

# **uart\_fifo**

# **systemverilog 검증**

2025.09.15

유지훈, 신상혁



# 프로젝트 개요

---

Introduction

BLOCK Diagram

UART\_TX

UART\_RX

FIFO

UART\_FIFO

Top Module 시뮬레이션

COUNTER\_DATAPATH

트러블슈팅&고찰

# Introduction

## 기능

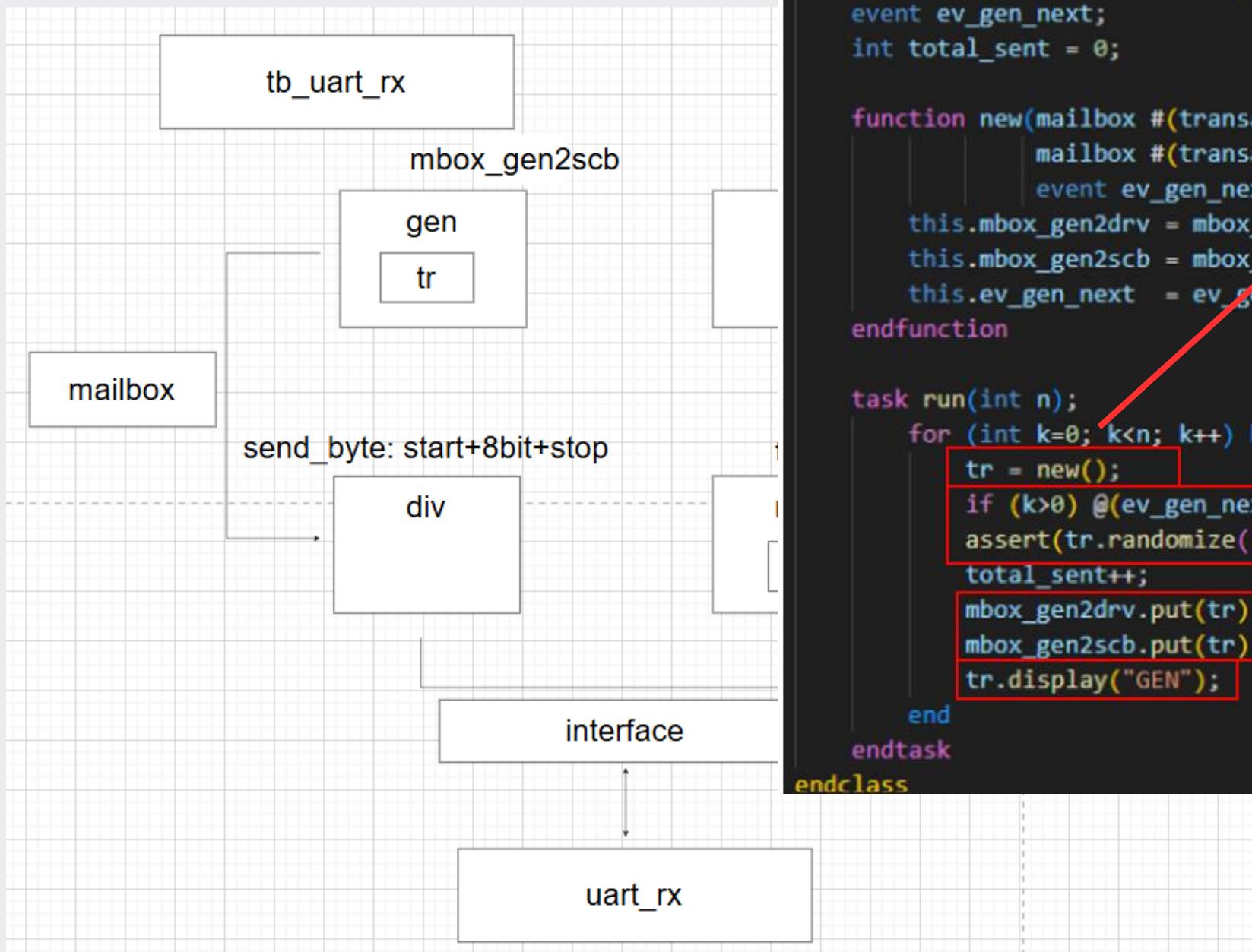
UART를 통해 PC와 데이터를 주고 받으며 FIFO를 통해 데이터 전송의 안정성을 확보하고 PC와 Button의 명령에 따라 COUNTER 값을 조정하고 그 값을 PC로 다시 받아온다.

## 목표

- 데이터 전송 시 FIFO 버퍼를 통해 데이터 유실 없이 안정적으로 처리되는지 확인한다.
- Counter 제어 기능 구현
- PC 명령어와 버튼 입력에 따라 Counter가 정상적으로 동작하는지 검증한다.
- UART + FIFO + Counter 모듈을 통합하여 전체 시스템이 올바르게 동작하는지 검증한다.

# BLOCK Diagram

uart\_rx



gen

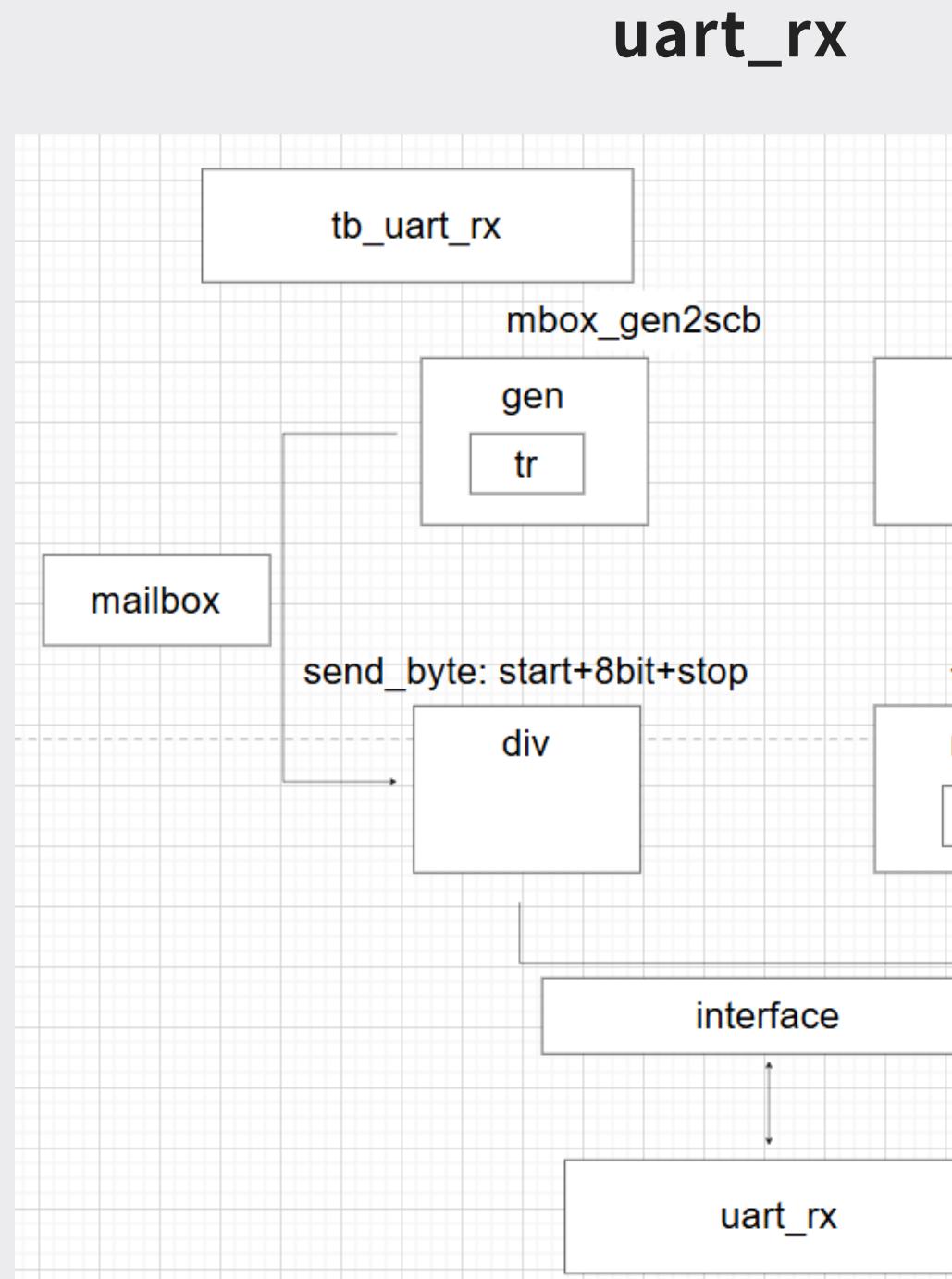
```
// ----- Generator -----
class generator;
  transaction tr;
  mailbox #(transaction) mbox_gen2drv;
  mailbox #(transaction) mbox_gen2scb;
  event ev_gen_next;
  int total_sent = 0;

  function new(mailbox #(transaction) mbox_gen2drv,
              mailbox #(transaction) mbox_gen2scb,
              event ev_gen_next);
    this.mbox_gen2drv = mbox_gen2drv;
    this.mbox_gen2scb = mbox_gen2scb;
    this.ev_gen_next = ev_gen_next;
  endfunction

  task run(int n);
    for (int k=0; k<n; k++) begin
      tr = new();
      if (k>0) @ev_gen_next;
      assert(tr.randomize()) else $error("[GEN] randomize error");
      total_sent++;
      mbox_gen2drv.put(tr);
      mbox_gen2scb.put(tr);
      tr.display("GEN");
    end
  endtask
endclass
```

새로운 transaction 객체 생성

# BLOCK Diagram



gen

```
// ----- Generator -----
class generator;
  transaction tr;
  mailbox #(transaction) mbox_gen2drv;
  mailbox #(transaction) mbox_gen2scb;
  event ev_gen_next;
  int total_sent = 0;

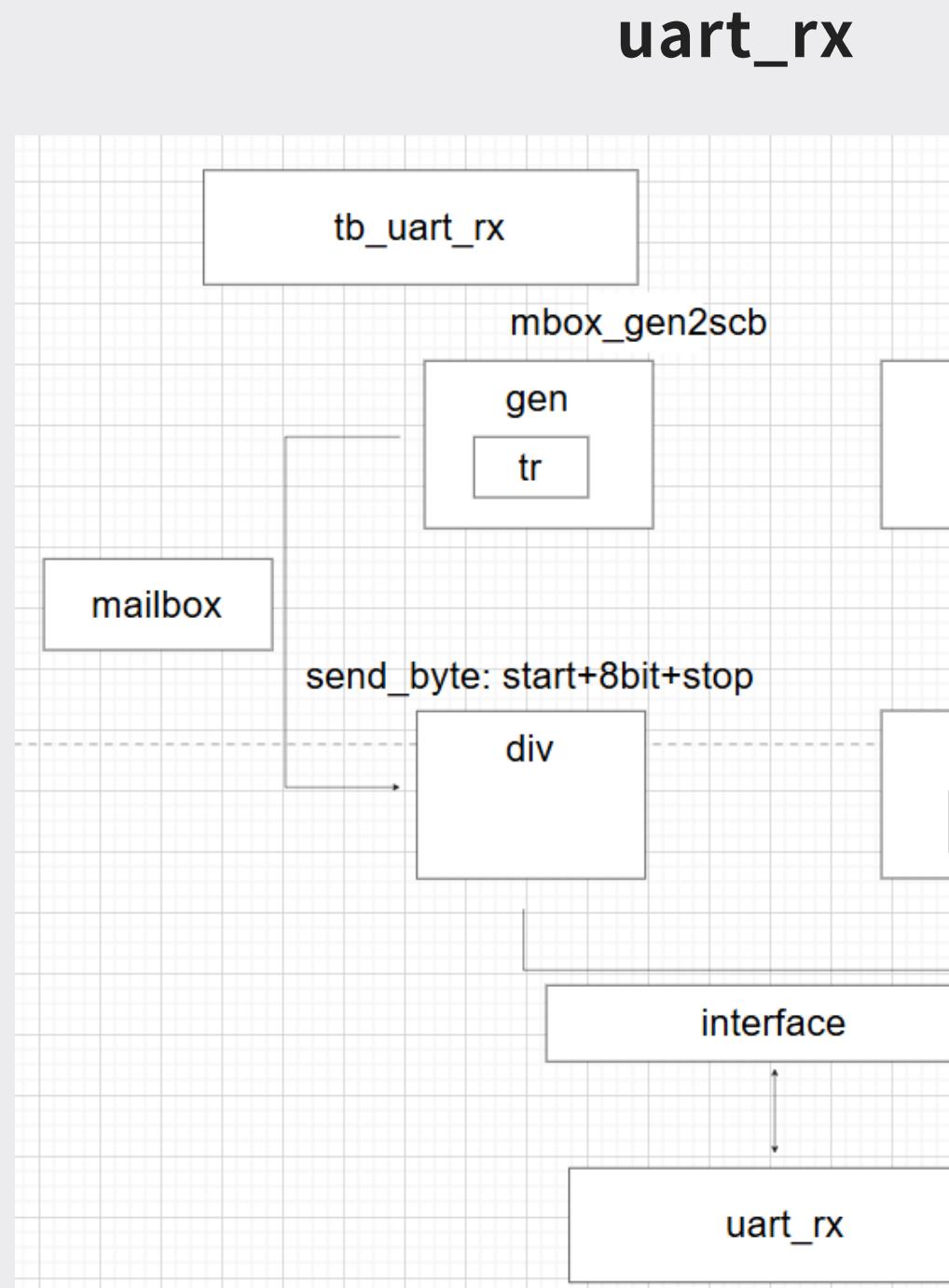
  function new(mailbox #(transaction) mbox_gen2drv,
              mailbox #(transaction) mbox_gen2scb,
              event ev_gen_next);
    this.mbox_gen2drv = mbox_gen2drv;
    this.mbox_gen2scb = mbox_gen2scb;
    this.ev_gen_next = ev_gen_next;
  endfunction

  task run(int n);
    for (int k=0; k<n; k++) begin
      tr = new();
      if (k>0) @ev_gen_next;
      assert(tr.randomize()) else $error("[GEN] randomize error");
      total_sent++;
      mbox_gen2drv.put(tr);
      mbox_gen2scb.put(tr);
      tr.display("GEN");
    end
  endtask
endclass
```

새로운 transaction 객체 생성

첫 번째 이후는 이벤트 기다림/랜덤 값 생성

# BLOCK Diagram



gen

```
// ----- Generator -----
class generator;
  transaction tr;
  mailbox #(transaction) mbox_gen2drv;
  mailbox #(transaction) mbox_gen2scb;
  event ev_gen_next;
  int total_sent = 0;

  function new(mailbox #(transaction) mbox_gen2drv,
              mailbox #(transaction) mbox_gen2scb,
              event ev_gen_next);
    this.mbox_gen2drv = mbox_gen2drv;
    this.mbox_gen2scb = mbox_gen2scb;
    this.ev_gen_next = ev_gen_next;
  endfunction

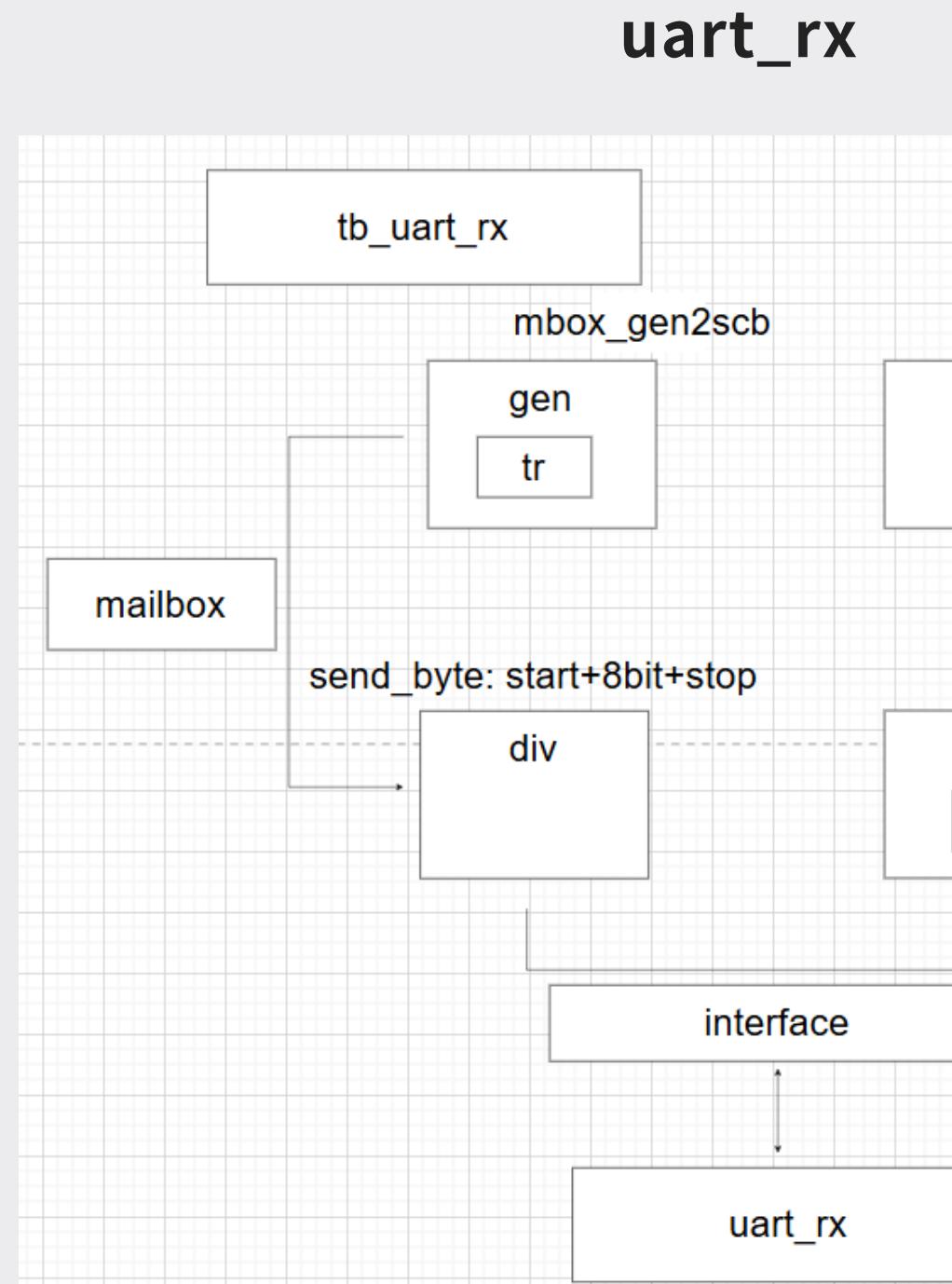
  task run(int n);
    for (int k=0; k<n; k++) begin
      tr = new();
      if (k>0) @ev_gen_next;
      assert(tr.randomize()) else $error("[GEN] randomize error");
      total_sent++;
      mbox_gen2drv.put(tr);
      mbox_gen2scb.put(tr);
      tr.display("GEN");
    end
  endtask
endclass
```

새로운 transaction 객체 생성

첫 번째 이후는 이벤트 기다림/랜덤 값 생성

Driver 쪽 mailbox로 전송  
Scoreboard 쪽 mailbox로 전송

# BLOCK Diagram



gen

```
// ----- Generator -----
class generator;
  transaction tr;
  mailbox #(transaction) mbox_gen2drv;
  mailbox #(transaction) mbox_gen2scb;
  event ev_gen_next;
  int total_sent = 0;

  function new(mailbox #(transaction) mbox_gen2drv,
             mailbox #(transaction) mbox_gen2scb,
             event ev_gen_next);
    this.mbox_gen2drv = mbox_gen2drv;
    this.mbox_gen2scb = mbox_gen2scb;
    this.ev_gen_next = ev_gen_next;
  endfunction

  task run(int n);
    for (int k=0; k<n; k++) begin
      tr = new();
      if (k>0) @ev_gen_next;
      assert(tr.randomize()) else $error("[GEN] randomize error");
      total_sent++;
      mbox_gen2drv.put(tr);
      mbox_gen2scb.put(tr);
      tr.display("GEN");
    end
  endtask
endclass
```

새로운 transaction 객체 생성

첫 번째 이후는 이벤트 기다림/랜덤 값 생성

Driver 쪽 mailbox로 전송  
Scoreboard 쪽 mailbox로 전송

생성된 트랜잭션 출력

# BLOCK Diagram

```
uart_rx    drv
           +-----+ Driver +-----+
class driver;
  transaction tr;
  mailbox #(transaction) mbox_gen2drv;
  virtual uart_rx_if u_if;
  event ev_mon_next;

  function new(mailbox #(transaction) mbox_gen2drv,
              virtual uart_rx_if u_if,
              event ev_mon_next);
    this.mbox_gen2drv = mbox_gen2drv;
    this.u_if        = u_if;
    this.ev_mon_next = ev_mon_next;
  endfunction

  task reset();
    u_if.rst <= 1;
    u_if.rx  <= 1;
    repeat(5) @(posedge u_if.clk);
    u_if.rst <= 0;
    repeat(50) @(posedge u_if.clk);
    $display("[DRV] Reset done");
  endtask
endclass
```

mailbox

## 1. 리셋 활성화

- `u_if.rst <= 1;`: DUT에 리셋 걸어서 내부 상태 모두 초기화.
- `u_if.rx <= 1;`: UART 입력선을 idle 상태(=논리 1)로 둠.

# BLOCK Diagram

```
uart_rx    drv
           +-----+ Driver +-----+
class driver;
  transaction tr;
  mailbox #(transaction) mbox_gen2drv;
  virtual uart_rx_if u_if;
  event ev_mon_next;

  function new(mailbox #(transaction) mbox_gen2drv,
              virtual uart_rx_if u_if,
              event ev_mon_next);
    this.mbox_gen2drv = mbox_gen2drv;
    this.u_if        = u_if;
    this.ev_mon_next = ev_mon_next;
  endfunction

  task reset();
    u_if.rst <= 1;
    u_if.rx  <= 1;
    repeat(5) @(posedge u_if.clk);
    u_if.rst <= 0;
    repeat(50) @(posedge u_if.clk);
    $display("[DRV] Reset done");
  endtask
endclass
```

## 1. 리셋 활성화

- u\_if.rst <= 1; : DUT에 리셋 걸어서 내부 상태 모두 초기화.
- u\_if.rx <= 1; : UART 입력선을 idle 상태(=논리 1)로 둠.

## 1. 리셋 유지

- repeat(5) @(posedge u\_if.clk);
- 시스템 클럭 5사이클 동안 리셋 유지 → 충분히 DUT 내부 레지스터들이 클리어될 시간 확보.

