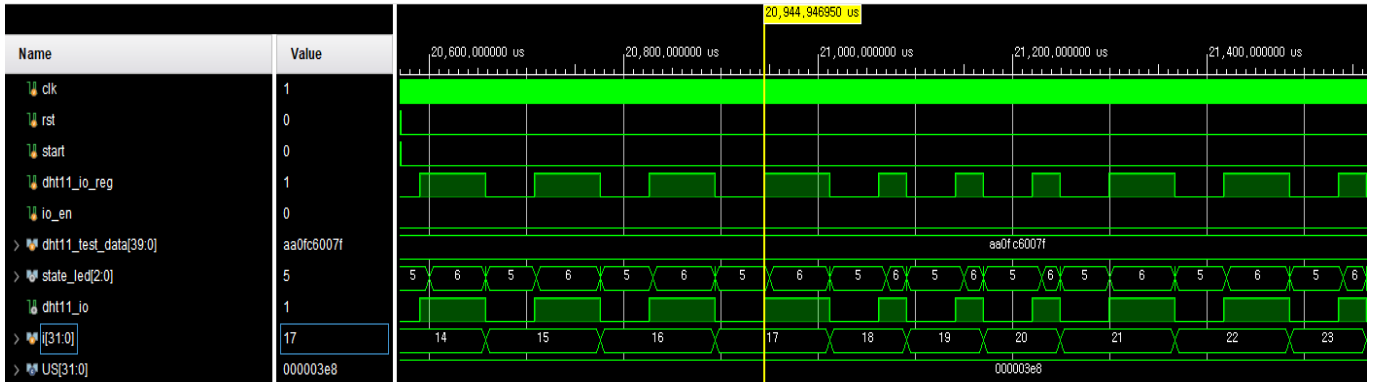
- Trouble Shooting

오류



- FSM이 DATA_DETECT에 멈추며 40 비트 수신 완료 조건 미충족
- Done 신호 출력 실패, 데이터 수신 중단

정상



- DATA_DETECT 상태를 분리하여 판단 및 저장을 안정적으로 수행
- 비트 카운트 진행, 정상적인 done 출력 및 데이터 수신 완료

트러블슈팅과 해결방안

■ DHT11(Hm/T) + UART

- Trouble Shooting solution

```
DATA_DETECT: begin //  6
    if (dht11_io & tick_edge_r) begin
        tick_cnt_next = tick_cnt_reg + 1;
    end
    if (dht11_io_f) begin
        if (tick_cnt_reg >= 5)
            data_next[data_cnt_reg] = 1;
        else
            data_next[data_cnt_reg] = 0;

        if (data_cnt_reg == 0) begin
            tick_cnt_next = 0;
            data_cnt_next = 39;
            dht11_done_next = 1;

            if (data_reg[39:32] + data_reg[31:24] +
                data_reg[23:16] + data_reg[15:8] == data_reg[7:0])
                dht11_valid_next = 1;
            else
                dht11_valid_next = 0;
                n_state = STOP;
        end else begin
            tick_cnt_next = 0;
            data_cnt_next = data_cnt_reg - 1;
        end
    end
end
```

```
DATA_DETECT: begin // 6
        tick_cnt_next = tick_cnt_reg + 1;
    end
    if (dht11_io_f) begin
        n_state = DATA_DETECT_f;
    end

VALID: begin
2] + data_reg[31:24] + data_reg[23:16] + data_reg[15:8] == data_reg[7:0]) begin
        dht11_valid_next = 1;
    end else begin
        dht11_valid_next = 0;
    end
    n_state = STOP;
DATA_DETECT_f: beg
>= 5) begin
        data_next[data_cnt_reg] = 1;
    end else begin
        data_next[data_cnt_reg] = 0;
    end

    if (data_cnt_reg == 0) begin
        tick_cnt_next = 0;
        data_cnt_next = 39;
        dht11_done_next = 1;
        n_state = VALID;
    end else begin
        n_state = DATA_SYNC;
        data_cnt_next = data_cnt_reg - 1;
    end
end
```