# BLOCK Diagram

```
uart_rx    drv
// ----- Driver -----
class driver;
  transaction tr;
  mailbox #(transaction) mbox_gen2drv;
  virtual uart_rx_if u_if;
  event ev_mon_next;

  function new(mailbox #(transaction) mbox_gen2drv,
              virtual uart_rx_if u_if,
              event ev_mon_next);
    this.mbox_gen2drv = mbox_gen2drv;
    this.u_if        = u_if;
    this.ev_mon_next = ev_mon_next;
  endfunction

  task reset();
    u_if.rst <= 1;
    u_if.rx  <= 1;
    repeat(5) @(posedge u_if.clk);
    u_if.rst <= 0;
    repeat(50) @(posedge u_if.clk);
    $display("[DRV] Reset done");
  endtask
endclass
```

## 1. 리셋 활성화

- `u_if.rst <= 1;`: DUT에 리셋 걸어서 내부 상태 모두 초기화.
- `u_if.rx <= 1;`: UART 입력선을 idle 상태(=논리 1)로 둠.

## 1. 리셋 유지

- `repeat(5) @(posedge u_if.clk);`
- 시스템 클럭 5사이클 동안 리셋 유지 → 충분히 DUT 내부 리지스터들이 클리어될 시간 확보.

## 1. 리셋 해제

- `u_if.rst <= 0;`: DUT을 정상 동작 모드로 전환.

# BLOCK Diagram

```
// ----- Driver -----
class driver;
    transaction tr;
    mailbox #(transaction) mbox_gen2drv;
    virtual uart_rx_if u_if;
    event ev_mon_next;

function new(mailbox #(transaction) mbox_gen2drv,
            virtual uart_rx_if u_if,
            event ev_mon_next);
    this.mbox_gen2drv = mbox_gen2drv;
    this.u_if        = u_if;
    this.ev_mon_next = ev_mon_next;
endfunction

task reset();
    u_if.rst <= 1;
    u_if.rx  <= 1;
    repeat(5) @(posedge u_if.clk);
    u_if.rst <= 0;
    repeat(50) @(posedge u_if.clk);
    $display("[DRV] Reset done");
endtask
```

- u\_if.rst <= 1; : DUT에 리셋 걸어서 내부 상태 모두 초기화.
  - u\_if.rx <= 1; : UART 입력선을 idle 상태(=논리 1)로 둠.

### 1. 리셋 유지

  - repeat(5) @ (posedge u\_if.clk);
  - 시스템 클럭 5사이클 동안 리셋 유지 → 충분히 DUT 내부 레지스터들이 클리어될 시간 확보.

### 1. 리셋 해제

  - u\_if.rst <= 0; : DUT을 정상 동작 모드로 전환.

### 1. 안정화 대기

  - repeat(50) @ (posedge u\_if.clk);
  - 리셋 해제 후 DUT 내부 FSM, 카운터 등이 안정될 때까지 50 클럭 기다림.

# BLOCK Diagram

```
// ----- Driver -----
class driver;
    transaction tr;
    mailbox #(transaction) mbox_gen2drv;
    virtual uart_rx_if u_if;
    event ev_mon_next;

    function new(mailbox #(transaction) mbox_gen2drv,
                virtual uart_rx_if u_if,
                event ev_mon_next);
        this.mbox_gen2drv = mbox_gen2drv;
        this.u_if         = u_if;
        this.ev_mon_next = ev_mon_next;
    endfunction

    task reset();
        u_if.rst <= 1;
        u_if.rx  <= 1;
        repeat(5) @(posedge u_if.clk);
        u_if.rst <= 0;
        repeat(50) @(posedge u_if.clk);
        $display("[DRV] Reset done");
    endtask
endclass
```

- u\_if.rst <= 1; : DUT에 리셋 걸어서 내부 상태 모두 초기화.
  - u\_if.rx <= 1; : UART 입력선을 idle 상태(=논리 1)로 둠.

### 1. 리셋 유지

  - repeat(5) @(posedge u\_if.clk);
  - 시스템 클럭 5사이클 동안 리셋 유지 → 충분히 DUT 내부 레지스터들이 클리어될 시간 확보.

### 1. 리셋 해제

  - u\_if.rst <= 0; : DUT을 정상 동작 모드로 전환.

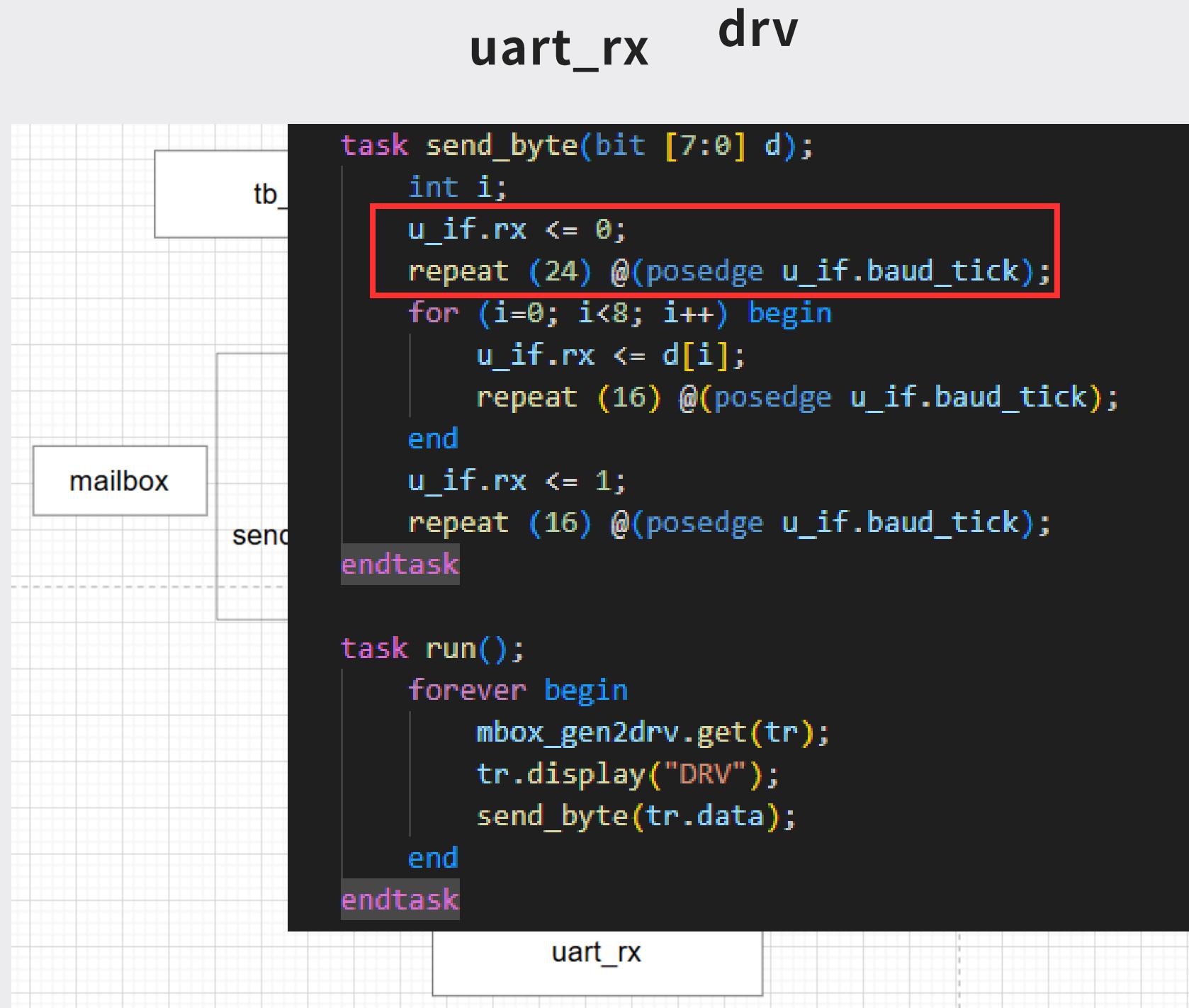
### 1. 안정화 대기

  - repeat(50) @(posedge u\_if.clk);
  - 리셋 해제 후 DUT 내부 FSM, 카운터 등이 안정될 때까지 50 클럭 기다림.

### 1. 로그 출력

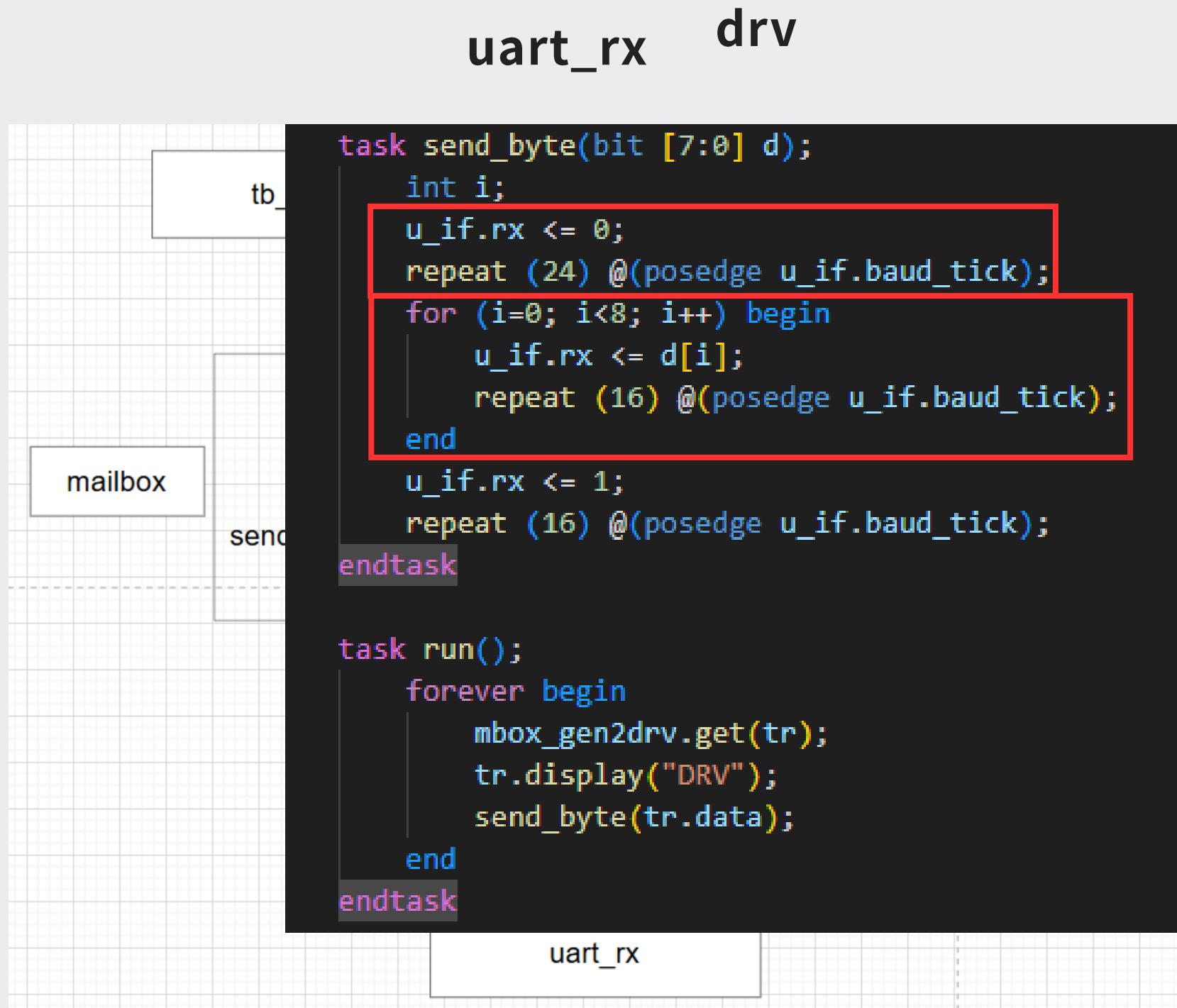
  - \$display("[DRV] Reset done");
  - 시뮬레이션 콘솔에 리셋이 완료됐음을 알려줌.

# BLOCK Diagram



- Start bit : rx=0을 24틱(=1.5비트) 동안 유지.  
RX 쪽이 안정적으로 동기화할 수 있게 확보.

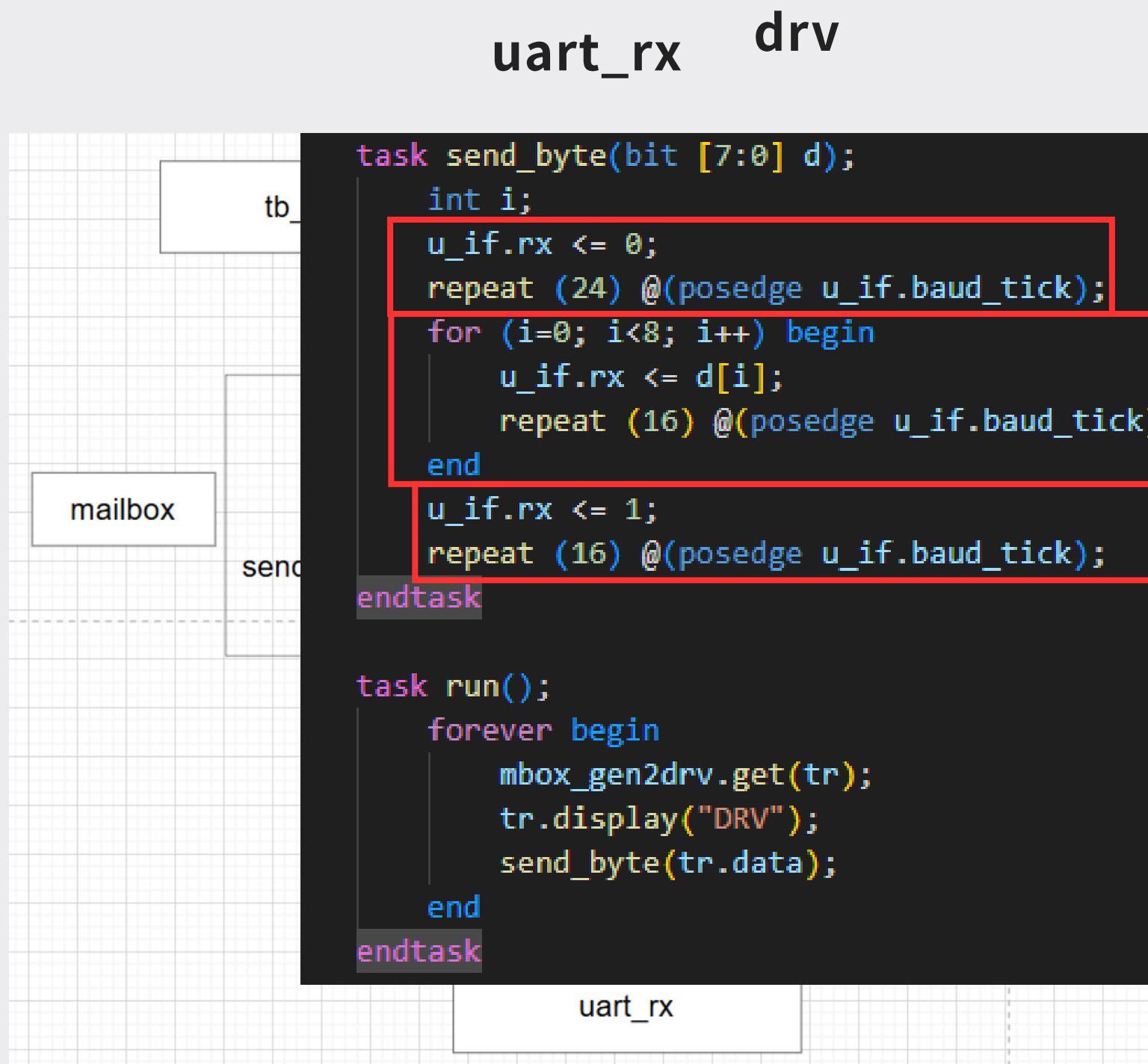
# BLOCK Diagram



- Start bit : rx=0을 24틱(=1.5비트) 동안 유지.

- Data bits : 8비트 데이터를 LSB-first로 전송.
  - 각 비트는 16틱(=1비트 길이).

# BLOCK Diagram

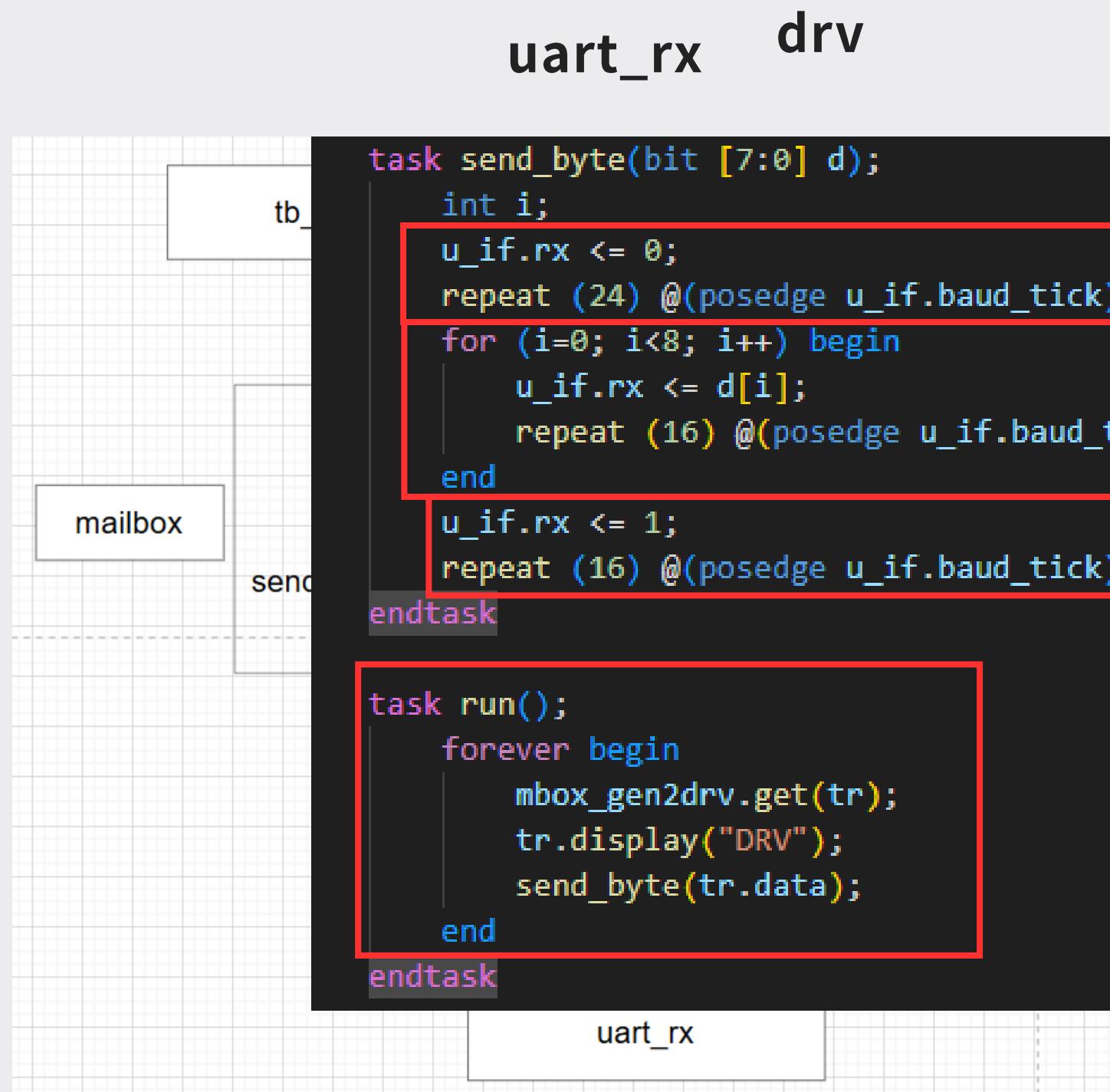


- Start bit : rx=0을 24틱(=1.5비트) 동안 유지.

- Data bits : 8비트 데이터를 LSB-first로 전송.
  - 각 비트는 16틱(=1비트 깅이)

- Stop bit : rx=1을 16틱 동안 유지.

# BLOCK Diagram



- Start bit : rx=0을 24틱(=1.5비트) 동안 유지.
- Data bits : 8비트 데이터를 LSB-first로 전송.
  - 각 비트는 16틱(=1비트 깊이)
- Stop bit : rx=1을 16틱 동안 유지.
- mbox\_gen2drv.get(tr)
  - Generator가 mailbox로 넣어둔 transaction(data)을 꺼냄.
  - Stimulus 전달 과정.
  - tr.display("DRV")
- 수신한 데이터를 로그로 출력 → 디버깅 용도.
  - send\_byte(tr.data)
- 받아온 데이터를 UART 프레임 형태로 변환해서 실제 DUT rx 신호에 밀어 넣음.
- 여기서 위의 send\_byte() task가 실행됨.

# BLOCK Diagram

uart\_rx

scb

The diagram shows a sequence of interactions between three components: uart\_rx, mailbox, and scb. The process starts with a synchronous message from uart\_rx to mailbox. This is followed by a synchronous message from mailbox to scb. Finally, a return message is sent from scb back to uart\_rx.

```
sequenceDiagram
    participant U as uart_rx
    participant M as mailbox
    participant S as scb
    U->>M: 
    activate M
    M->>S: 
    activate S
    S-->>U: 
```

```
task report();
    $display("===== Test report =====");
    $display("== Total sent : %d", gen.total_sent);
    $display("== Checked : %d", scb.total_cnt);
    $display("== PASS : %d", scb.pass_cnt);
    $display("== FAIL : %d", scb.fail_cnt);
    $display("=====");
endtask

task run(int n=20, int timeout=1_000_000);
    -> ev_gen_next;
    fork
        gen.run(n);
        drv.run();
        mon.run();
        scb.run();
    join_none

    fork
        begin wait (scb.total_cnt==n);
        begin repeat(timeout) @(posedge clk);
            if (scb.total_cnt < n)
                $error("[ENV] Timeout");
        end
        join_any
    end

    report();
    $stop;
endtask 
```

mailbo

task report();  
task run(int n=20, int timeout=1\_000\_000);  
begin  
end  
join  
end  
report();  
\$stop;  
endtask

• gen.run(n) : Stimulus n개 생성.  
• drv.run() : Stimulus를 DUT로 전송.  
• mon.run() : DUT 출력 관찰.  
• scb.run() : Golden data와 비교해  
PASS/FAIL 판정.