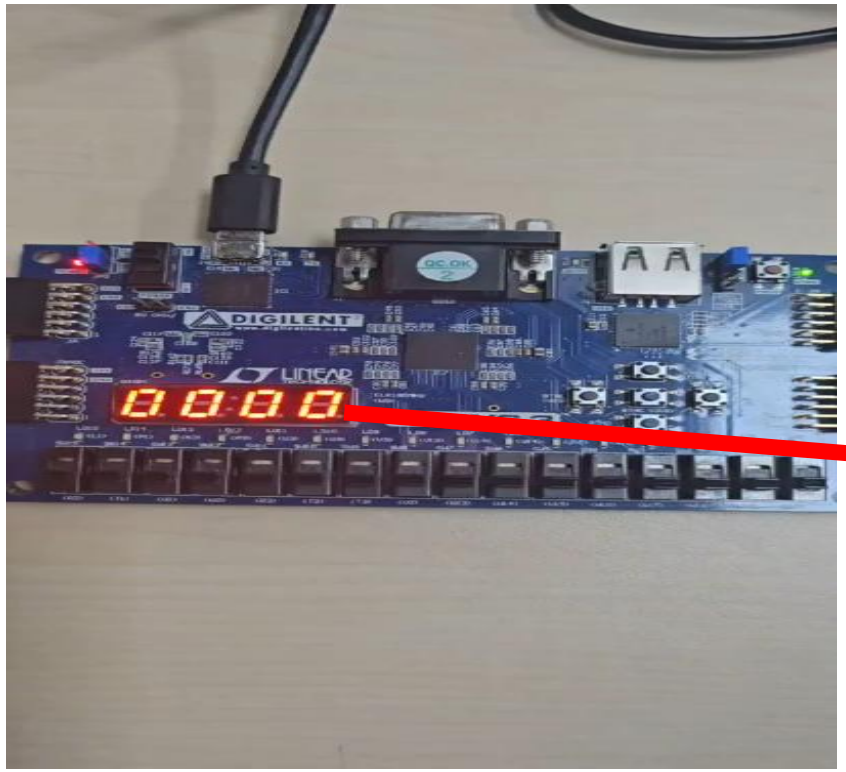
- 비트 판별, 저장, checksum 검증, 상태 전이 →DATA_DETECT에서 모두 처리
- tick 카운트 도중 상태가 전이되며 오동작 발생

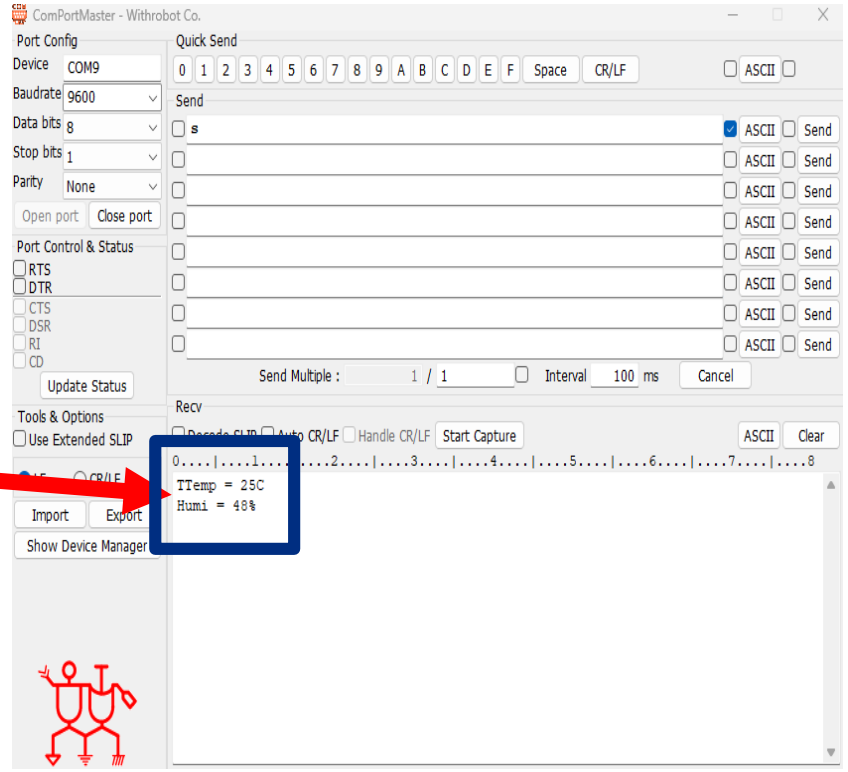
- DATA_DETECT → DATA_DETECT_f → VALID 단계 분리
- 각 단계 역할을 분리하여 타이밍 안정성과 FSM 흐름 개선