# BLOCK Diagram

uart\_rx

scb

```
task report()
    $display("SCB] PASS exp=0x68 got=0x68")
    $display("508855000 [GEN] data=0xa2 captured=0x0 rx_done=0")
    $display("508865000 [DBG] rx_done=1, rx_data=0x68")
    $display("511245000 [DRV] data=0xa2 captured=0x0 rx_done=0")
    $display("535735000 [MON] data=0x0 captured=0xa2 rx_done=1")
    [SCB] PASS exp=0xa2 got=0xa2
=====
===== Test report =====
== Total sent : 20
== Checked : 20
== PASS : 20
== FAIL : 0
=====
$stop called at time : 535735 ns : File "D:/jihun/0912
report();
$stop;
endtask
```

손.

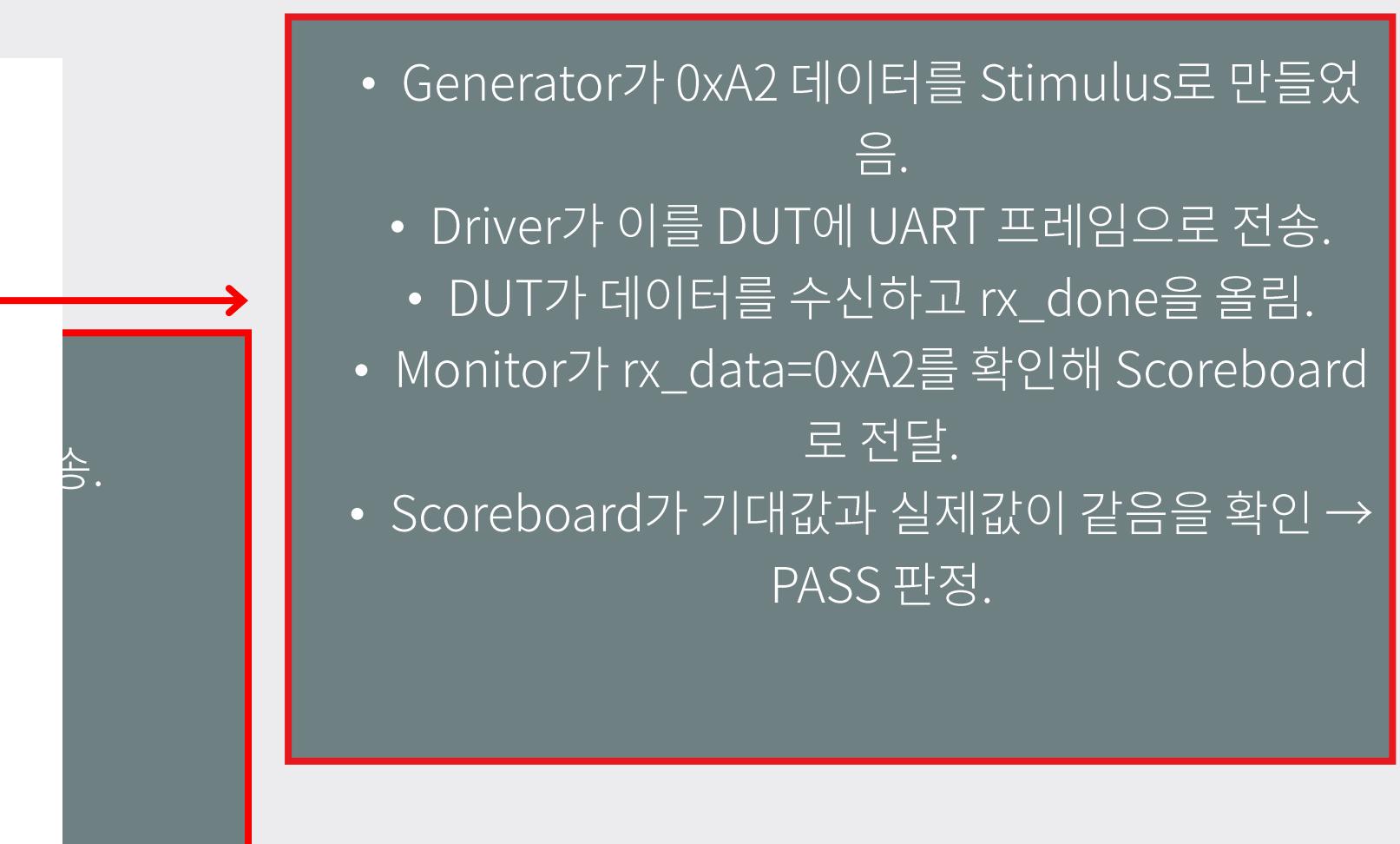
# BLOCK Diagram

```
uart_rx          scb

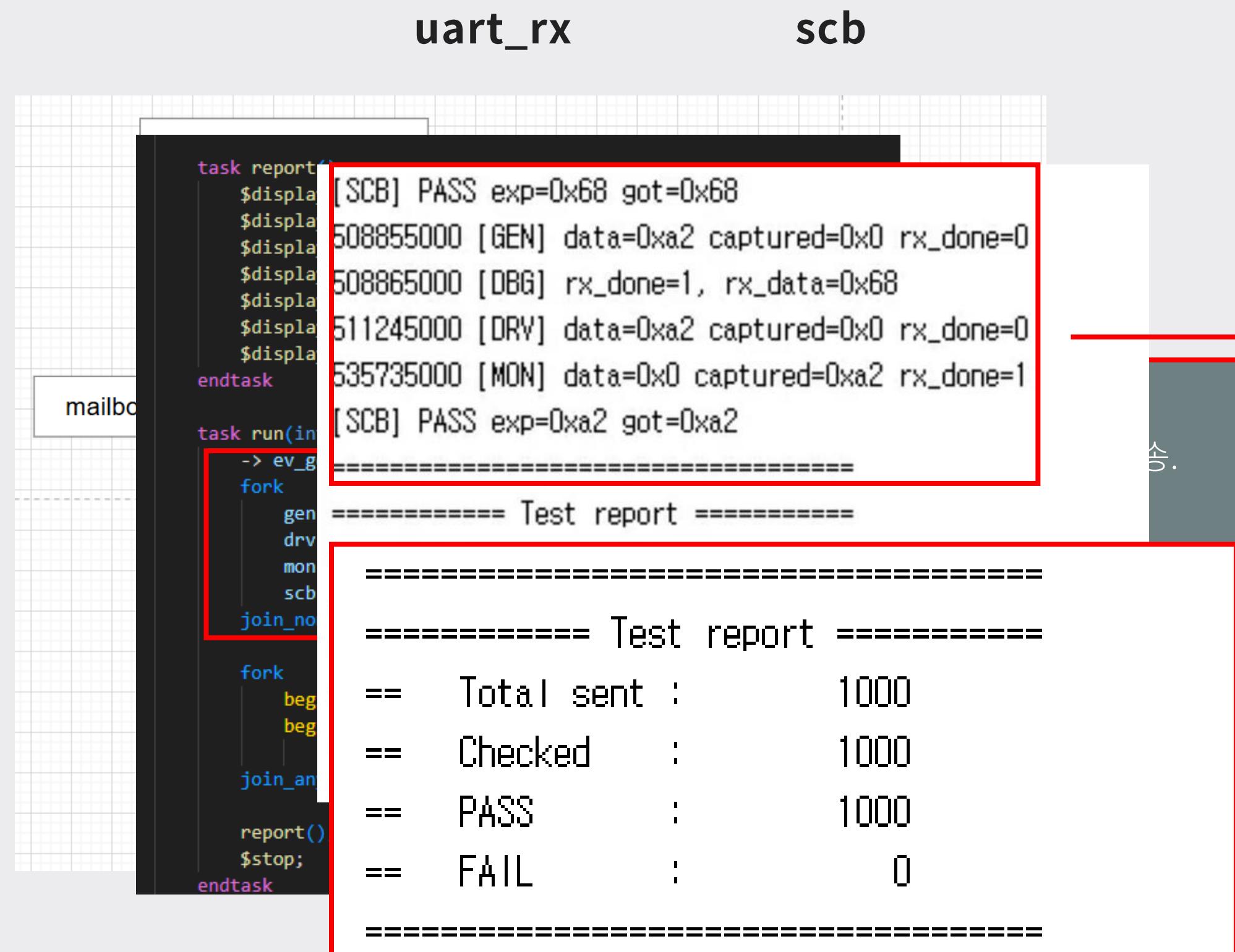
task report
$display
$display
$display
$display
$display
$display
$display
endtask

task run(in)
-> ev_g
fork
gen
drv
mon
scb
join_no
=====
===== Test report =====
== Total sent : 20
== Checked   : 20
== PASS      : 20
== FAIL      : 0
=====
join_an
$stop called at time : 535735 ns : File "D:/jihun/0912.

report();
$stop;
endtask
```



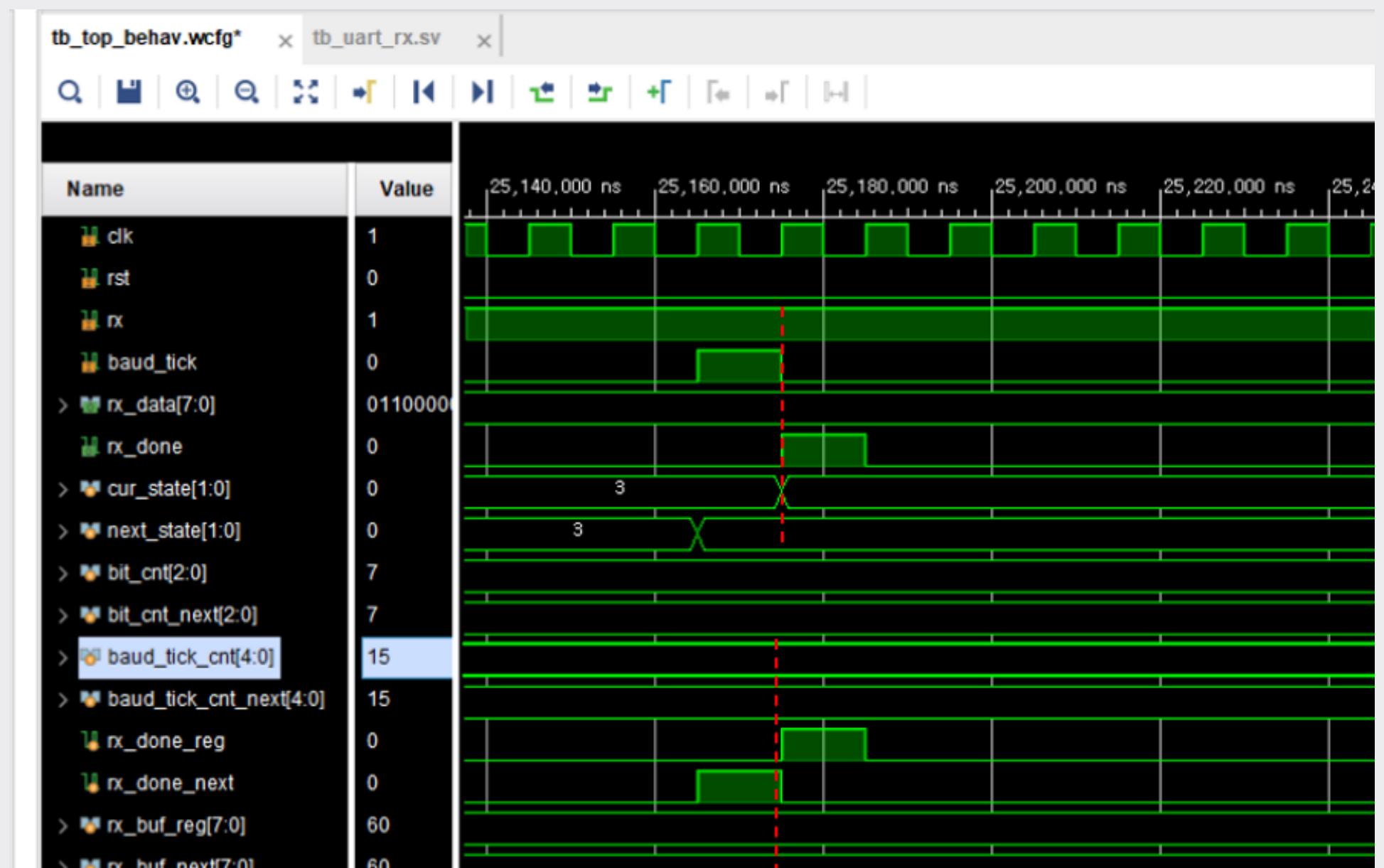
# BLOCK Diagram



- Generator가 0xA2 데이터를 Stimulus로 만들었음.
- Driver가 이를 DUT에 UART 프레임으로 전송.
- DUT가 데이터를 수신하고 rx\_done을 올림.
- Monitor가 rx\_data=0xA2를 확인해 Scoreboard로 전달.
- Scoreboard가 기대값과 실제값이 같음을 확인 → PASS 판정.

# 시뮬레이션 결과

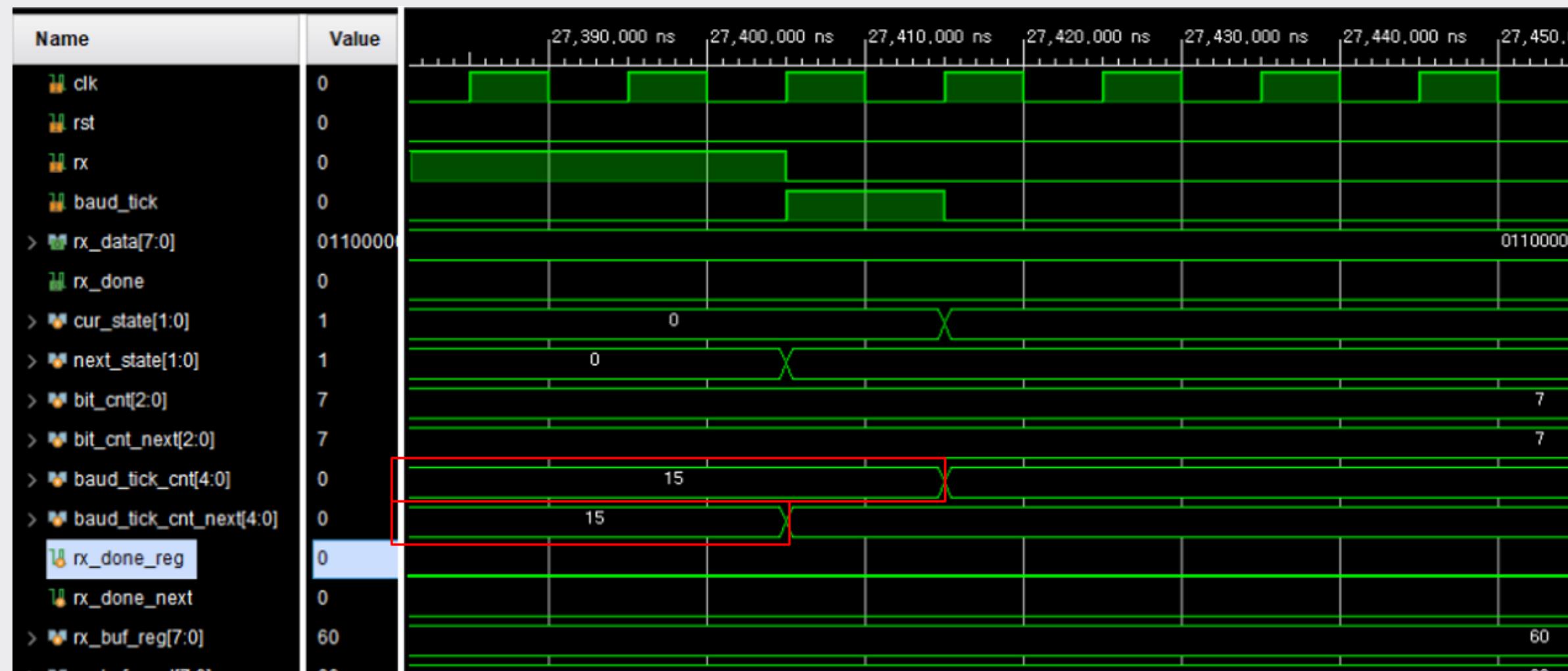
## uart\_rx



- bit\_cnt=7 → 마지막 데이터 비트를 샘플링 중.
- baud\_tick\_cnt=15 → 해당 비트 주기의 마지막 샘플링 타이밍.
- rx\_buf\_reg 값은 0x60 (0110\_0000) 으로 누적 완료.
- rx\_done=0 → 아직 수신 완료 신호는 안 올라온 상태.
- baud\_tick에서 Stop 비트를 확인하고 rx\_done이 올라갈 예정.

# 시뮬레이션 결과

## uart\_rx



- bit\_cnt=7 유지 → 이미 8비트 데이터 수신 완료.
- baud\_tick\_cnt=15 → Stop 비트 샘플링 마지막 타이밍.
- rx\_buf\_reg=0x60 → 최종 데이터가 버퍼에 저장됨.
- rx\_done\_next=1 (곧 rx\_done=1) → 수신 종료 직전.
- cur\_state/next\_state=0 → FSM이 Idle(혹은 Done) 상태로 전이 준비 중.
- 모든 데이터(0x60)를 수신 완료하고, Stop 비트 확인 중.

# BLOCK Diagram

```
uart_tx#(div) uart_tx
  task reset();
    u_if.rst      = 1;
    u_if.start_trig = 0;
    u_if.tx_data   = 0;
    repeat(2) @(posedge u_if.clk);
    u_if.rst = 0;
    repeat(2) @(posedge u_if.clk);
    $display("[DRV] Reset done");
  endtask

  task run();
    forever begin
      mbox_gen2drv.get(tr);
      while (u_if.tx_busy) @(posedge u_if.clk);
      u_if.start_trig <= tr.start_trig;
      u_if.tx_data   <= tr.tx_data;
      tr.display("DRV");
      @(posedge u_if.clk);
      u_if.start_trig <= 1'b0;
      @(posedge u_if.tx_busy);
      -> ev_mon_next;
    end
  endtask
endclass
```

- Stimulus 수신
- mbox\_gen2drv.get(tr) : Generator가 생성한 transaction(tx\_data, start\_trig)을 받음.
  - 대기
- while (u\_if.tx\_busy) : 만약 DUT가 전송 중이면 Busy가 내려올 때까지 기다림.
  - 데이터 입력
- DUT 인터페이스(tx\_data)에 전송할 데이터를 넣음.
- start\_trig을 1로 세워 송신 시작 요청.
  - Trigger 처리
- 한 클럭 유지 후 start\_trig <= 0 → 한 사이클짜리 Start 신호를 DUT에 전달.
- Busy 확인
- DUT의 tx\_busy가 올라올 때까지 기다림 → 전송 시작됐음을 보장.
  - Monitor 동기화
- -> ev\_mon\_next; : Monitor에게 “새로운 전송 시작됨”을 알림.

# BLOCK Diagram

div

uart\_tx

```
task run();
    logic [7:0] d;
    int i;
    forever begin
        @(ev_mon_next);

        tr = new();
        tr.start_trig = u_if.start_trig;
        tr.tx_data    = u_if.tx_data;

        if (u_if.tx == 1'b1) @(negedge u_if.tx);
        @(posedge u_if.baud_tick);

        repeat (OSR/2) @(posedge u_if.baud_tick);
        if (u_if.tx !== 0) $display("[MON] Start not LOW at %0t", $time);

        repeat (OSR) @(posedge u_if.baud_tick);

        d = '0;
        for (i=0; i<8; i++) begin
            d[i] = u_if.tx;
            if (i!=7) repeat (OSR) @(posedge u_if.baud_tick);
        end

        repeat (OSR) @(posedge u_if.baud_tick);
        if (u_if.tx !== 1) $display("[MON] Stop not HIGH at %0t", $time);

        tr.captured_bits = d;
        tr.display("MON");

        @(posedge u_if.clk);
        mbox_mon2scb.put(tr);
    end
endtask
endclass
```

대기

- @(ev\_mon\_next) : Driver가 Stimulus를 DUT에 보냈다는 신호(이벤트)를 받으면 시작.

# BLOCK Diagram

div

uart\_tx

```
task run();
    logic [7:0] d;
    int i;
    forever begin
        @(ev_mon_next);

        tr = new();
        tr.start_trig = u_if.start_trig;
        tr.tx_data    = u_if.tx_data;

        if (u_if.tx == 1'b1) @(negedge u_if.tx);
        @(posedge u_if.baud_tick);

        repeat (OSR/2) @(posedge u_if.baud_tick);
        if (u_if.tx !== 0) $display("[MON] Start not LOW at %0t", $time);

        repeat (OSR) @(posedge u_if.baud_tick);

        d = '0;
        for (i=0; i<8; i++) begin
            d[i] = u_if.tx;
            if (i!=7) repeat (OSR) @(posedge u_if.baud_tick);
        end

        repeat (OSR) @(posedge u_if.baud_tick);
        if (u_if.tx !== 1) $display("[MON] Stop not HIGH at %0t", $time);

        tr.captured_bits = d;
        tr.display("MON");

        @(posedge u_if.clk);
        mbox_mon2scb.put(tr);
    end
endtask
endclass
```

대기

- Start Bit 검출
  - @(negedge u\_if.tx) : tx 라인이 1 → 0으로 떨어지는 순간 (Start bit)을 감지.
  - 이후 OSR/2 (오버샘플링 비율의 절반)만큼 기다렸다가 샘플링 → Start bit가 0인지 확인.