동작 영상 및 설명

■ DHT11(Hm/T) + UART

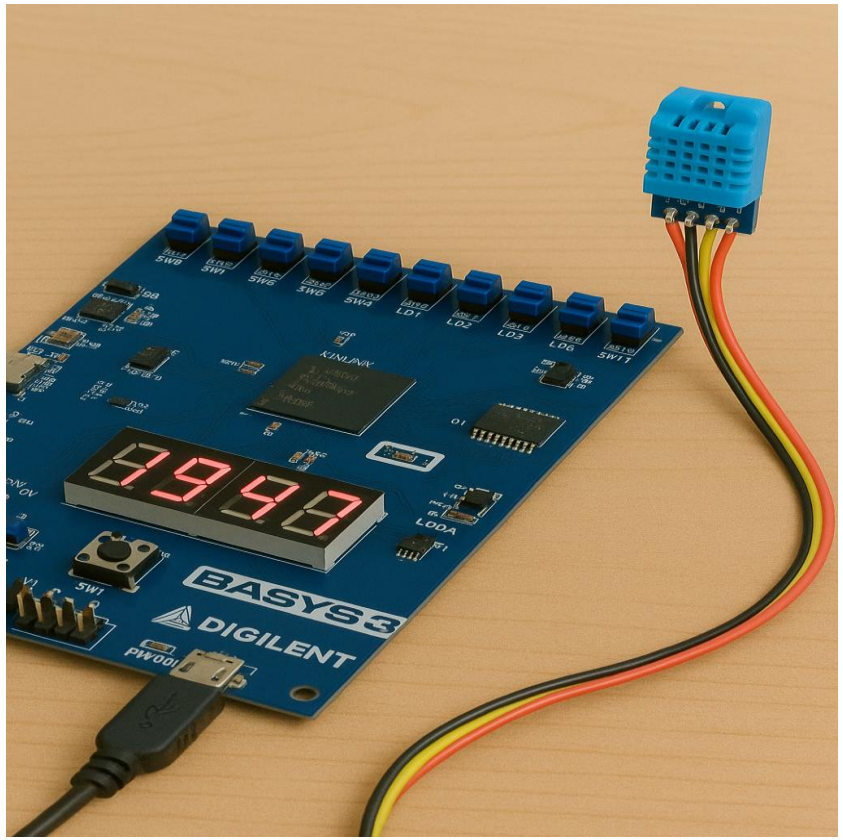
- 동작영상





고찰

DHT11(Hm/T) + UART



- 하드웨어 타이밍 제어의 중요성

이번 프로젝트를 통해 수 μs 단위의 타이밍 조절이 얼마나 중요한지 체감했다. 특히 FSM 상태 간 전이가 빠를 경우, 신호가 안정되기 전에 처리가 이뤄져 통신 오류로 이어질 수 있음을 알게 되었다.

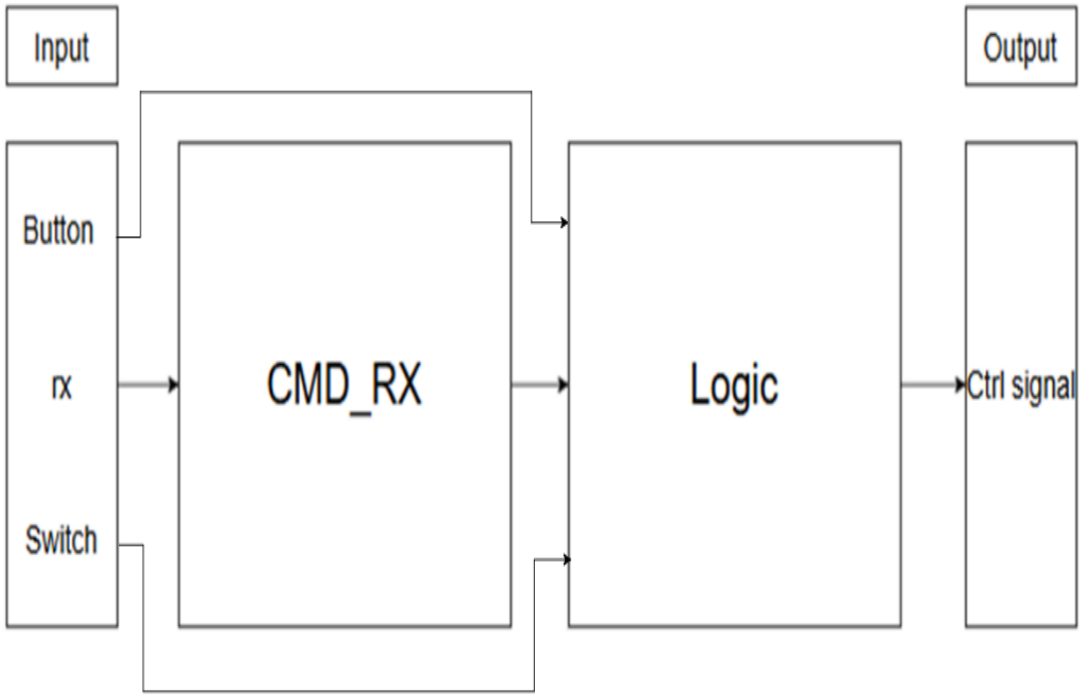
- 실제 센서 프로토콜의 복잡성 체험

DHT11은 단순한 센서처럼 보이지만, 단일 버스 통신에서의 동기화, 비트 해석, checksum 처리까지 직접 구현하면서 하드웨어 프로토콜 설계의 복잡성을 경험할 수 있었다.