# BLOCK Diagram

div

uart\_tx

```
task run();
    logic [7:0] d;
    int i;
    forever begin
        @(ev_mon_next);

        tr = new();
        tr.start_trig = u_if.start_trig;
        tr.tx_data    = u_if.tx_data;

        if (u_if.tx == 1'b1) @(negedge u_if.tx);
        @posedge u_if.baud_tick;

        repeat (OSR/2) @posedge u_if.baud_tick;
        if (u_if.tx !== 0) $display("[MON] Start not LOW at %0t", $time);

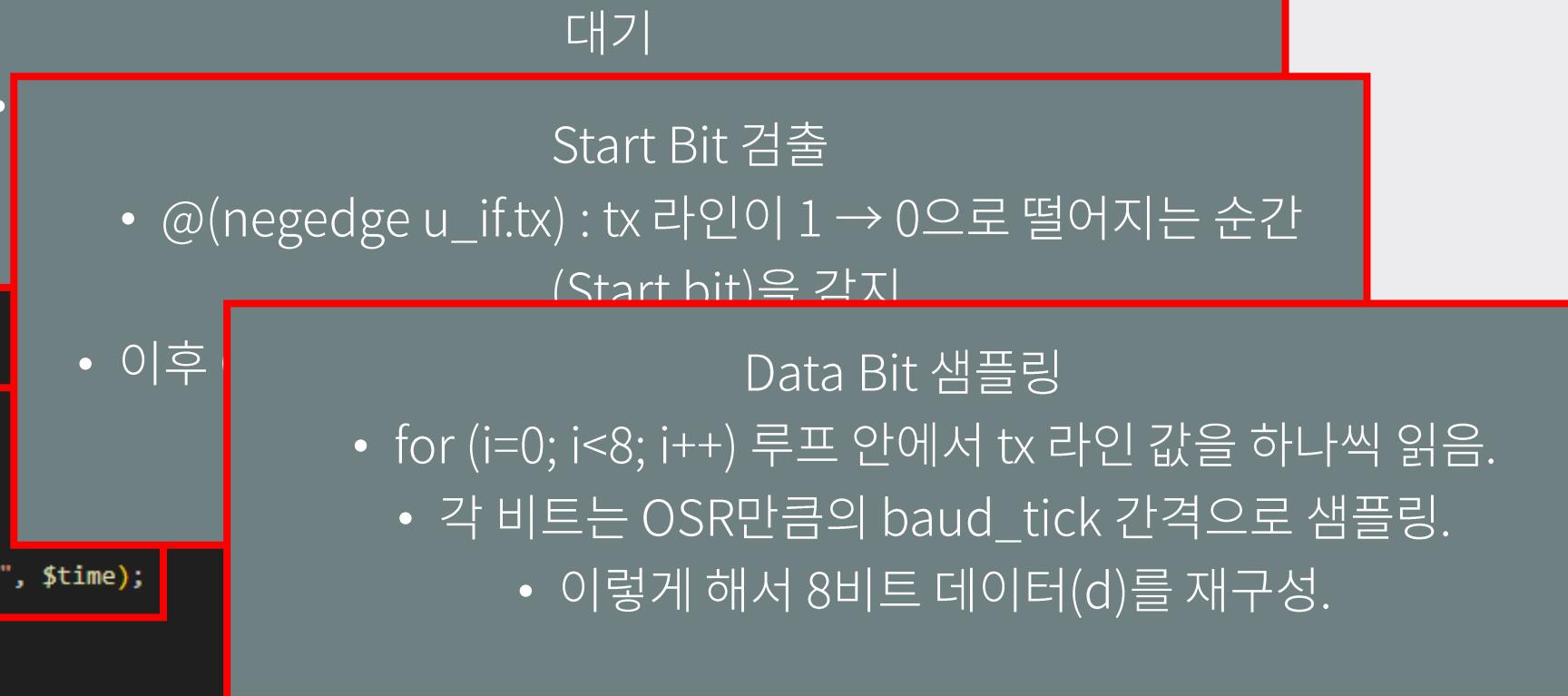
        repeat (OSR) @posedge u_if.baud_tick;

        d = '0;
        for (i=0; i<8; i++) begin
            d[i] = u_if.tx;
            if (i!=7) repeat (OSR) @posedge u_if.baud_tick;
        end

        repeat (OSR) @posedge u_if.baud_tick;
        if (u_if.tx !== 1) $display("[MON] Stop not HIGH at %0t", $time);

        tr.captured_bits = d;
        tr.display("MON");

        @(posedge u_if.clk);
        mbox_mon2scb.put(tr);
    end
endtask
endclass
```



# BLOCK Diagram

div

uart\_tx

```
task run();
    logic [7:0] d;
    int i;
    forever begin
        @(ev_mon_next);

        tr = new();
        tr.start_trig = u_if.start_trig;
        tr.tx_data    = u_if.tx_data;

        if (u_if.tx == 1'b1) @(negedge u_if.tx);
        @(posedge u_if.baud_tick);

        repeat (OSR/2) @(posedge u_if.baud_tick);
        if (u_if.tx !== 0) $display("[MON] Start not LOW at %0t", $time);

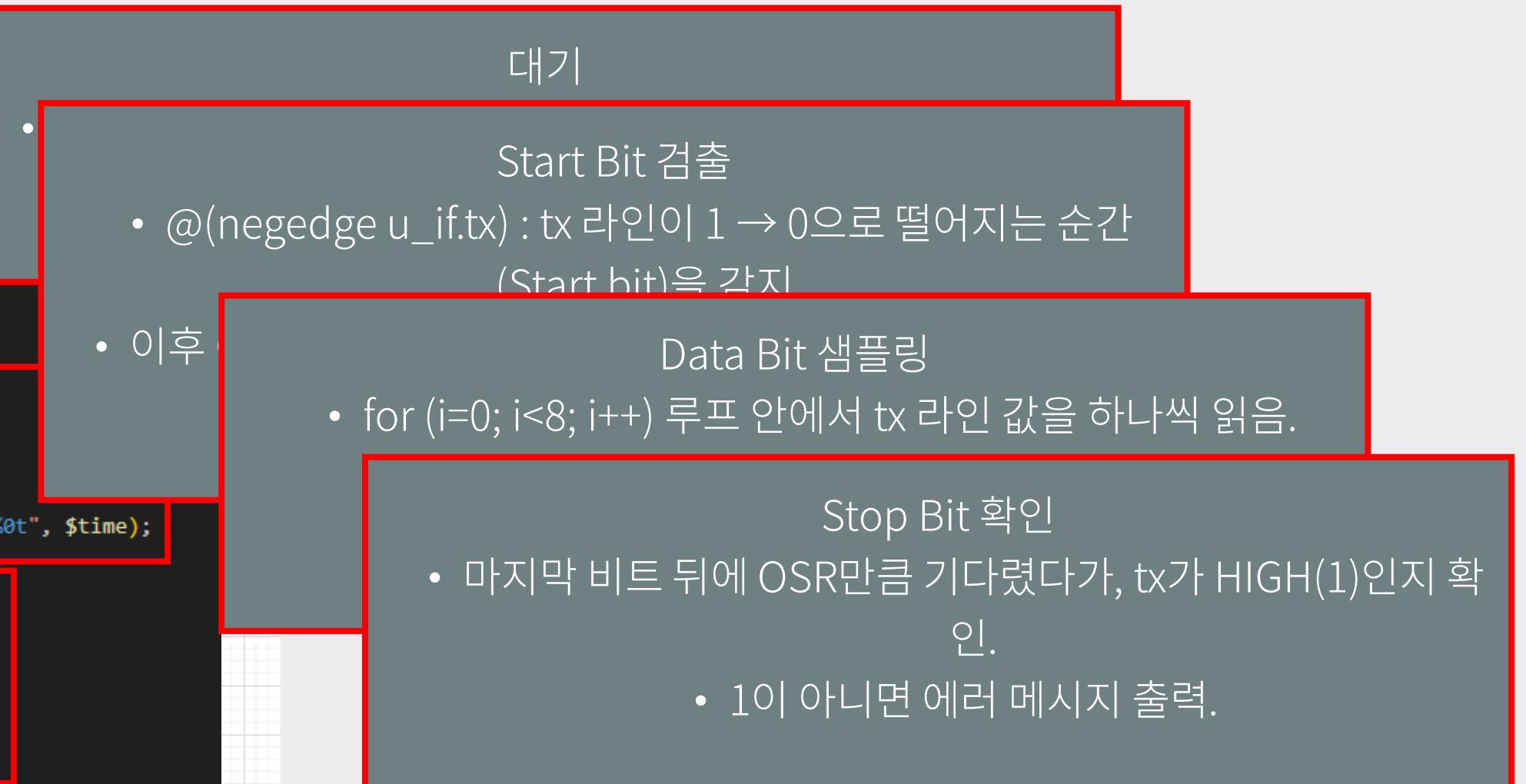
        repeat (OSR) @(posedge u_if.baud_tick);

        d = '0;
        for (i=0; i<8; i++) begin
            d[i] = u_if.tx;
            if (i!=7) repeat (OSR) @(posedge u_if.baud_tick);
        end

        repeat (OSR) @(posedge u_if.baud_tick);
        if (u_if.tx !== 1) $display("[MON] Stop not HIGH at %0t", $time);

        tr.captured_bits = d;
        tr.display("MON");

        @(posedge u_if.clk);
        mbox_mon2scb.put(tr);
    end
endtask
endclass
```



# BLOCK Diagram

div

uart\_tx

```
task run();
    logic [7:0] d;
    int i;
    forever begin
        @(ev_mon_next);

        tr = new();
        tr.start_trig = u_if.start_trig;
        tr.tx_data    = u_if.tx_data;

        if (u_if.tx == 1'b1) @(negedge u_if.tx);
        @posedge u_if.baud_tick;

        repeat (OSR/2) @posedge u_if.baud_tick;
        if (u_if.tx !== 0) $display("[MON] Start not LOW at %0t", $time);

        repeat (OSR) @posedge u_if.baud_tick;

        d = '0;
        for (i=0; i<8; i++) begin
            d[i] = u_if.tx;
            if (i!=7) repeat (OSR) @posedge u_if.baud_tick;
        end

        repeat (OSR) @posedge u_if.baud_tick;
        if (u_if.tx !== 1) $display("[MON] Stop not HIGH at %0t", $time);

        tr.captured_bits = d;
        tr.display("MON");

        @(posedge u_if.clk);
        mbox_mon2scb.put(tr);
    end
endtask
endclass
```

대기

Start Bit 검출

- @(negedge u\_if.tx) : tx 라인이 1 → 0으로 떨어지는 순간  
(Start bit)을 간지

이후

Data Bit 샘플링

- for (i=0; i<8; i++) 루프 안에서 tx 라인 값을 하나씩 읽음.

Stop Bit 확인

- 마지막 비트 뒤에 OSR만큼 기다렸다가, tx가 HIGH(1)인지 확인.

결과 전달

- tr.captured\_bits = d : 모니터링한 데이터를 transaction에 저장.
  - Scoreboard로 mbox\_mon2scb.put(tr) 전송.
    - 로그 출력(MON).

# BLOCK Diagram

div

uart\_tx

```
task run();
    int i;
    bit all_match;
    forever begin
        mbox_mon2scb.get(tr);
        all_match = 1;
        for (i=0; i<8; i++) begin
            if (tr.captured_bits[i] != tr.tx_data[i]) begin
                all_match = 0;
                $display("[SCB] BIT MISMATCH @%0d exp=%0b got=%0b",
                         i, tr.tx_data[i], tr.captured_bits[i]);
            end
        end
        total_cnt++;
        if (all_match) begin
            pass_cnt++;
            $display("[SCB] PASS exp=0x%0h got=0x%0h",
                     tr.tx_data, tr.captured_bits);
        end else begin
            fail_cnt++;
            $display("[SCB] FAIL exp=0x%0h got=0x%0h",
                     tr.tx_data, tr.captured_bits);
        end
        $display("-----");
        -> ev_gen_next;
    end
endtask
```

mailbox

Monitor → Scoreboard 데이터 전달

- mbox\_mon2scb.get(tr) : Monitor가 보낸 transaction을 수신.
  - 이 안에는 tx\_data(Generator/Driver가 보낸 값), captured\_bits(DUT에서 실제 나온 값)가 들어 있음.

# BLOCK Diagram

div

uart\_tx

```
task run();
    int i;
    bit all_match;
    forever begin
        mbox_mon2scb.get(tr);
        all_match = 1;
        for (i=0; i<8; i++) begin
            if (tr.captured_bits[i] != tr.tx_data[i]) begin
                all_match = 0;
                $display("[SCB] BIT MISMATCH @%0d exp=%0b got=%0b",
                         i, tr.tx_data[i], tr.captured_bits[i]);
            end
        end
        total_cnt++;
        if (all_match) begin
            pass_cnt++;
            $display("[SCB] PASS exp=0x%0h got=0x%0h",
                     tr.tx_data, tr.captured_bits);
        end else begin
            fail_cnt++;
            $display("[SCB] FAIL exp=0x%0h got=0x%0h",
                     tr.tx_data, tr.captured_bits);
        end
        $display("-----");
        -> ev_gen_next;
    end
endtask
```

Monitor → Scoreboard 데이터 전달

- mbox\_mon2scb.get(tr) : Monitor가 보낸 transaction을 수신.
  - 이 안에는 tx\_data(Generator/Driver가 보낸 값), captured\_bits(DUT에서 실제 나온 값)가 들어 있음.

비트 단위 비교

- for (i=0; i<8; i++) 루프에서 8비트를 하나씩 비교.
  - 다르면 BIT MISMATCH 메시지 출력.

# BLOCK Diagram

div

uart\_tx

```
task run();
    int i;
    bit all_match;
    forever begin
        mbox_mon2scb.get(tr);
        all_match = 1;
        for (i=0; i<8; i++) begin
            if (tr.captured_bits[i] != tr.tx_data[i]) begin
                all_match = 0;
                $display("[SCB] BIT MISMATCH @%0d exp=%0b got=%0b",
                         i, tr.tx_data[i], tr.captured_bits[i]);
            end
        end
        total_cnt++;
        if (all_match) begin
            pass_cnt++;
            $display("[SCB] PASS exp=0x%0h got=0x%0h",
                     tr.tx_data, tr.captured_bits);
        end else begin
            fail_cnt++;
            $display("[SCB] FAIL exp=0x%0h got=0x%0h",
                     tr.tx_data, tr.captured_bits);
        end
        $display("-----");
        -> ev_gen_next;
    end
endtask
```

- Monitor → Scoreboard 데이터 전달
  - mbox\_mon2scb.get(tr) : Monitor가 보낸 transaction을 수신.
  - 이 안에는 tx\_data(Generator/Driver가 보낸 값), captured\_bits(DUT에서 실제 나온 값)가 들어 있음.
- 비트 단위 비교
  - for (i=0; i<8; i++) 루프에서 8비트를 하나씩 비교.
  - 다르면 BIT MISMATCH 메시지 출력.
- all\_match == 1이면 PASS → 기대값과 실제값 동일.
  - 아니면 FAIL → 기대값과 불일치.
  - PASS/FAIL 카운트 업데이트.

# BLOCK Diagram

uart\_tx      scb

```
-----  
485295000 [GEN] : start=1, tx_data=0x3f, captured=0xxx  
486435000 [DRV] : start=1, tx_data=0x3f, captured=0xxx  
510885000 [MON] : start=0, tx_data=0x3f, captured=0x3f  
[SCB] PASS exp=0x3f got=0x3f  
-----
```

```
===== Test report =====
```

```
-- Total sent : 20  
-- Checked    : 20  
-- PASS       : 20  
-- FAIL       : 0  
=====
```

```
$stop called at time : 510895 ns : File "D:/jihun/0912_uart
```

# BLOCK Diagram

scb

uart\_tx

th\_uart\_tx

```
-----  
485295000 [GEN] : start=1, tx_data=0x3f, captured=0xxx  
486435000 [DRV] : start=1, tx_data=0x3f, captured=0xxx  
510885000 [MON] : start=0, tx_data=0x3f, captured=0x3f  
[SCB] PASS exp=0x3f got=0x3f  
-----
```

```
===== Test report =====
```

```
-- Total sent : 20  
-- Checked : 20  
-- PASS : 20  
-- FAIL : 0  
=====
```

\$stop called at time : 510895 ns : File "D:/jihun/0912\_uart

개별 단계 출력

# BLOCK Diagram

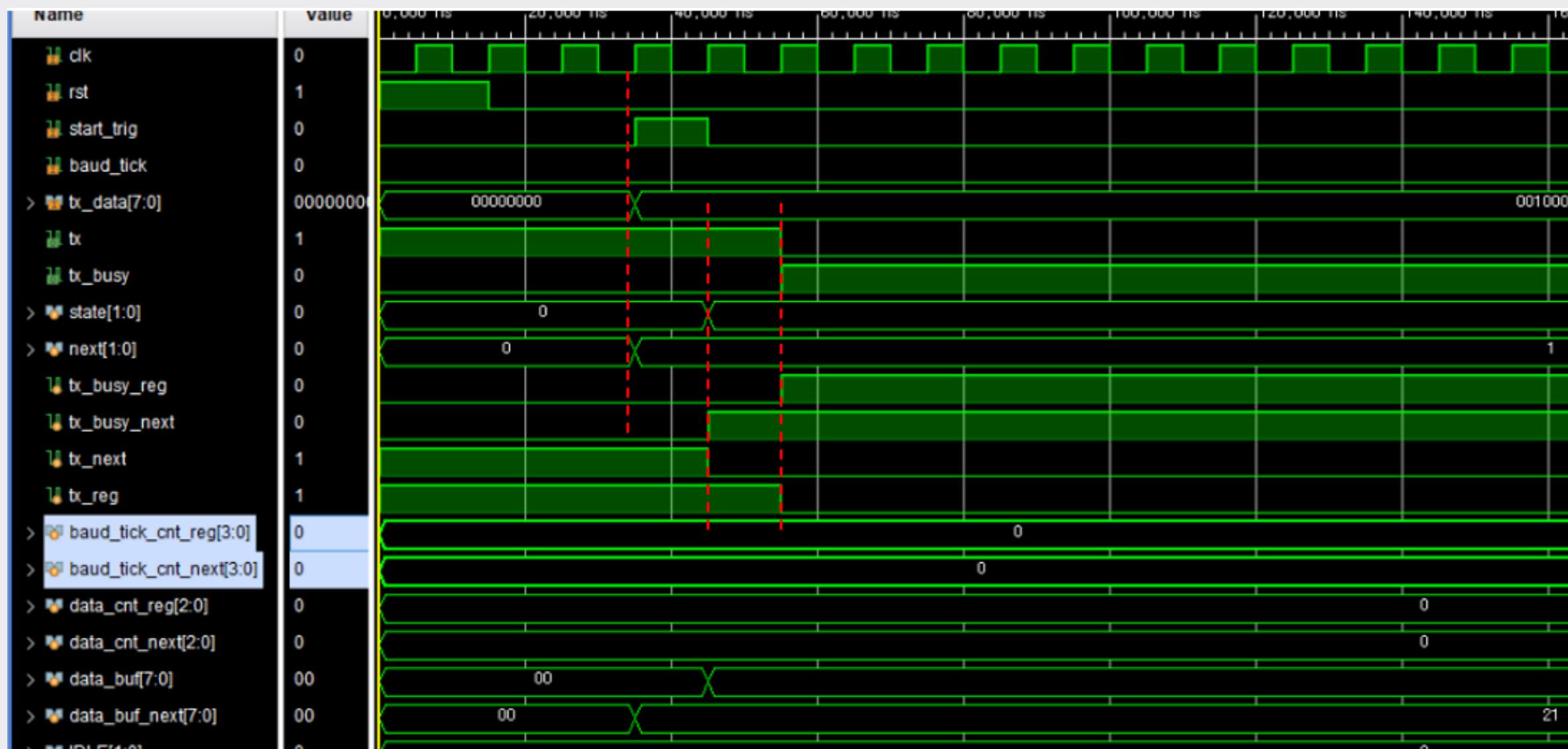
```
uart_tx          scb
-----  
485295000 [GEN] : start=1, tx_data=0x3f, captured=0xxx  
486435000 [DRV] : start=1, tx_data=0x3f, captured=0xxx  
510885000 [MON] : start=0, tx_data=0x3f, captured=0x3f  
[SCB] PASS exp=0x3f got=0x3f  
-----  
===== Test report =====  
== Total sent : 1000  
== Checked : 1000  
== PASS : 1000  
== FAIL : 0  
$stop  
=====
```

## 개별 단계 출력

- [GEN] : Generator에서 stimulus 생성 → start=1, tx\_data=0x3f
- [DRV] : Driver가 DUT에 데이터를 전달 → 동일하게 start=1, tx\_data=0x3f
- [MON] : Monitor가 DUT 출력에서 데이터를 캡처 → captured=0x3f (정확히 재구성됨)
- [SCB] PASS exp=0x3f got=0x3f
- → Scoreboard에서 기대값과 실제 DUT 출력이 일치한다고 확인.

# 시뮬레이션 결과

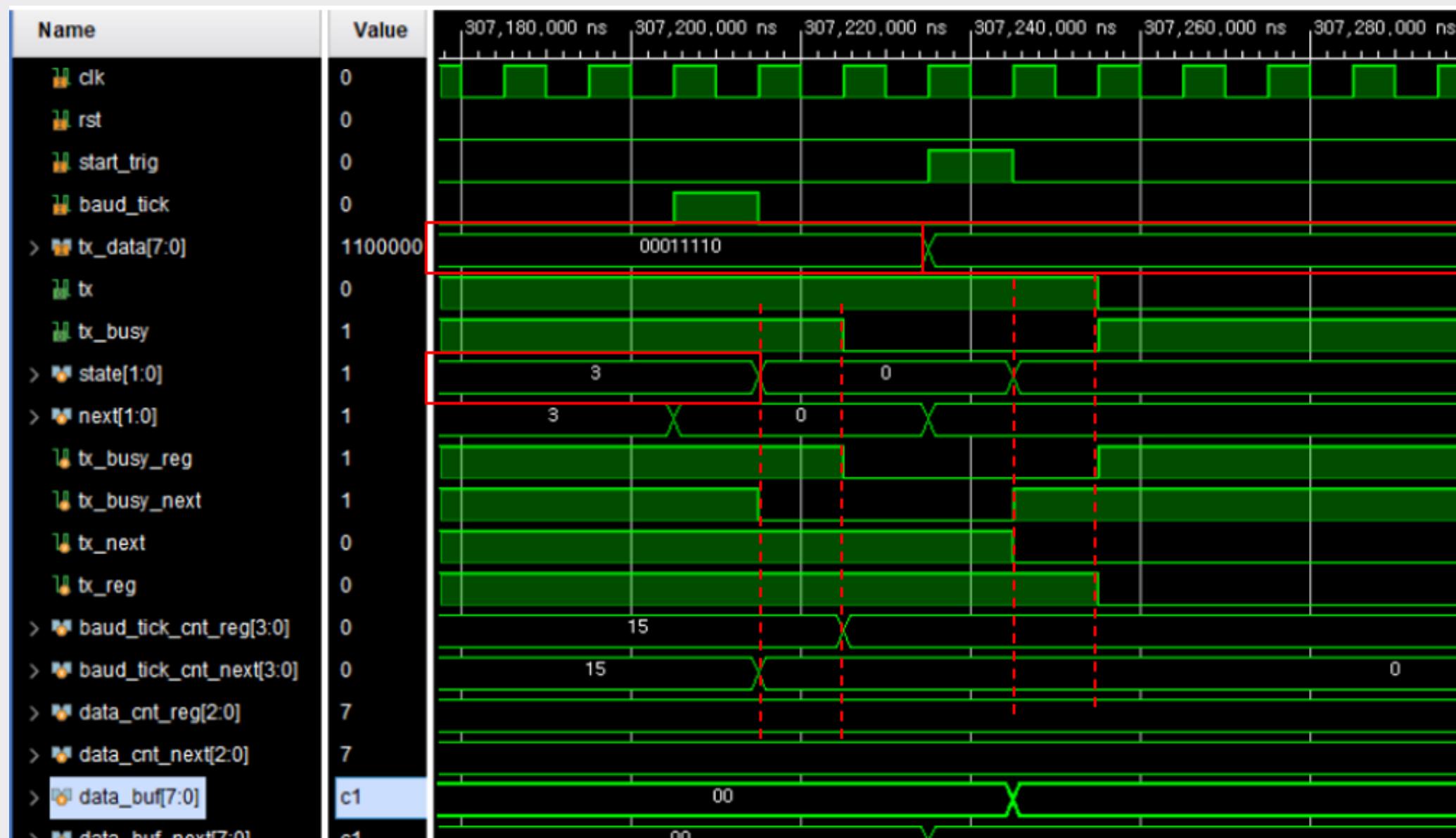
## uart\_tx



- tx\_data = 0x00 으로 설정된 상태에서 전송 시작.
- start\_trig 이 올라가면서 tx\_busy=1 → 송신 동작 시작됨.
- FSM state 가 Idle(0) → Start(1) 로 전이.
- tx 라인이 LOW(0)로 내려가며 Start bit 전송.
- baud\_tick\_cnt 0 → 비트 샘플링 카운터 초기화됨.