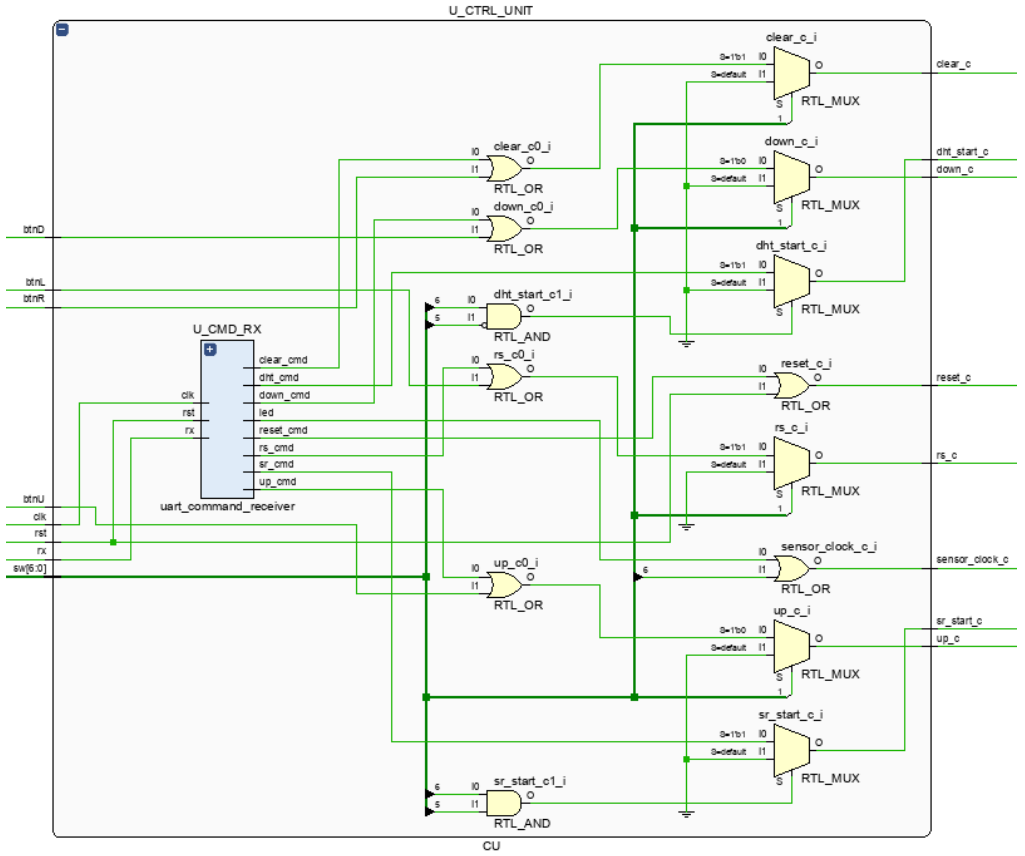
설계 과정

Control Unit

- Block Diagram



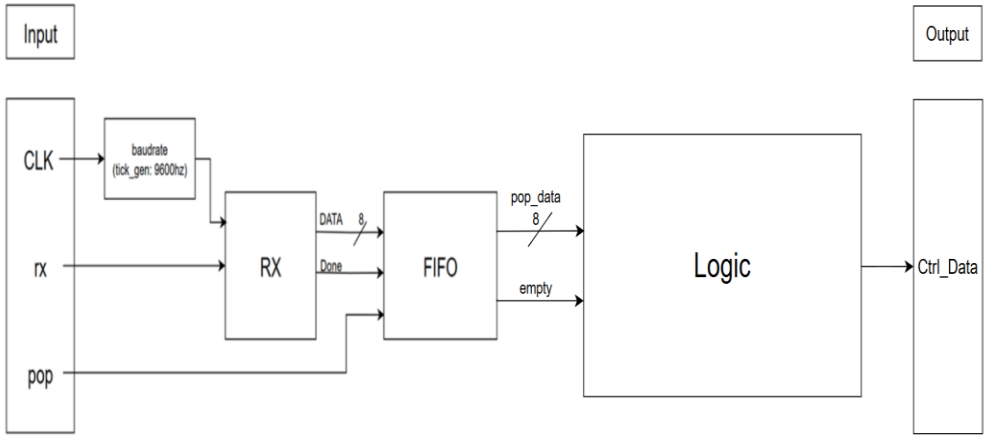
- Schematic

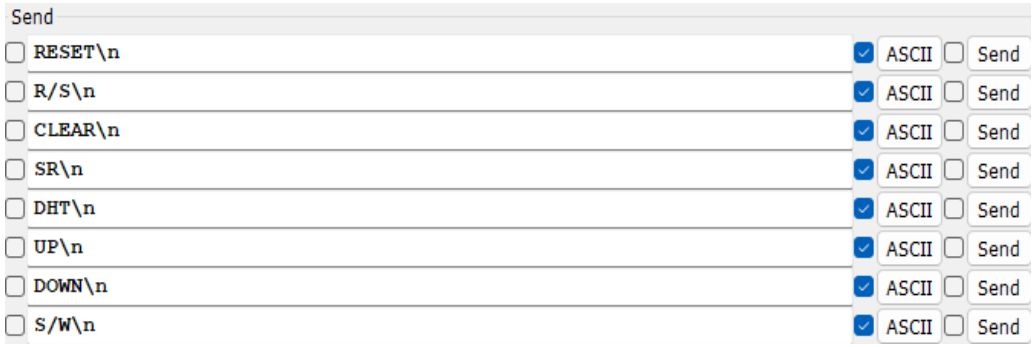


설계 과정

CMD_RX(Command RX)

- Block Diagram





Sw6과 연동(or)