# 시뮬레이션 결과

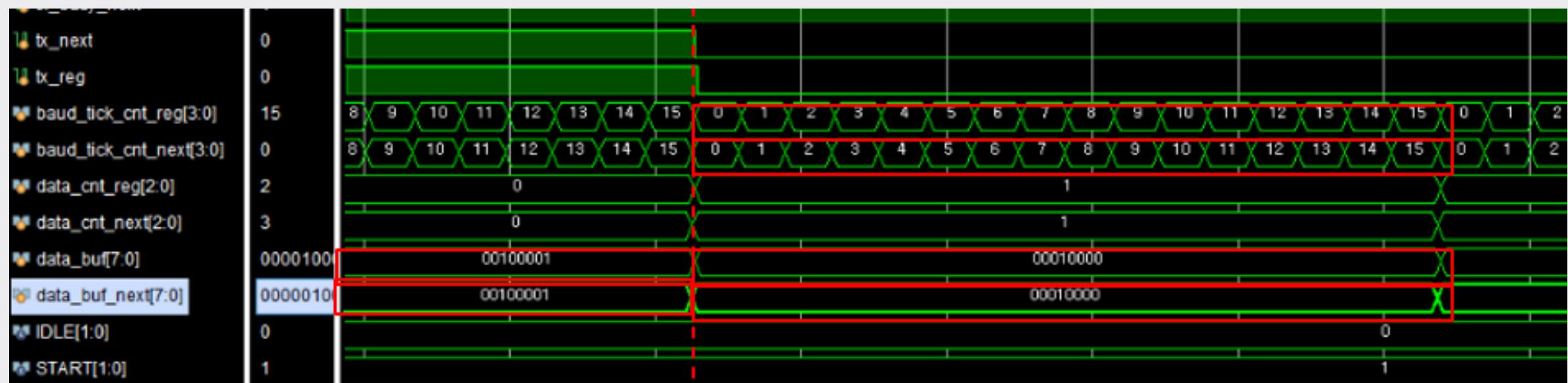
## uart\_tx



- FSM state=3 → Data 전송 상태.
- bit\_cnt=7 까지 올라감 → 마지막 데이터 비트를 내보내는 중.
- tx 라인이 데이터 비트 패턴(0과 1)을 순차적으로 내보냄.
- baud\_tick\_cnt=15 에서 비트 주기가 끝나고 다음 비트로 넘어감

# 시뮬레이션 결과

## uart\_tx



- baud\_tick\_cnt: 각 비트 구간을 16틱으로 나눠 카운트 → 타이밍 검증에 중요.
- data\_cnt: 8비트 카운트 끝나고 Stop 비트 처리 중.
- data\_buf: 실제 전송되는 데이터 값이 0x04 → 0x02로 이동.

# 트러블슈팅&고찰



## Baud Tick 타이밍 문제

- 처음에는 baud\_tick(오버샘플링 카운터)과 bit\_cnt의 싱크가 어긋나는 경우가 있었음.
- baud\_tick\_cnt가 정확히 15까지 세고 리셋되지 않으면 → 다음 비트 샘플링이 밀림.
- 해결: 카운터 리셋 조건을 Start/Stop 구간 포함해서 확실히 걸어줌.



## Stop Bit 확인

- Stop bit가 제대로 1로 유지되는지 체크 안 하면, 잘못된 데이터도 PASS로 나오는 문제가 발생.
- 해결: Monitor에서 Stop bit를 OSR만큼 샘플링 → 1이 아닐 시 Error 출력.

## Start Bit 검출



- rx 라인을 너무 일찍 샘플링하면 잡음(noise) 때문에 잘못된 Start bit로 인식하는 문제가 있었음.
- 해결: Start bit 검출 후 OSR/2 (반비트 시간) 만큼 기다렸다가 첫 샘플링.

```
485815000 [GEN] data=0xa2 captured=0x0 rx_done=0
485825000 [DBG] rx_done=1, rx_data=0x68
486925000 [DRV] data=0xa2 captured=0x0 rx_done=0
511415000 [MON] data=0x0 captured=0xa2 rx_done=1
[SCB] FAIL exp=0x45 got=0xa2
=====
===== Test report =====
== Total sent : 20
== Checked    : 20
== PASS       : 1
== FAIL        : 19
=====
```

# 트러블슈팅&고찰

## Baud Tick 타이밍 문제



- 처음에는 baud\_tick(오버샘플링 카운터)과 bit\_cnt의 싱크가 어긋나는 경우가 있었음.
- baud\_tick\_cnt가 정확히 15까지 세고 리셋되지 않으면 → 다음 비트 샘플링이 밀림.
- 해결: 카운터 리셋 조건을 Start/Stop 구간 포함해서 확실히 걸어줌.

## Start Bit 검출



- rx 라인을 너무 일찍 샘플링하면 잡음(noise) 때문에 잘못된 Start bit로 인식하는 문제가 있었음.
- 해결: Start bit 검출 후 OSR/2 (반비트 시간) 만큼 기다렸다가 첫 샘플링.

## Stop Bit 확인



- Stop bit가 제대로 1로 유지되는지 체크 안 하면, 잘못된 데이터도 PASS로 나오는 문제가 발생.
- 해결: Monitor에서 Stop bit를 OSR만큼 샘플링 → 1이 아닐 시 Error 출력.

```
10001000 10001000 10001000 10001000 10001000  
reg [3:0] cnt;  
always @(posedge u_if.clk or posedge u_if.rst) begin  
    if (u_if.rst) begin  
        cnt <= 0;  
        u_if.baud_tick <= 0;  
    end else begin  
        if (cnt == 15) begin  
            cnt <= 0;  
            u_if.baud_tick <= 1;  
        end else begin  
            cnt <= cnt + 1;  
            u_if.baud_tick <= 0;  
        end  
    end  
end
```

# 트러블슈팅&고찰



## Baud Tick 타이밍 문제

- 처음에는 baud\_tick(오버샘플링 카운터)과 bit\_cnt의 싱크가 어긋나는 경우가 있었음.
- baud\_tick\_cnt가 정확히 15까지 세고 리셋되지 않으면 → 다음 비트 샘플링이 밀림.
- 해결: 카운터 리셋 조건을 Start/Stop 구간 포함해서 확실히 걸어줌.



## Start Bit 검출

- rx 라인을 너무 일찍 샘플링하면 잡음(noise) 때문에 잘못된 Start bit로 인식하는 문제가 있었음.
- 해결: Start bit 검출 후 OSR/2 (반비트 시간) 만큼 기다렸다가 첫 샘플링.



## Stop Bit 확인

- Stop bit가 제대로 1로 유지되는지 체크 안 하면, 잘못된 데이터도 PASS로 나오는 문제가 발생.
- 해결: Monitor에서 Stop bit를 OSR만큼 샘플링 → 1이 아닐 시 Error 출력.

```
10001000 10001000 10001000 10001000  
reg [3:0] d;
always @(posedge u_if.tx)
begin
    if (u_if.tx == 0) begin
        repeat (OSR/2) @ (posedge u_if.baud_tick);
        if (u_if.tx != 0) $display("[MON] Start not LOW at %t", $time);
    end
    else begin
        if (u_if.tx == 1) begin
            repeat (OSR) @ (posedge u_if.baud_tick);
            if (u_if.tx == 0) begin
                d = '0;
                for (i=0; i<8; i++) begin
                    if (u_if.tx == 1) begin
                        d[i] = u_if.tx;
                    end
                end
            end
            repeat (OSR) @ (posedge u_if.baud_tick);
            if (u_if.tx == 0) begin
                $display("[MON] Stop not HIGH at %t", $time);
            end
        end
    end
end
end
```

# FIFO 검증



# FIFO 검증

등수가 정해진 육상에서는 2등은 1등을 제칠 수 없다.



R\_addr  
후발주자



W\_addr  
선발주자

# FIFO 검증

EMPTY!!! 더 이상 따라잡으면 안된다!!



R\_addr  
후발주자

W\_addr  
선발주자

# FIFO 검증

FULL!!! 더 이상 차이가 나면 안된다!!



R\_addr

후발주자



W\_addr  
선발주자

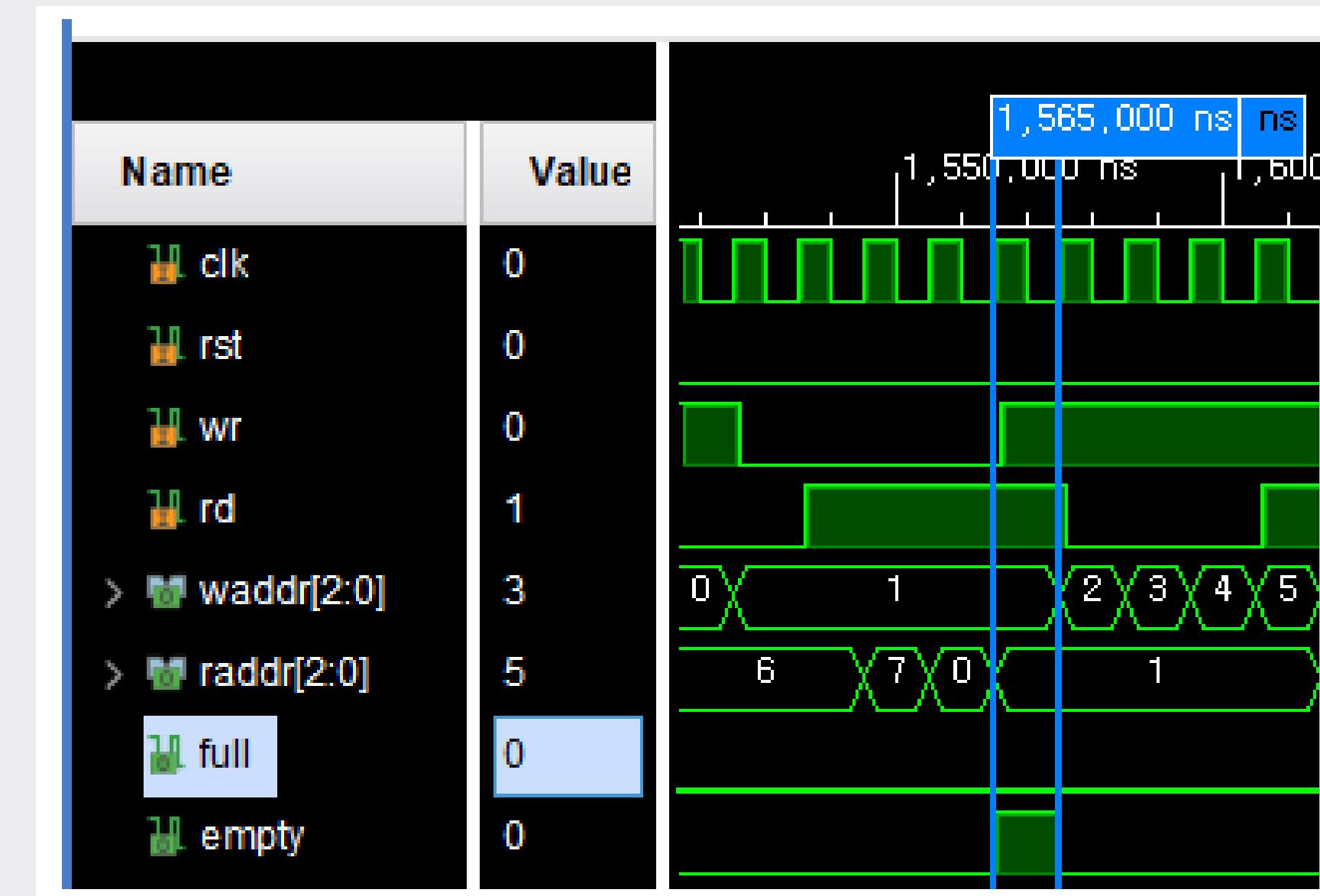
# FIFO 검증

EMPTY → 따라잡으면 안된다

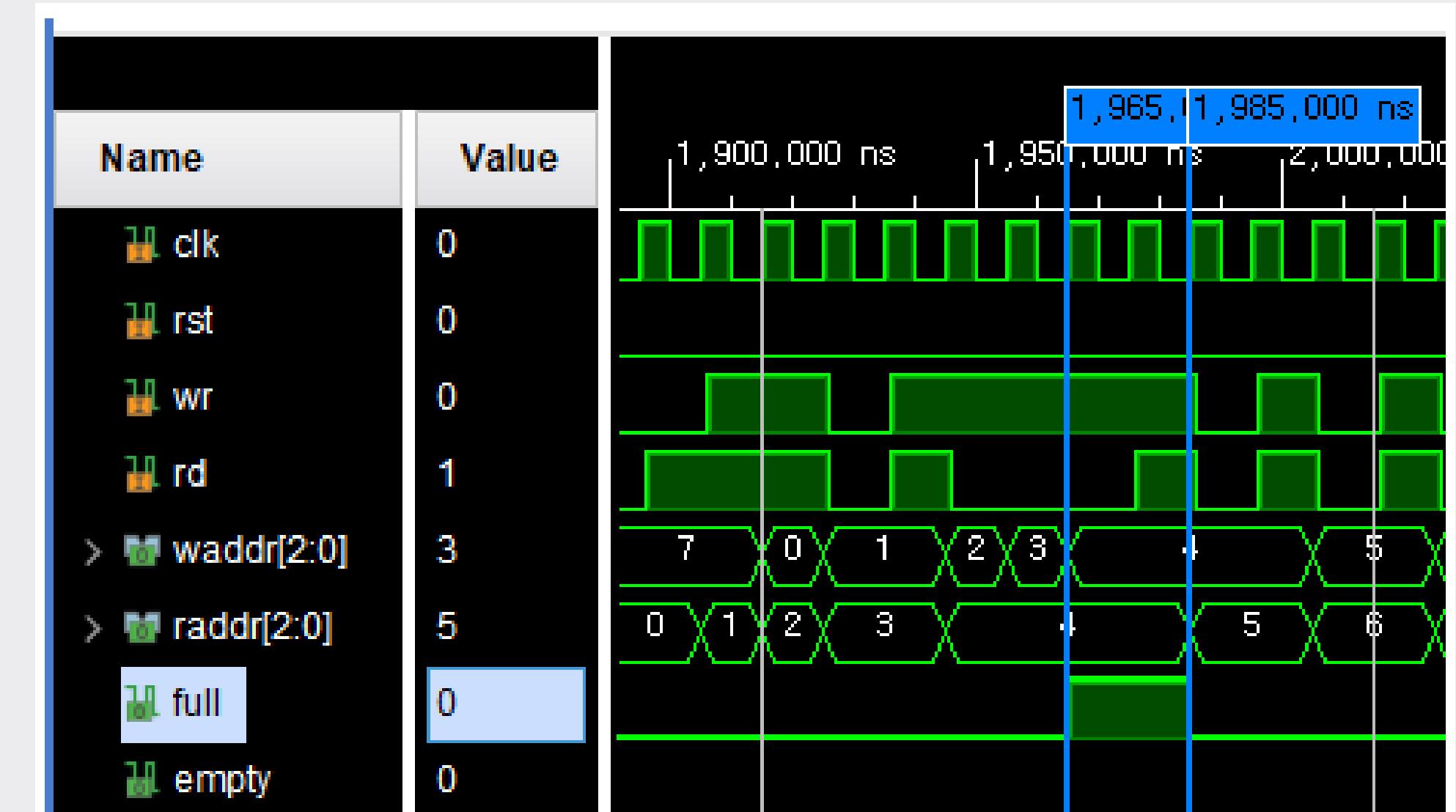


W\_addr  
선발주자

R\_addr  
후발주자



# FIFO 검증



FULL → 트랙 한 바퀴 이상 차이!



R\_addr  
후발주자



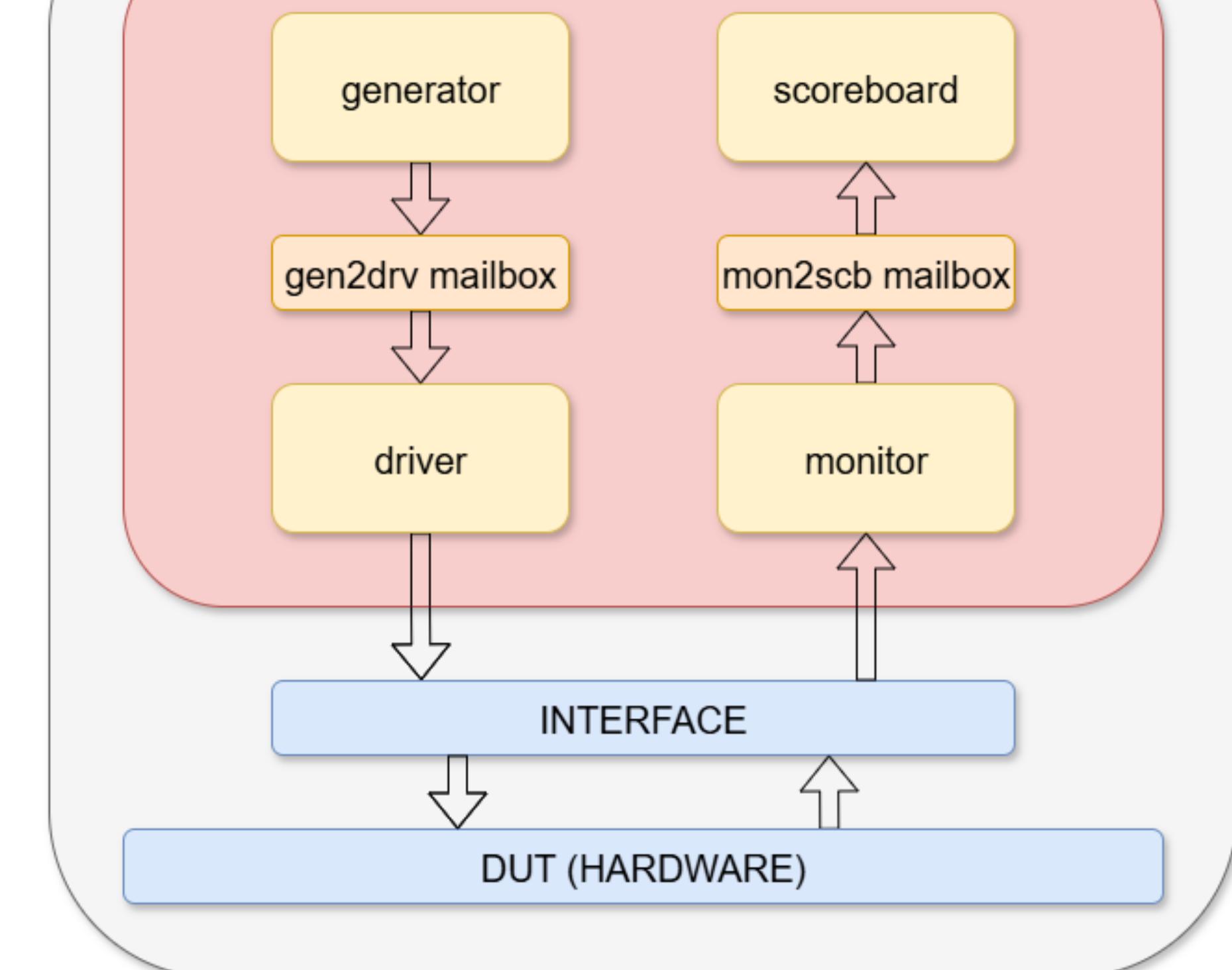
W\_addr  
선발주자

# FIFO 검증

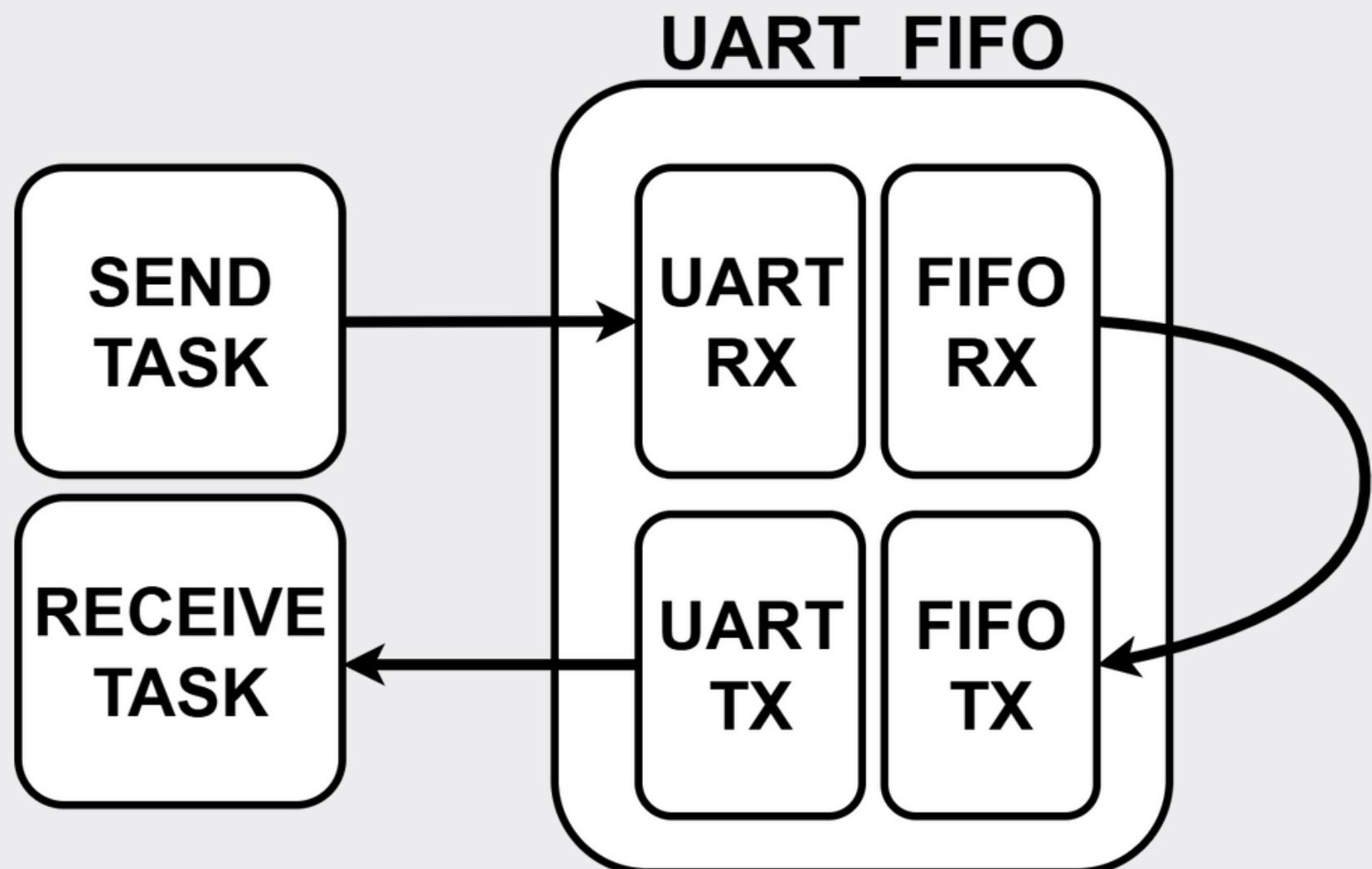
우리가 집중할 것은 임계점이다.  
넘어가면 안되는 순간에  
assertion 을 이용해  
변수가 잘못된 값으로 가는 순간을  
캐치할 수 있다!!

## Testbench

### environment

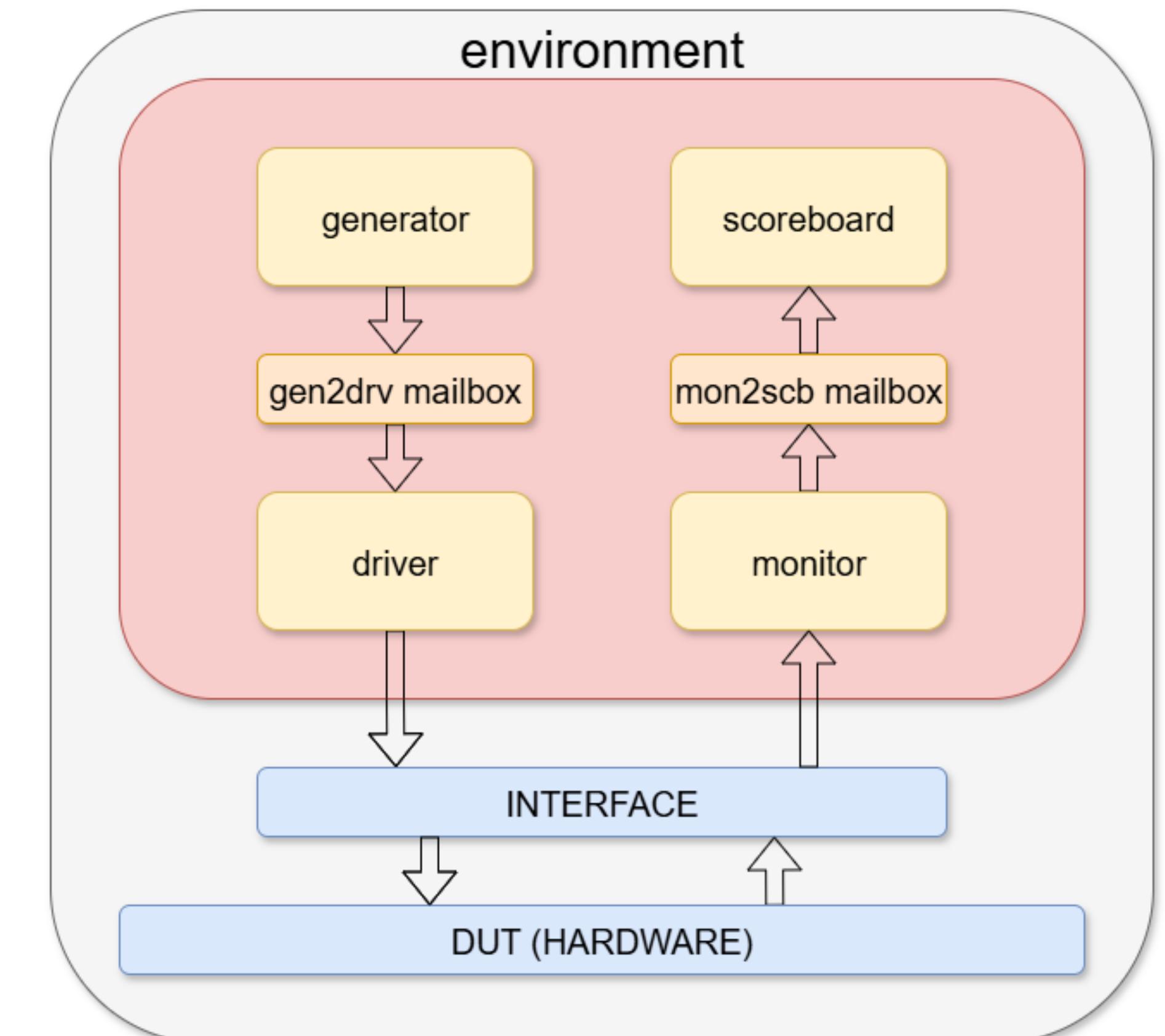


# UART\_FIFO 검증



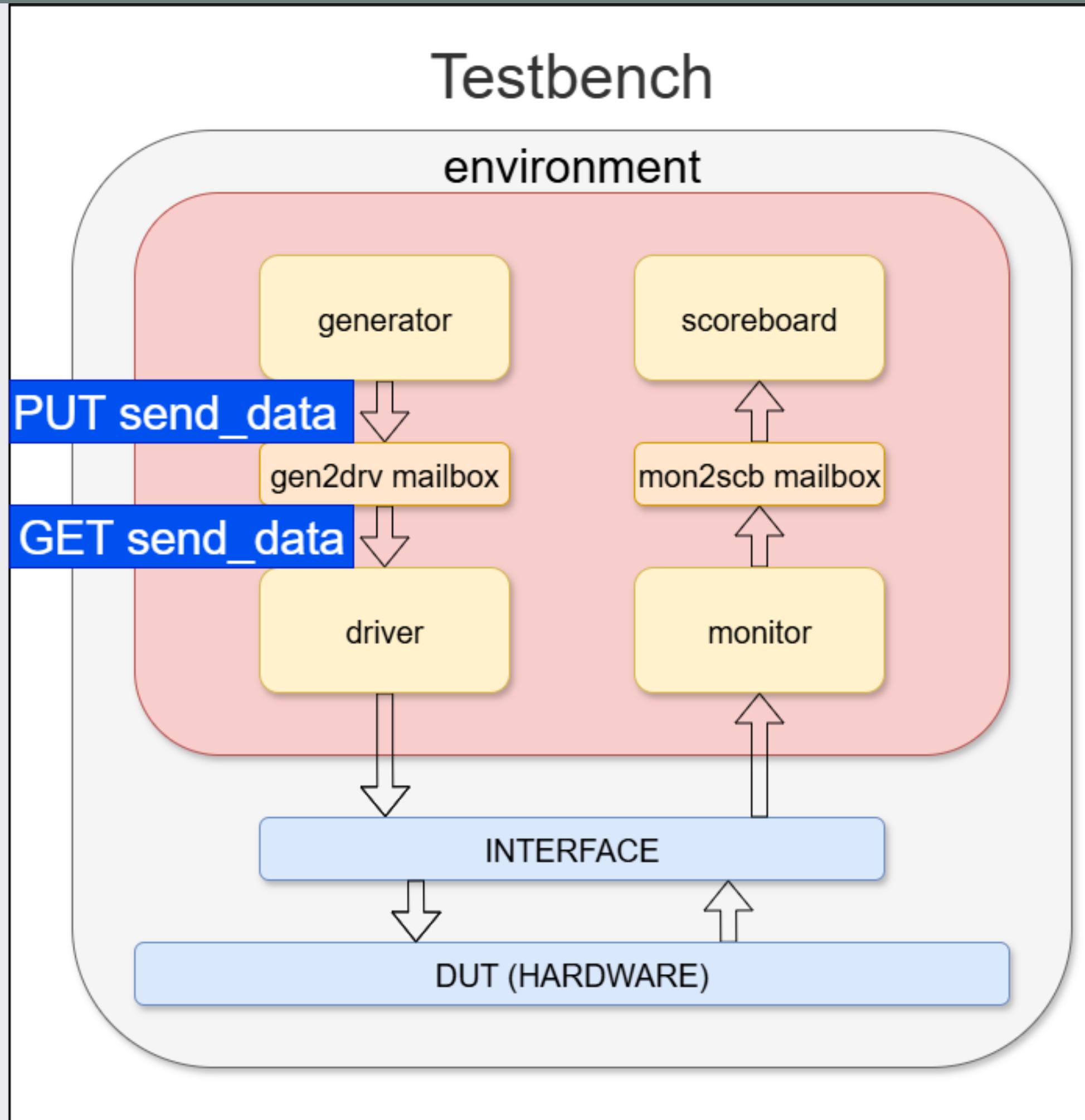
## Testbench

environment



# UART\_FIFO 검증

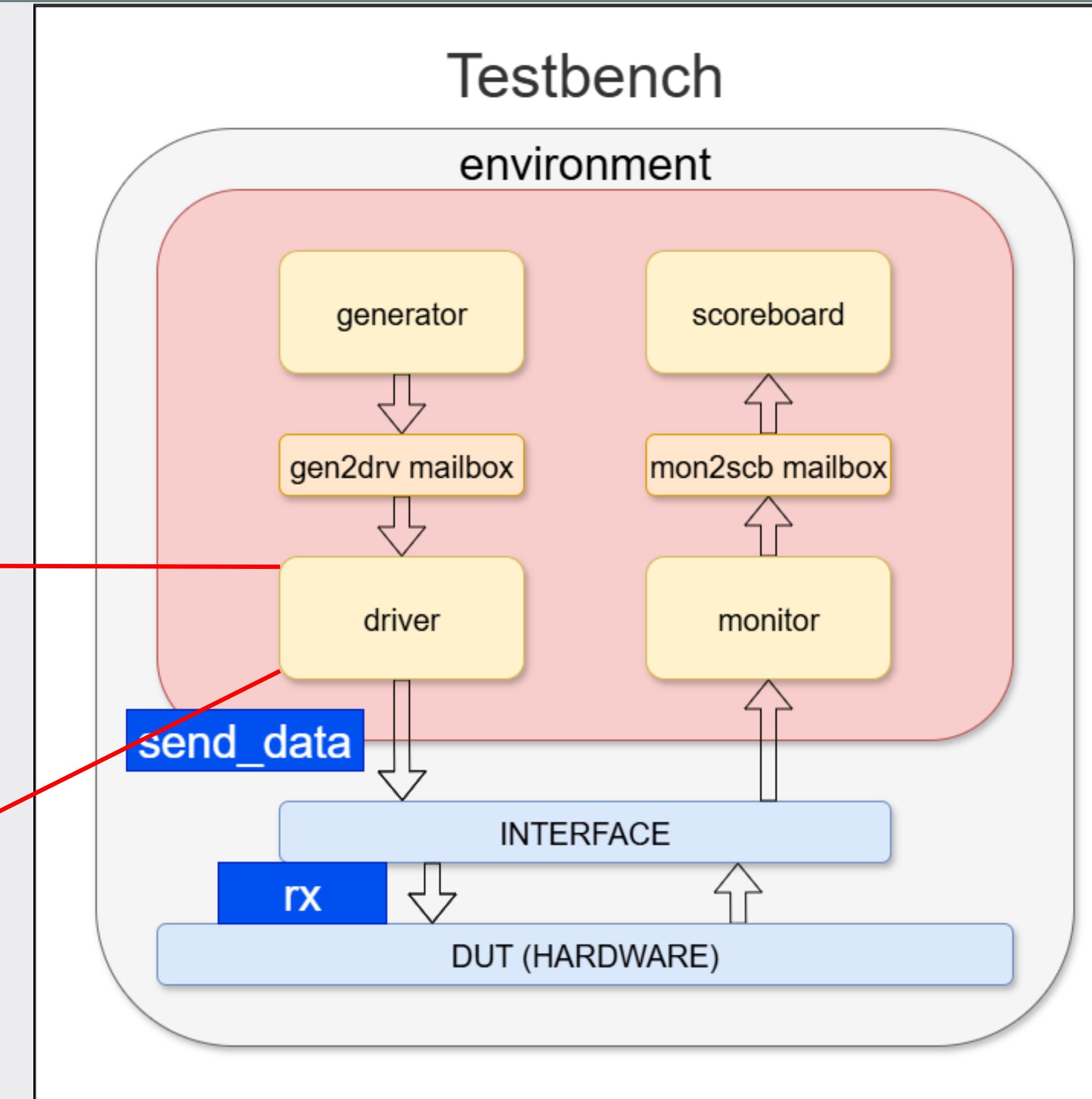
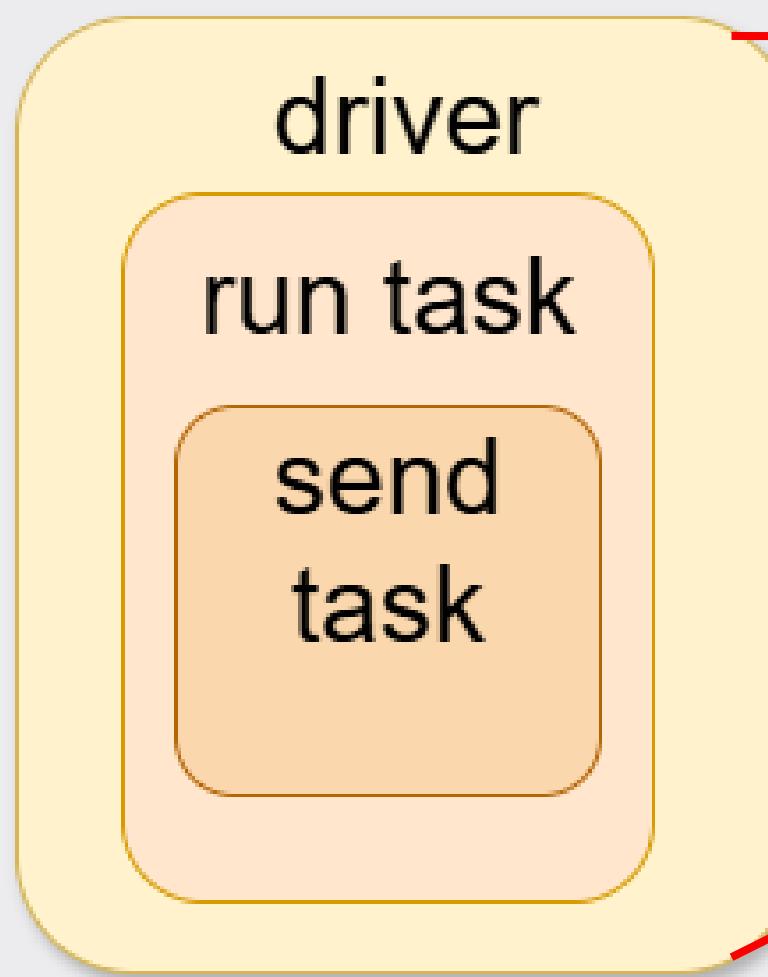
generator에서 생성한 **send\_data**를  
mailbox를 통해 driver로 보내준다.



# UART\_FIFO 검증

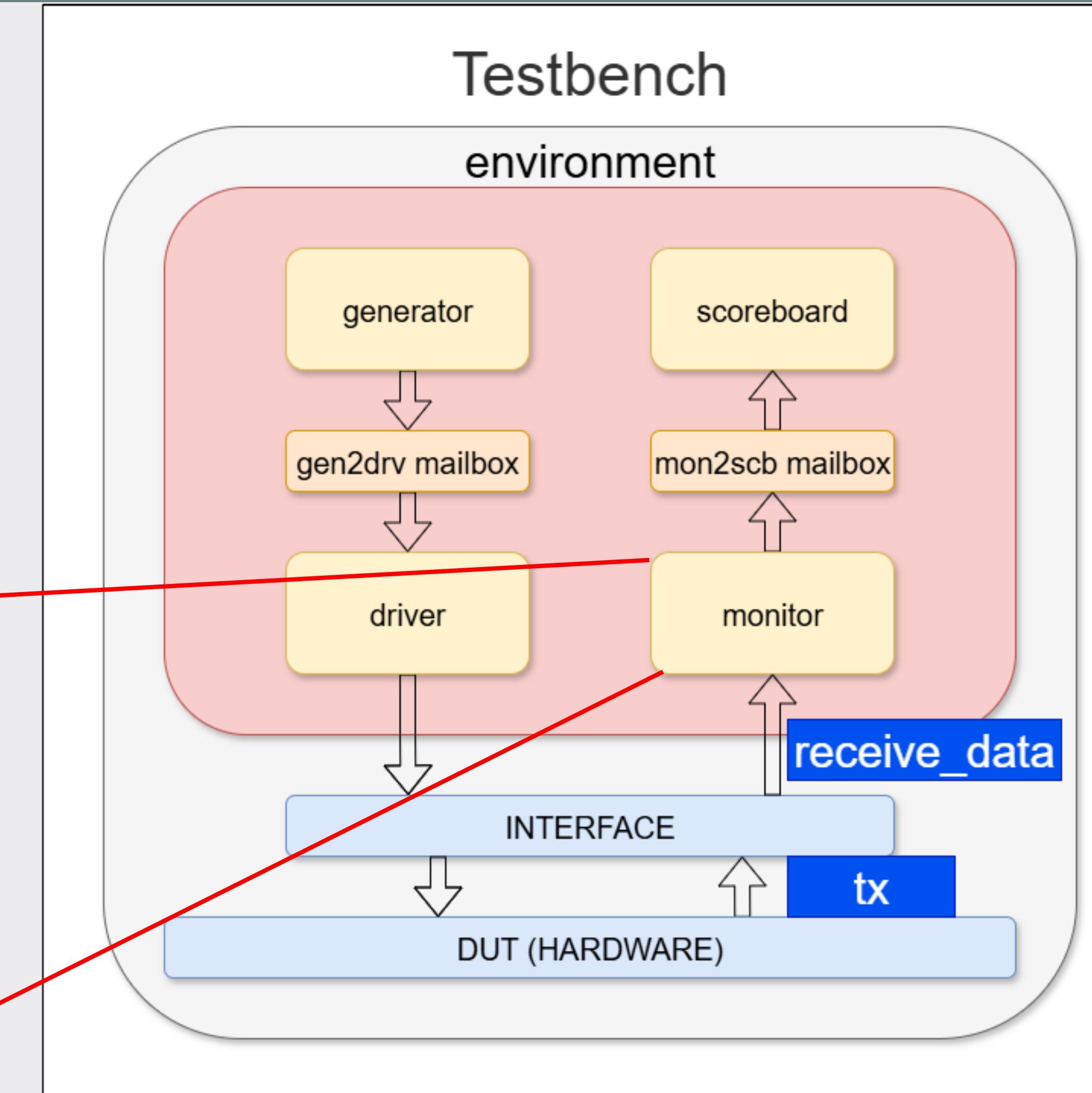
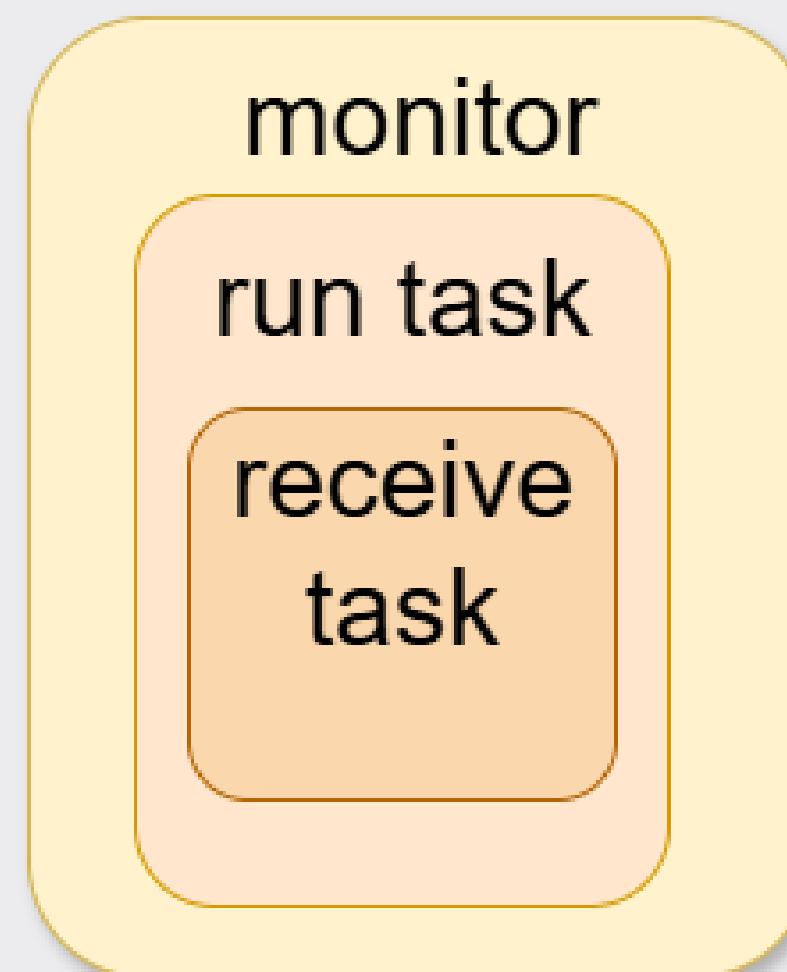
driver에 run task 안에  
send task를 만든다

send\_data를 interface를 통해  
DUT의 rx로 보내준다.



# UART\_FIFO 검증

monitor의 run\_task 안에 receive task를 만들어서 DUT를 지난 tx를 receive\_data로 재구성한다.