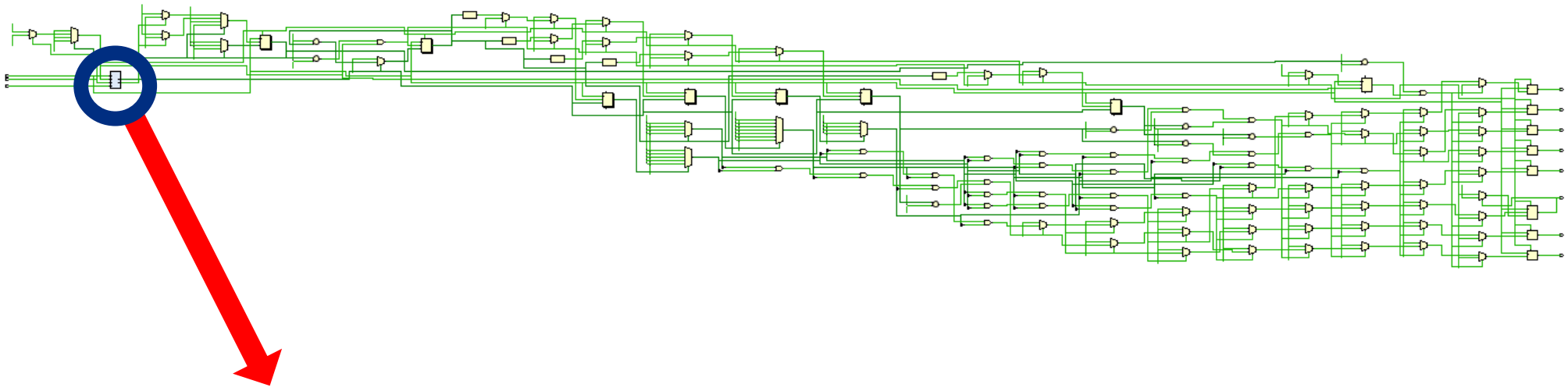
LED6으로 상태확인가능
LED6: ON(센서)
OFF(시계)

Command	기능
RESET₩n	Reset
R/S₩n	STOPWATCH에서 Run/Stop
CLEAR₩n	STOPWATCH에서 CLEAR
SR₩n	SR04 start
DHT₩n	DHT11 start
UP₩n	WATCH에서 UP
DOWN₩n	WATCH에서 DOWN
S/W₩n	센서/시계 모드