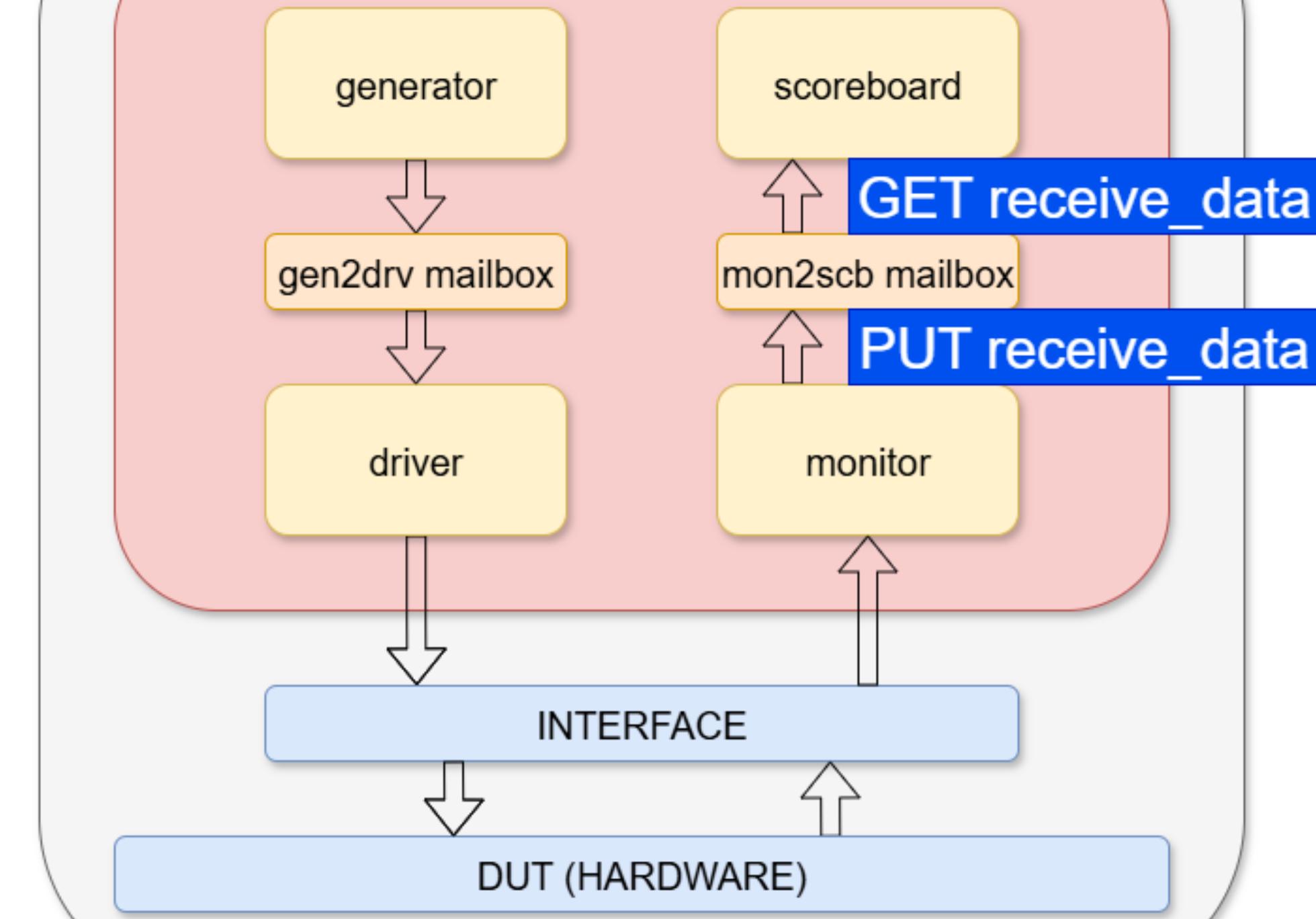


# UART\_FIFO 검증

monitor에서 만든 receive\_data를  
다시 mailbox를 통해서  
scoreboard로 보내준다.

## Testbench

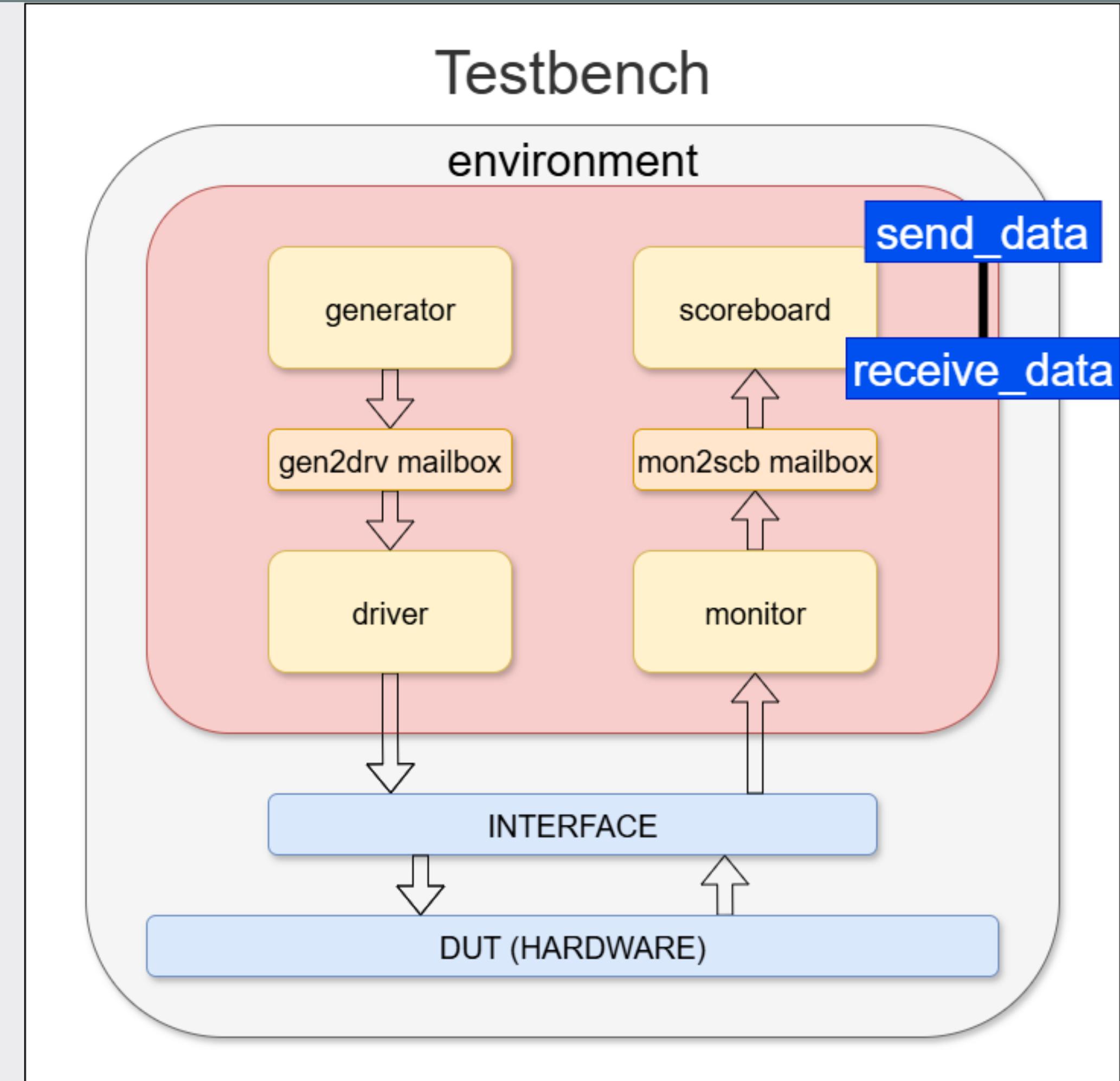
environment



# UART\_FIFO 검증

scoreboard에서 send\_data와 receive\_data를 비교해서 pass or fail count를 체크한다.

```
=====
===== test report =====
=====
===== Total test: 100
===== PASS test : 100
===== FAIL test : 0
=====
===== Testbench is finished =====
=====
```



# UART\_FIFO 검증\_트러블 슈팅\_1\_정보의 흐름

## 문제 발생

generator에서 생성한 **send\_data**를 비교해야 하는데  
monitor와 scoreboard에서 못 바라보는 문제 발생.

```
[MON] send_data = X, receive_data = 30
[SCB] send_data = X, receive_data = 30
```

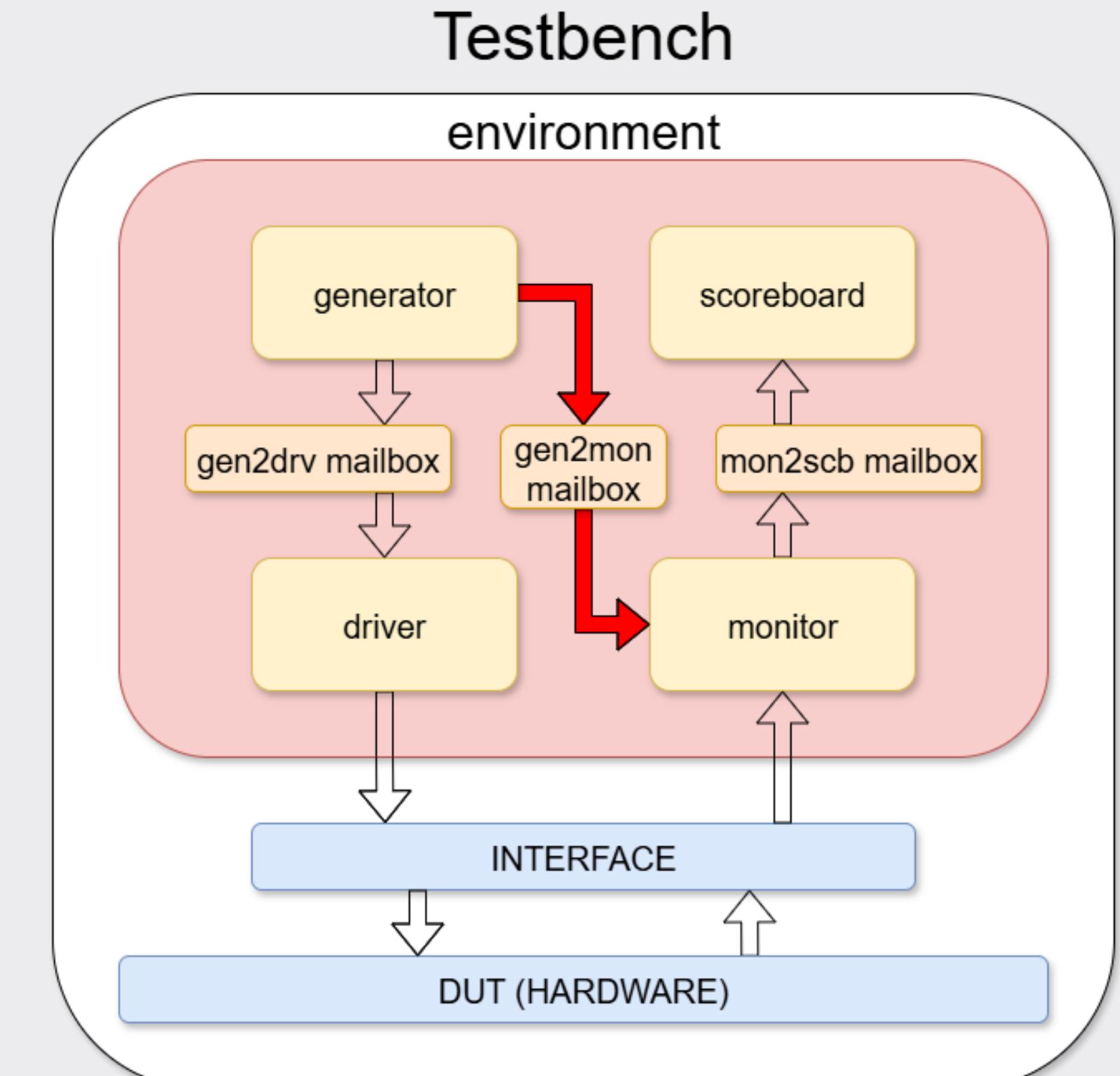
# UART\_FIFO 검증\_트러블 슈팅\_1\_정보의 흐름

## 해결 시도\_1

mailbox로 generator에서 monitor와 scoreboard로 직접 보내주는 방법.

gen2mon mailbox 선언해서 연결!

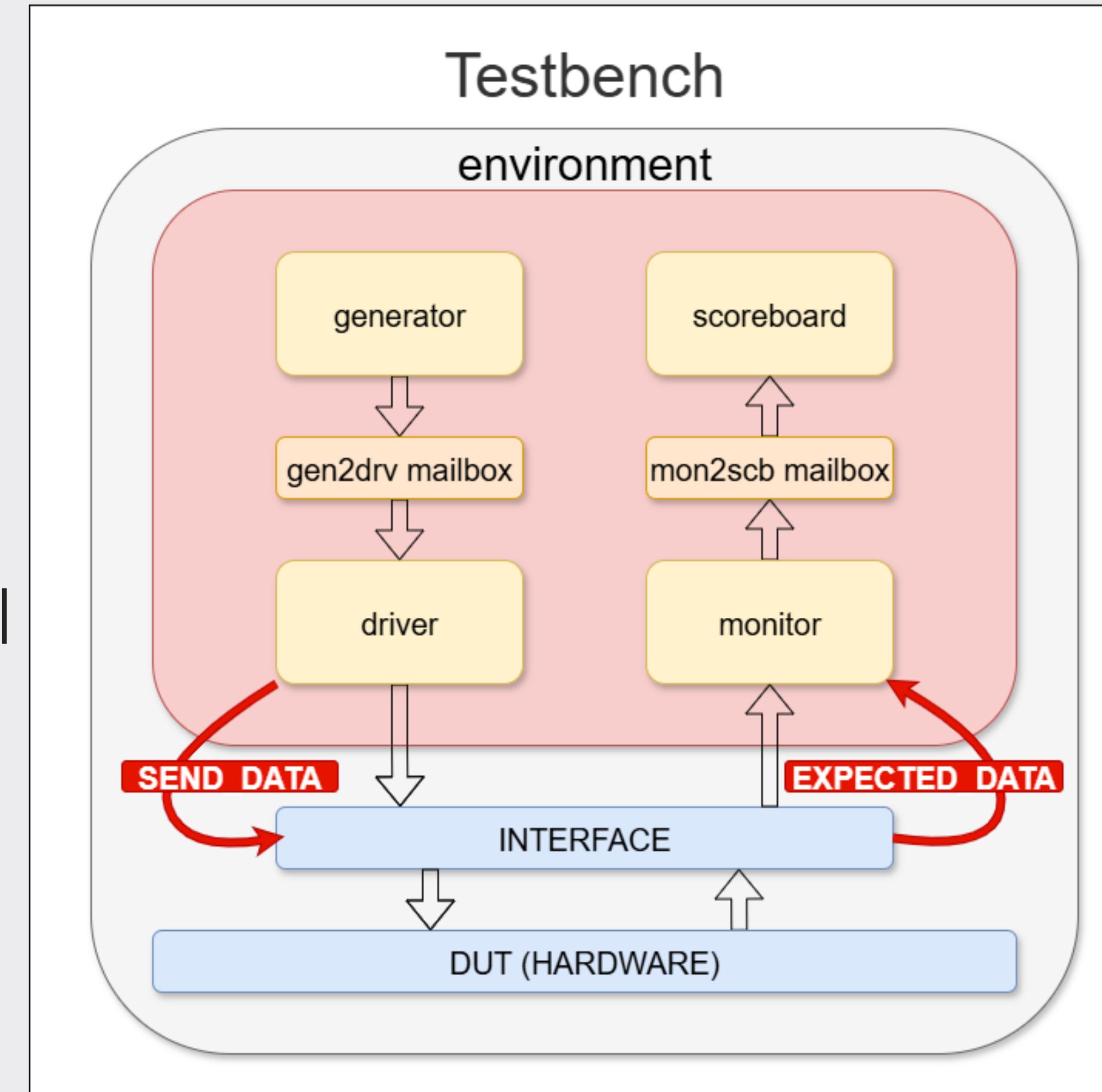
send\_data를 직접 보내준다.



# UART\_FIFO 검증

## 해결 시도\_2

generator의 send\_data를  
INTERFACE의 port로  
monitor 이후에 expected\_data까지  
포트를 통해서 옮겨주는 방법.



# UART\_FIFO 검증\_트러블 슈팅\_1\_정보의 흐름

## 해결 시도\_3

INTERFACE에 실제로 탑모듈의 input output이 아니더라도 전송용으로 port를 더 뚫어줄 수 있다.

```
3  interface uart_fifo_interface;
4      logic clk;
5      logic rst;
6      logic rx;
7      logic tx;
8      logic [7:0] expected_data; // send_data 전송용. //input 이자 output
9  endinterface //uart_fifo_interface
10 class transaction;
```

top 모듈의 실제 포트  
전송용 포트



# UART\_FIFO 검증\_트러블 슈팅\_1\_정보의 흐름

## 해결 성공

generator에서 생성한 send\_data와  
인터페이스를 통해 받은 expected\_data가  
일치하는 것을 확인했습니다.

```
202070445000 : [GEN] send_data = 128, receive_data = x
204153645000 : tx = 1, receive_bit_count = 8, receive_data = 10000000
204153645000 : [MON] send_data = x, receive_data = 128
204153645000 : [SCB] send_data = x, receive_data = 128
```

expected\_data 10000000

2진수 10\_000\_000 = 10진수 128

# UART\_FIFO 검증\_트러블 슈팅\_2\_동기화의 중복

**문제 발생** 일정한 타이밍인 3 bit의 expected\_data를 놓치고 이후의 tx부터 receive\_data로 쌓아가는 문제 발생.

Fail : Data Mismatched, Fail count :

expected\_data 00001010 **3 bit 놓침**

receive\_data 11100001

Fail : Data Mismatched, Fail count :

expected\_data 01111010 **3 bit 놓침**

receive\_data 11101111

1

2

# UART\_FIFO 검증\_트러블 슈팅\_2\_동기화의 중복

## 해결 시도\_1

BIT\_PERIOD를 일정하게 놓치는 문제이면 타이밍이면  
parameter의 사용 문제일 수 있지 않을까?  
아니면 중간값을 기다리려다가 절대적인 시간을  
너무 오래 지연한 것이 아닐까?

```
127 class monitor;
128     // monitor.receive를 위한 parameter 선언.
129     parameter CLOCK_PERIOD_NS = 10; //100MHz
130     parameter BITPERCLOCK = 10416; // 1bit per clock 1/9600
131     parameter BIT_PERIOD = BITPERCLOCK * CLOCK_PERIOD_NS; // number of clock * 10
```

# UART\_FIFO 검증\_트러블 슈팅\_2\_동기화의 중복

## 해결 실패\_1

3개씩 놓치다가 2개씩 놓치거나 하나도 안 놓치는 상황이 발생

```
| Fail : Data Mismatched, Fail count :  
| expected_data 00001010 ↗3 bit 놓침  
| receive_data 11100001  
| Fail : Data Mismatched, Fail count :  
| expected_data 01111010 ↗3 bit 놓침  
| receive_data 11101111  
| Fail : Data Mismatched, Fail count :  
| expected_data 00111001 ↗2 bit 놓침  
| receive_data 11001110
```

1 Pass, Data Matched, Pass  
2 expected\_data 11111111  
receive\_data 11111111  
3 11111111 ↓ 그대로 받음

# UART\_FIFO 검증\_트러블 슈팅\_2\_동기화의 중복

## 해결시도\_2\_에러의 time 값 추출

monitor에서 시간 지연 코드마다 시간을  
디스플레이 해보면서 에러를 분석

```
$display("%t : tx = %d, receive_bit_count = %d, receive_data = %b",
         $time, uart_fifo_if.tx, receive_bit_count,
         tr.receive_data);
```

```
1466785000 : tx = 1, receive_bit_count = 1, receive_data = 00000001
1570945000 : tx = 0, receive_bit_count = 1, receive_data = 00000001
```

run all

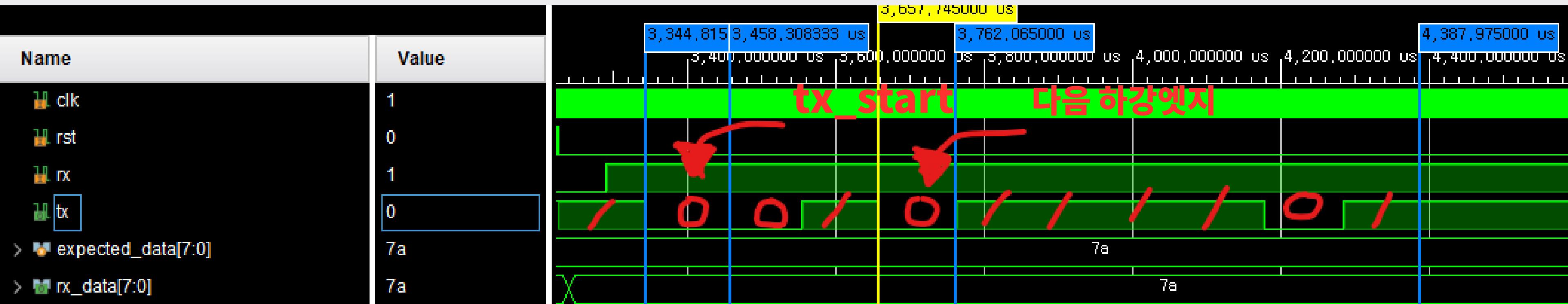
Time	tx	receive_bit_count	receive_data
1362625000	0	-	00000000
1466785000	1	-	00000000
1466785000	1	-	00000001
1466785000	1	-	00000001
1570945000	0	-	00000001
1570945000	0	-	00000001
1570945000	0	-	00000001
1675105000	0	-	00000001
1675105000	0	-	00000001
1675105000	0	-	00000001
1779265000	0	-	00000001
1779265000	0	-	00000001
1779265000	0	-	00000001
1779265000	0	-	00000001
1883425000	0	-	00000001
1883425000	0	-	00000001
1883425000	0	-	00000001
1987585000	1	-	00000001
1987585000	1	-	00100001
1987585000	1	-	00100001
2091745000	1	-	00100001
2091745000	1	-	01100001
2091745000	1	-	01100001
2195905000	1	-	01100001
2195905000	1	-	11100001
2195905000	1	-	11100001
2352145000	1	-	11100001
2352145000	[MON]	send_data = x, receive_data = 225	
2352145000	[SCB]	send_data = x, receive_data = 225	

Fail : Data Mismatched, Fail count : 1  
expected\_data 00001010  
receive\_data 11100001

# UART\_FIFO 검증\_트러블 슈팅\_2\_동기화의 중복

## 에러의 규칙성 발견

| Fail : Data Mismatched, Fail count : 2  
| expected\_data 01111010  
| receive\_data 11101111



tx의 첫 하강엣지를 start로 인식하는 것이 아니라 그 이후의 하강엣지를 start 신호로 인식해서 tx\_data의 0이 있어야 문제가 발생.