설계 과정

- **CMD_RX(Command RX)**

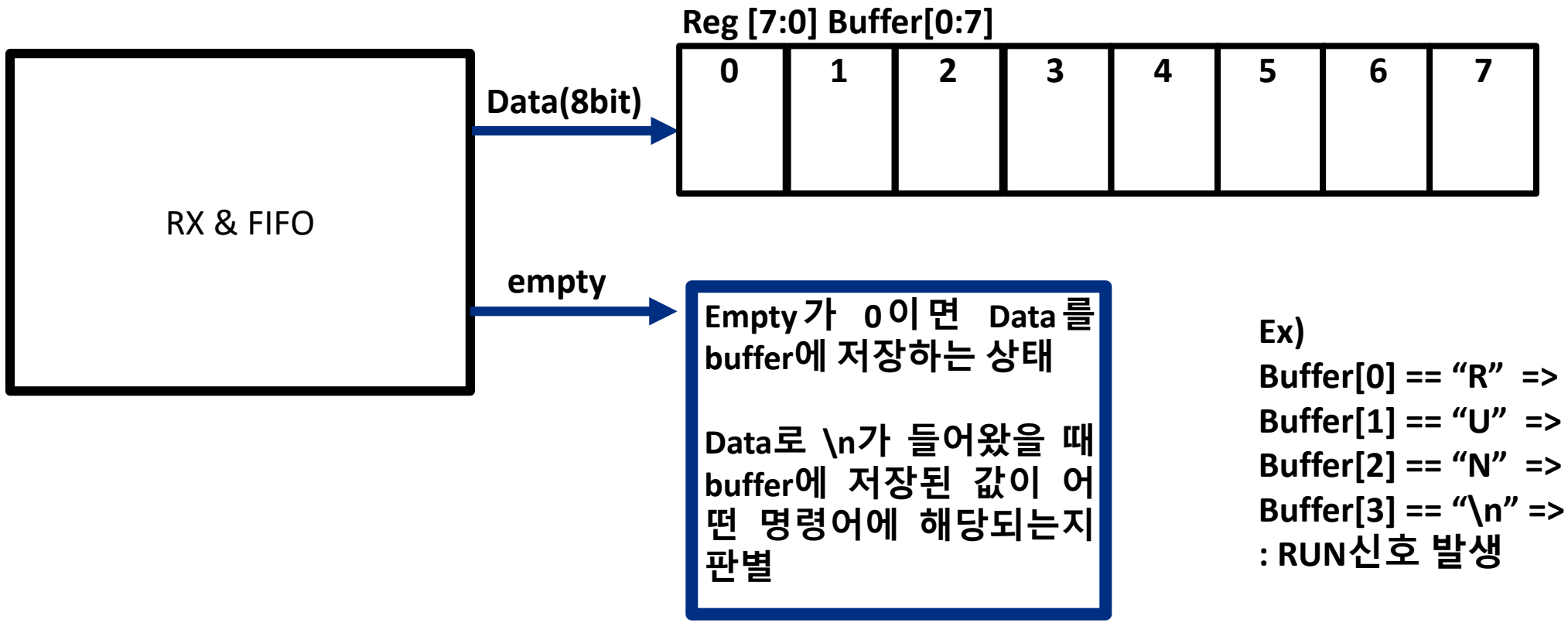
  - **Schematic**



설계 과정

- CMD_RX(Command RX)

  -동작방식



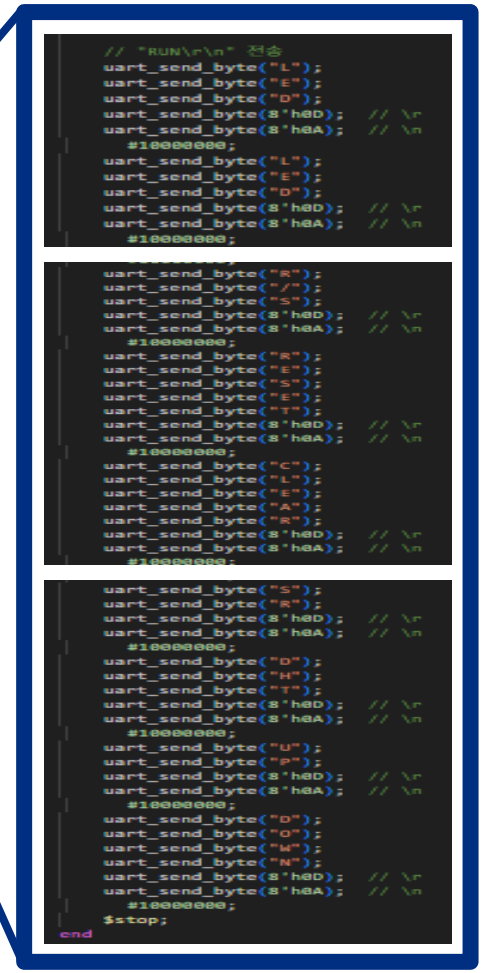
Ex)
Buffer[0] == “R” => RECV상태
Buffer[1] == “U” => RECV상태
Buffer[2] == “N” => RECV상태
Buffer[3] == “\n” => check상태
: RUN신호 발생

기능 검증 및 설명

Control Unit

- Simulation

CU 시뮬레이션: 확인하고자 하는 것: buffer에 단어를 넣어서 output 신호를 제어할 수 있는가.



특성	OUTPUT						
틱발생 ← 버튼	Reset_cmd	Rs_cmd	Clear_cmd	Sr_cmd	dht_cmd	Up_cmd	Down_cmd
상태반전 ← 스위치	Led						

nextchip

KFIA 한국팹리스산업협회 Korea Fabless Industry Association

대한상공회의소 서울기술교육센터

동작 영상 및 설명

Entire Module

LED 정리

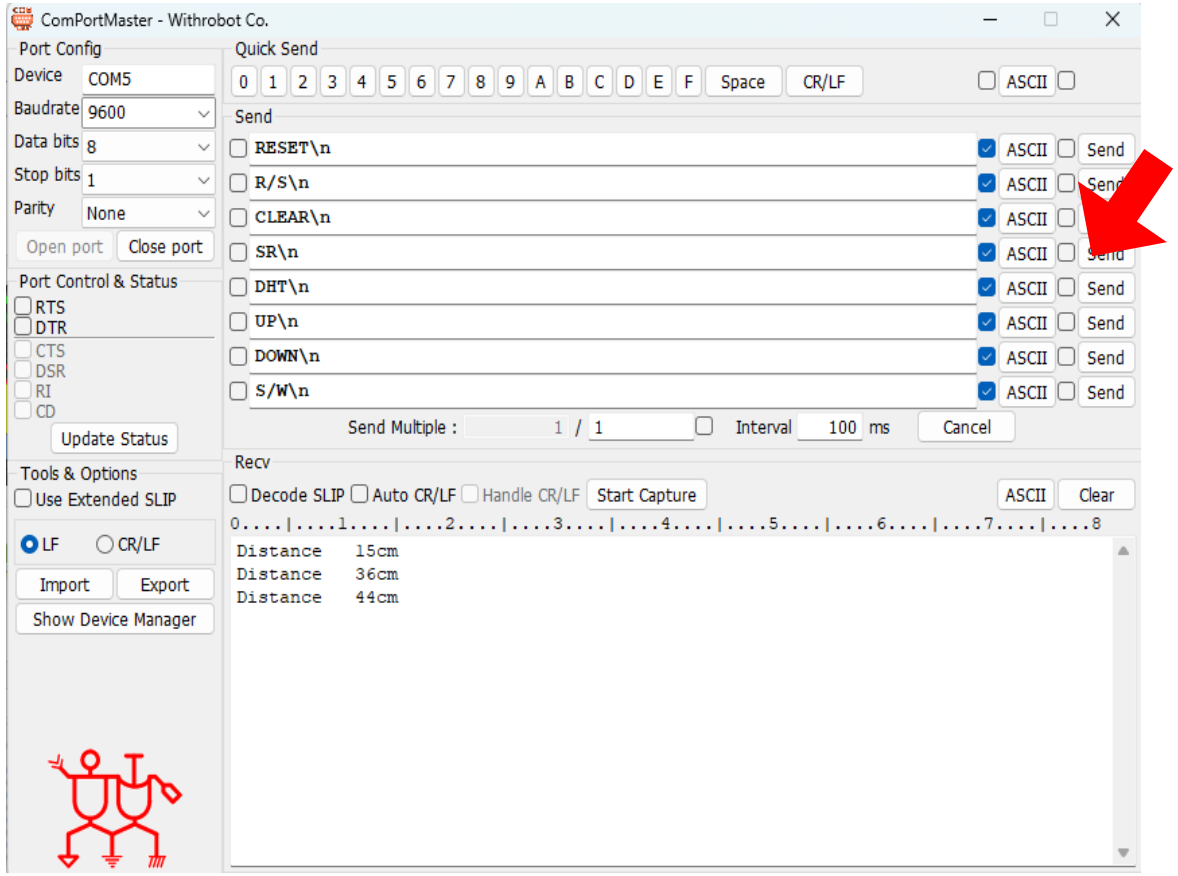
LED0	LED1	LED2	LED3	LED4	LED5	LED6	LED7	LED8	LED9	LED10	LED11	LED12	LED13	LED14	LED15
sw[1]과 sw[0]의 상태						모드			Dht11 상태				Dht11 valid		뻐꾸기

SWITCH 정리

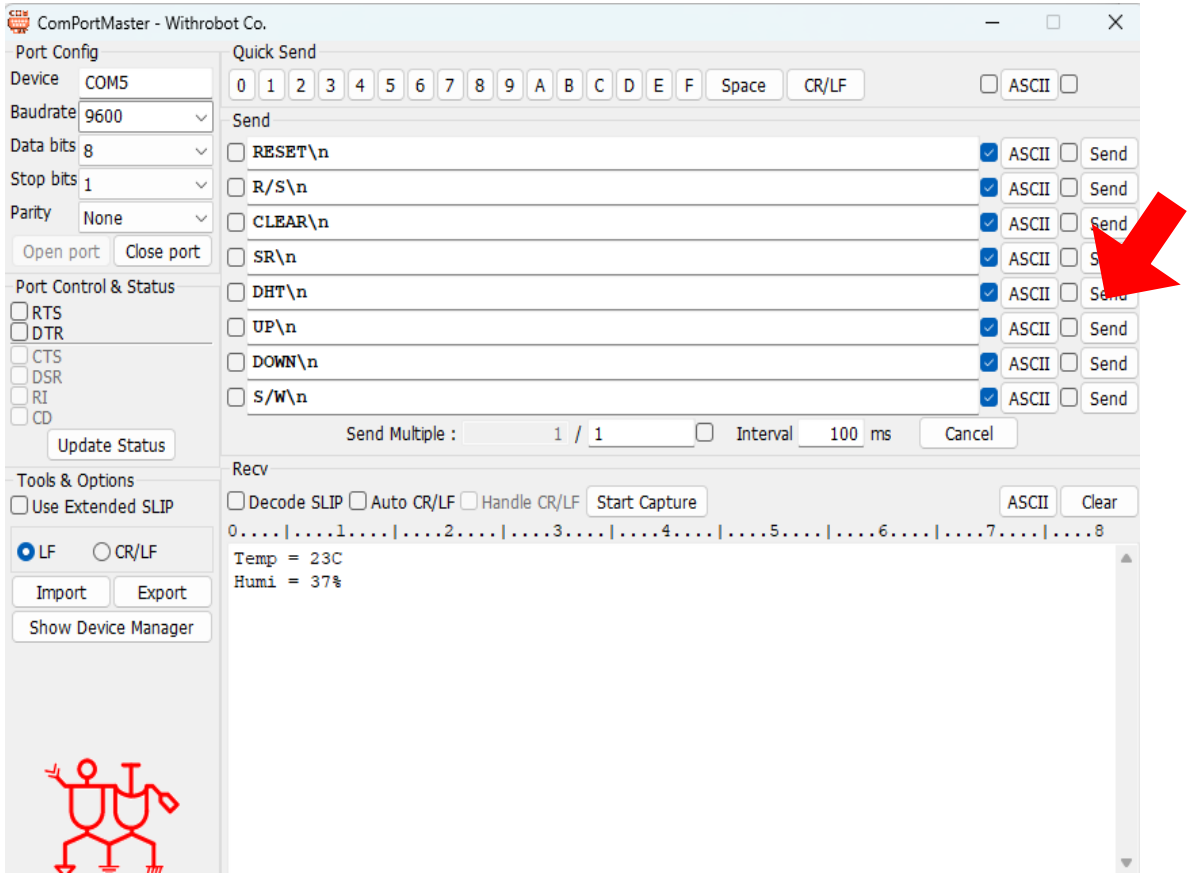
SW[0]	SW[1]	SW[2]	SW[3]	SW[4]	SW[5]	SW[6]
FND : 시분(1)/초ms(0)	Stopwatch(1)/watch(0)	초제어	분제어	시제어	초음파(1)/온습도(0)	모드

동작 영상 및 설명

■ 초음파 센서



■ 온습도 센서



nextchip

KFIA 한국팹리스산업협회 Korea Fabless Industry Association

대한상공회의소 서울기술교육센터

동작 영상 및 설명

■ **Entire Module**

고찰

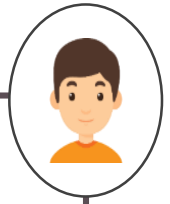
서윤철

각 모듈을 합치는 과정에서 모듈 이름과 모듈내 input, output 구성 차이가 존재하여, 그 차이를 분별하고 정리하는데 어려움을 겪었습니다. 프로젝트 초반으로 다시 돌아간다면, 공통된 모듈을 한번 정리한 후 이름과 port의 이름을 통일한다면 더 효율적으로 프로젝트를 진행할 수 있지 않았나 라는 아쉬움이 남습니다.-_-

박승헌

프로젝트를 진행하며 처음에는 FSM 구조를 단순화하려고 하나의 상태에 여러 기능을 넣었지만, 오히려 동작이 불안정했습니다.
이후 상태를 세분화하고 각 기능을 나눔으로써 정확성과 가독성 모두 개선되었고, 설계의 안정성은 '단순함'이 아니라 '역할의 분리'에서 온다는 교훈을 얻었습니다.

권희식



Testbench 작성과 시뮬레이션 기반 디지털 회로 검증 역량을 키울 수 있었습니다. 단순히 RTL 코드를 작성하는 데서 그치는 것이 아니라, 테스트 벤치를 통해 모듈별로 원하는 기능이 정상 동작하는지 검증하고, 시뮬레이션 파형을 해석하여 신호 흐름을 시각적으로 확인하는 과정을 반복하면서 디지털 회로 검증의 중요성을 알 수 있었습니다.

오정일

센서를 제어하는 Verilog 컨트롤러를 구현하면서 상태 멈춤 문제를 겪었고, 이를 해결하기 위해 상태 초기화와 조건문 처리 순서를 점검해야 했습니다. 출력 결과와 파형, 레지스터를 통해 상태 머신의 실제 동작을 파악하는 과정이 중요하다는 것을 느꼈고, 디지털 회로는 타이밍 안에서의 동작이 핵심이며, 안정적인 설계를 위해 구조적 설계와 반복적인 검증이 필요하다는 점을 배웠습니다.

감사합니다