# UART\_FIFO 검증\_트러블 슈팅\_2\_동기화의 중복

## 해결 성공\_첫번째 동기화

```
class driver;
    task run();
        forever begin
            @(posedge uart_fifo_if.clk); // 클럭이 오면
            gen2drv mbox.get(tr); // transaction에서 데이터 가져온다.
            send(tr.send_data); // 데이터를 전송하고
            ->mon_next_event; // monitor에게 시작하라고 한다.
        end
    endtask //run
```

```
class monitor;
    task run();
        forever begin
            @mon_next_event;
            ...
        end
    endtask
```

```
task send(input [7:0] send_data);
begin
    // start bit
    uart_fifo_if.rx = 0;
    uart_fifo_if.expected_data = tr.send_data;
    #(BIT_PERIOD);
    for (
        send_bit_count = 0;
        send_bit_count < 8;
        send_bit_count = send_bit_count + 1
    ) begin
        if (tr.send_data[send_bit_count]) begin
            uart_fifo_if.rx = 1'b1;
        end else begin
            uart_fifo_if.rx = 1'b0;
        end
        #(BIT_PERIOD);
    end
    //stop bit
    uart_fifo_if.rx = 1'b1;
    #(BIT_PERIOD);
end
endtask
```

# UART\_FIFO 검증\_트러블 슈팅\_2\_동기화의 중복

## 해결 성공\_두번째 동기화

rx\_done이 되면 wire를 통해서 tx\_start가 됩니다. 따라서 이후에 receive task가 시작되면 이 tx\_start 때 발생하는 tx의 하강엣지 이후의 다음 하강엣지를 기다리는 시간지연이 발생합니다.

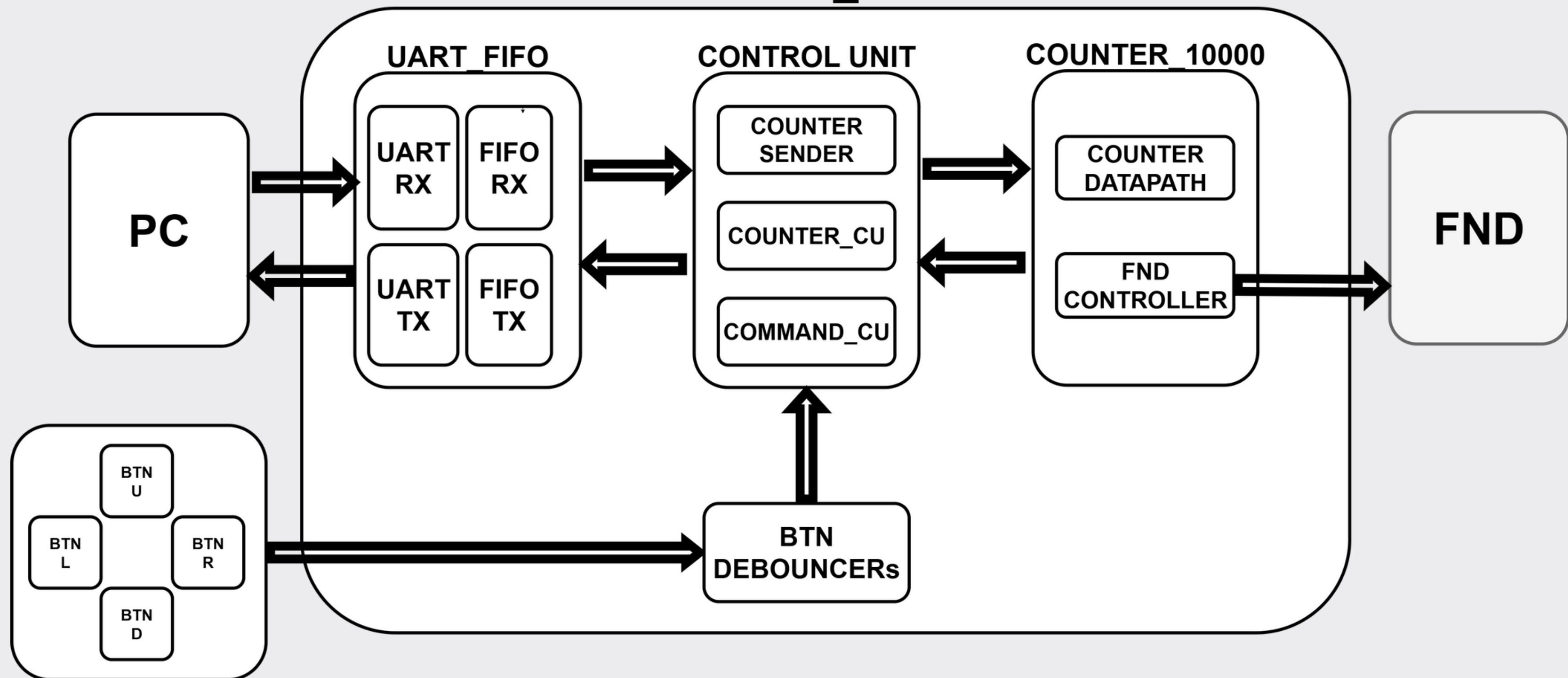
```
class monitor;
    task receive();
        begin
            tr.receive_data = 0;
            @(negedge uart_fifo_if.tx)
```

# UART\_FIFO 검증 완료

```
| Pass : Data Matched, Pass count : 100
| expected_data 00001110
| receive_data 00001110
| finished
| =====
| ===== test report =====
| =====
| ===== Total test: 100
| ===== PASS test : 100
| ===== FAIL test : 0
| =====
| ===== Testbench is finished =====
| =====
```

# COUNTER\_UART Block\_diagram

FINAL\_TOP



# COUNTER\_UART 요구 기능

PC에서 ascii code 입력시 uart로 COUNTER와 데이터 송수신!!

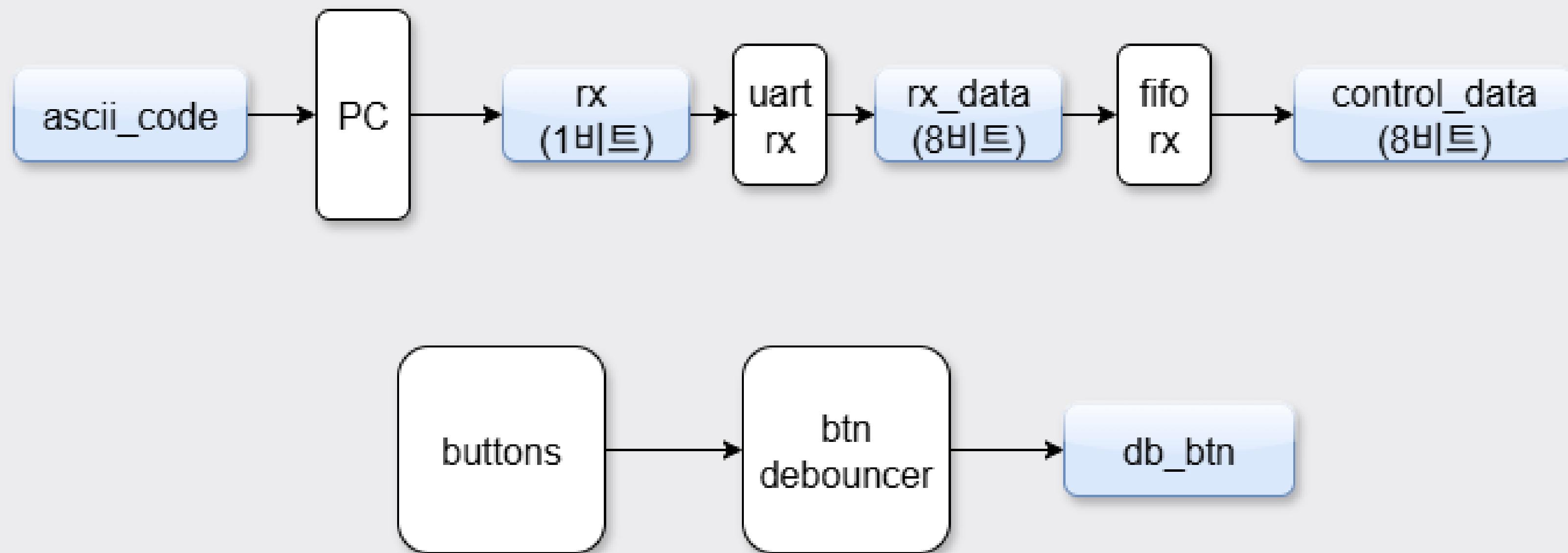
R (8'h52) : enable 변경 ( enable ==1 : run, enable == 0 : stop )

L (8'h4C) : clear (enable == 0) 일 때만 가능

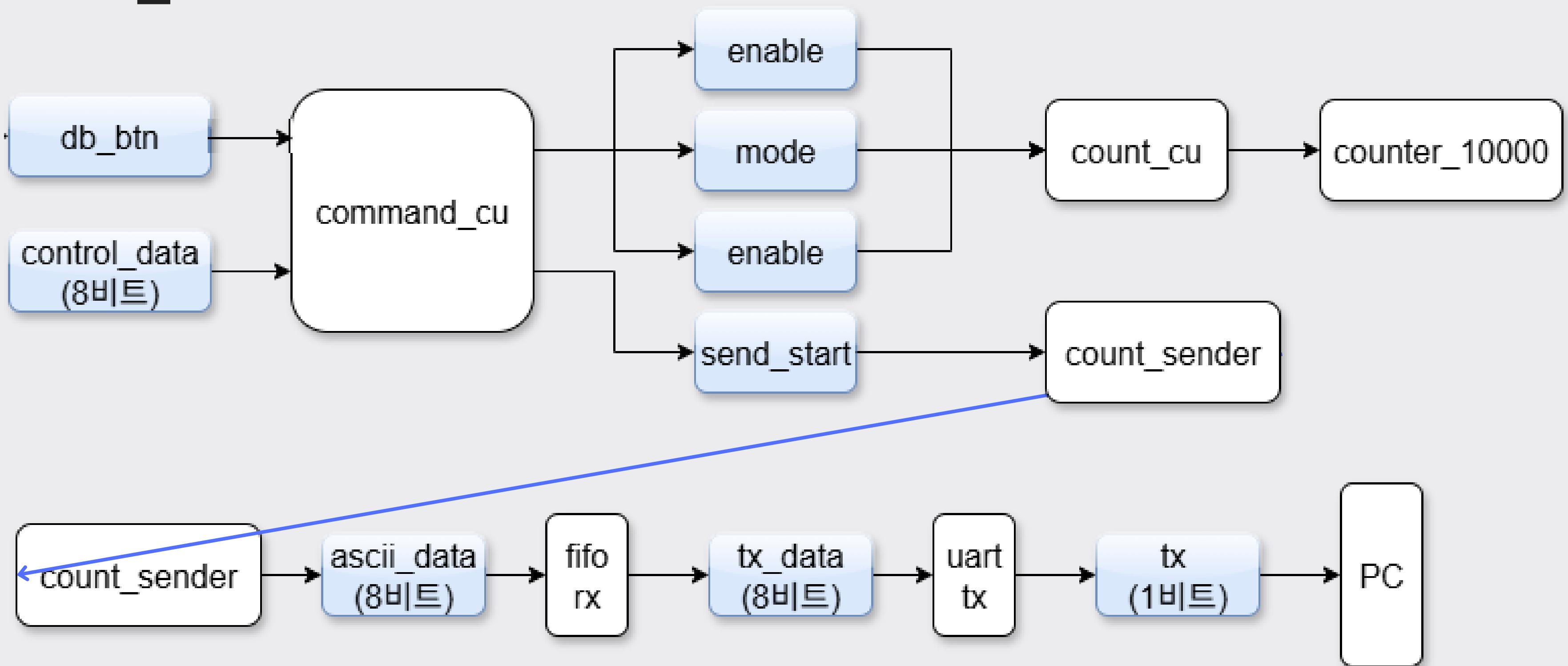
U (8'h55) : mode 변경 ( mode == 1 : UP, mode == 0 : DOWN)

D (8'h44) : PC로 COUNT 값을 전송!

# flow\_chart



# flow\_chart



# Testbench

```
// 물리 버튼 시뮬레이션
btnR = 1;
#10;
btnR = 0;
 #(100 * 1000000); // 0.1sec
btnL = 1;
#100;
btnL = 0;
#100;
#(300 * 1000000); // 0.3sec
btnU = 1;
#(100 * 1000000); // 0.1sec
btnR = 1;
#100 btnR = 0;
#100 #(100 * 1000000); // 0.1sec
```

```
// PC 통신 시뮬레이션
// test task
send_data = 8'h52; // R
single_uart_rx_test(send_data);
#100;
send_data = 8'h52; // R
single_uart_rx_test(send_data);
#100;
#(100 * 1000000); // 0.1sec
send_data = 8'h55; // U
single_uart_rx_test(send_data);
#(100 * 1000000); // 0.1sec
send_data = 8'h44; // D
single_uart_rx_test(send_data);
#(400 * 1000000); // 0.4sec
single_uart_rx_test(send_data);
#(200 * 1000000); // 0.2sec
```

# Testbench

```
// 물리 버튼 시뮬레이션  
btnR = 1;  
#50;  
btnR = 0;  
#(300 * 1000000); // 0.3sec  
btnU = 1;  
#50;  
btnU = 0;  
#(500 * 1000000); // 0.5sec  
btnR = 1;  
#50;  
btnR = 0;  
#(100 * 1000000); // 0.1sec  
btnL = 1;  
#50;  
btnL = 0;  
#(300 * 1000000); // 0.1sec
```

**ENABLE 0 → 1**

**MODE 0 → 1**

**ENABLE 1 → 0**

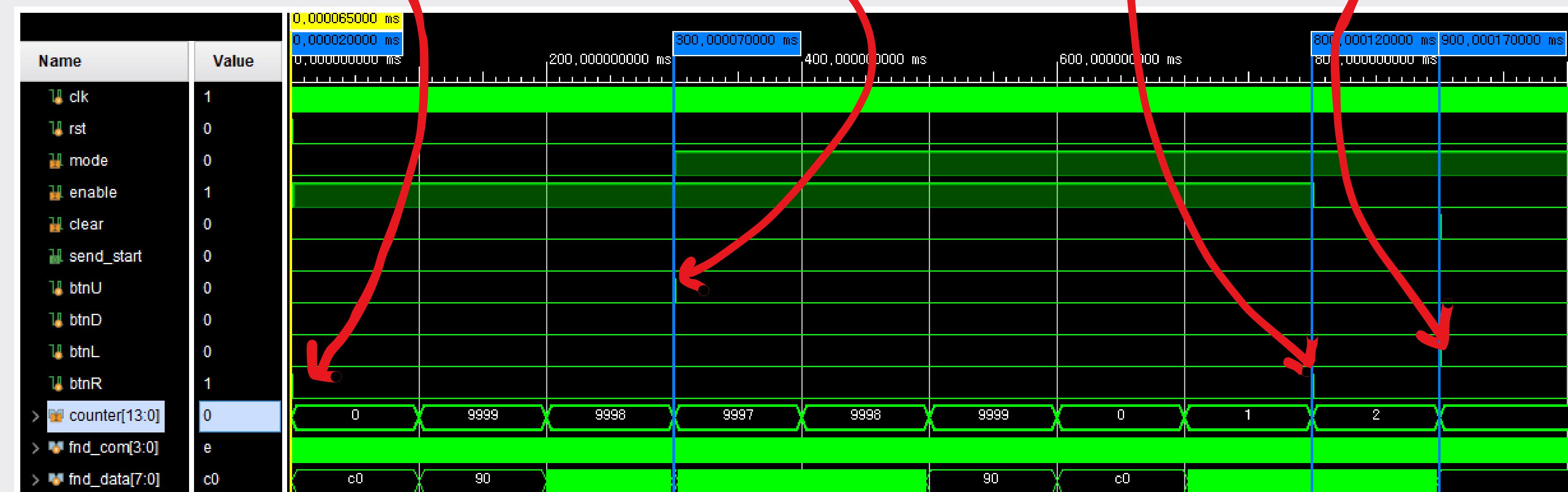
**CLEAR 0 → 1**

# Testbench

R : enable

U : mode

R : enable L : clear



# Testbench

**ENABLE 0 → 1**

**MODE 0 → 1**

**ENABLE 1 → 0**

**send\_start 0 → 1**

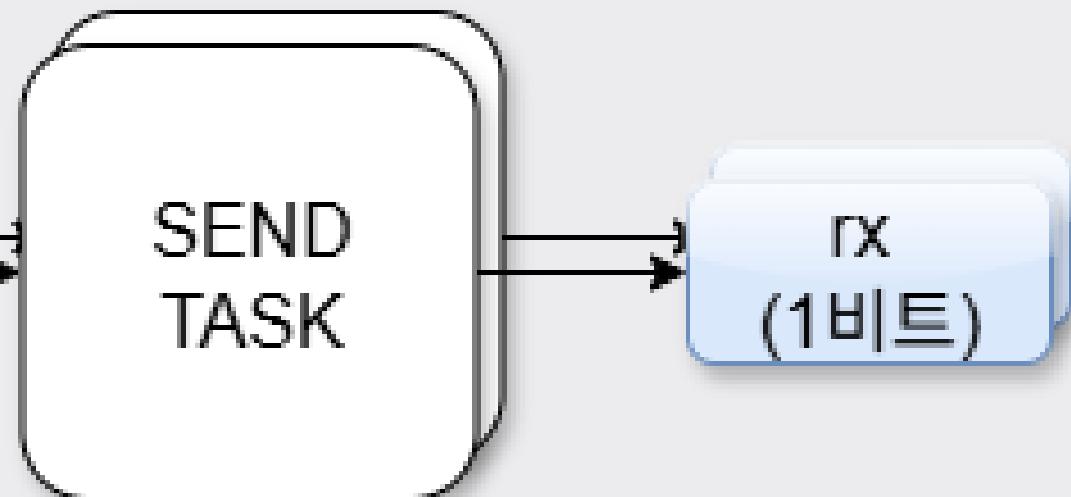
**CLEAR 0 → 1**

**send\_start 0 → 1**

```
// PC 통신 시뮬레이션  
// test task  
send_data = 8'h52; // R  
single_uart_rx_test(send_data);  
#(300 * 1000000); // 0.3sec  
send_data = 8'h55; // U  
single_uart_rx_test(send_data);  
#(500 * 1000000); // 0.5sec  
send_data = 8'h52; // R  
single_uart_rx_test(send_data);  
#(100 * 1000000); // 0.1sec  
send_data = 8'h44; // D  
single_uart_rx_test(send_data);  
#(100 * 1000000); // 0.1sec  
send_data = 8'h4C; //L 8'h4C  
single_uart_rx_test(send_data);  
#(200 * 1000000); // 0.2sec  
send_data = 8'h44; // D  
single_uart_rx_test(send_data);  
#(100 * 1000000); // 0.1sec
```

# Testbench

```
task send(input [7:0] send_data);
begin
    // start bit
    rx = 0;
    #(BIT_PERIOD);
    for (bit_count = 0; bit_count < 8; bit_count = bit_count + 1) begin
        if (send_data[bit_count]) begin
            rx = 1'b1;
        end else begin
            rx = 1'b0;
        end
        #(BIT_PERIOD);
    end
    //stop bit
    rx = 1'b1;
    #(BIT_PERIOD);
end
endtask
```



```
task receive_uart();
begin
    receive_data = 0;
    @(negedge tx);
    // middle of start bit
    #52080; //BIT_PERIOD / 2

    // start bit pass / fail
    if (tx) begin // fail일 경우
        $display("Fail Start bit");
    end

    // received data bit
    for (bit_count = 0; bit_count < 8; bit_count = bit_count + 1) begin
        #(BIT_PERIOD);
        receive_data[bit_count] = tx;
    end

    // check stop bit
    #(BIT_PERIOD);
    if (!tx) begin // fail일 경우
        $display("Fail STOP bit");
    end
    #52080; //BIT_PERIOD / 2
end
endtask
```

# Testbench



```
task receive_uart();
begin
    receive_data = 0;
    @(negedge tx);
    // middle of start bit
    #52080; //BIT_PERIOD / 2

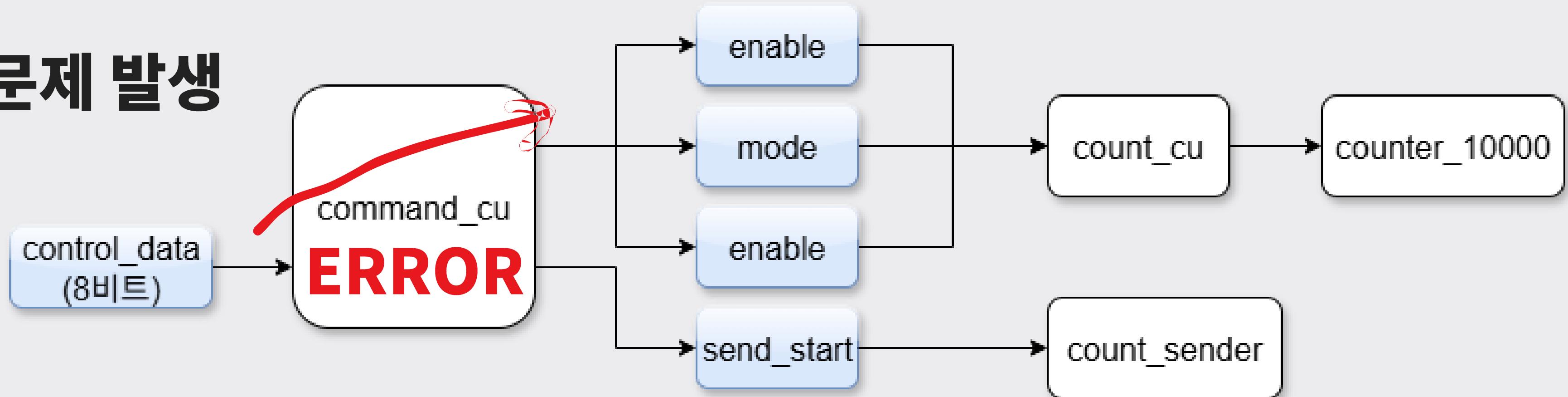
    // start bit pass / fail
    if (tx) begin // fail일 경우
        $display("Fail Start bit");
    end

    // received data bit
    for (bit_count = 0; bit_count < 8; bit_count = bit_count + 1) begin
        #(BIT_PERIOD);
        receive_data[bit_count] = tx;
    end

    // check stop bit
    #(BIT_PERIOD);
    if (!tx) begin // fail일 경우
        $display("Fail STOP bit");
    end
    #52080; //BIT_PERIOD / 2
end
endtask
```

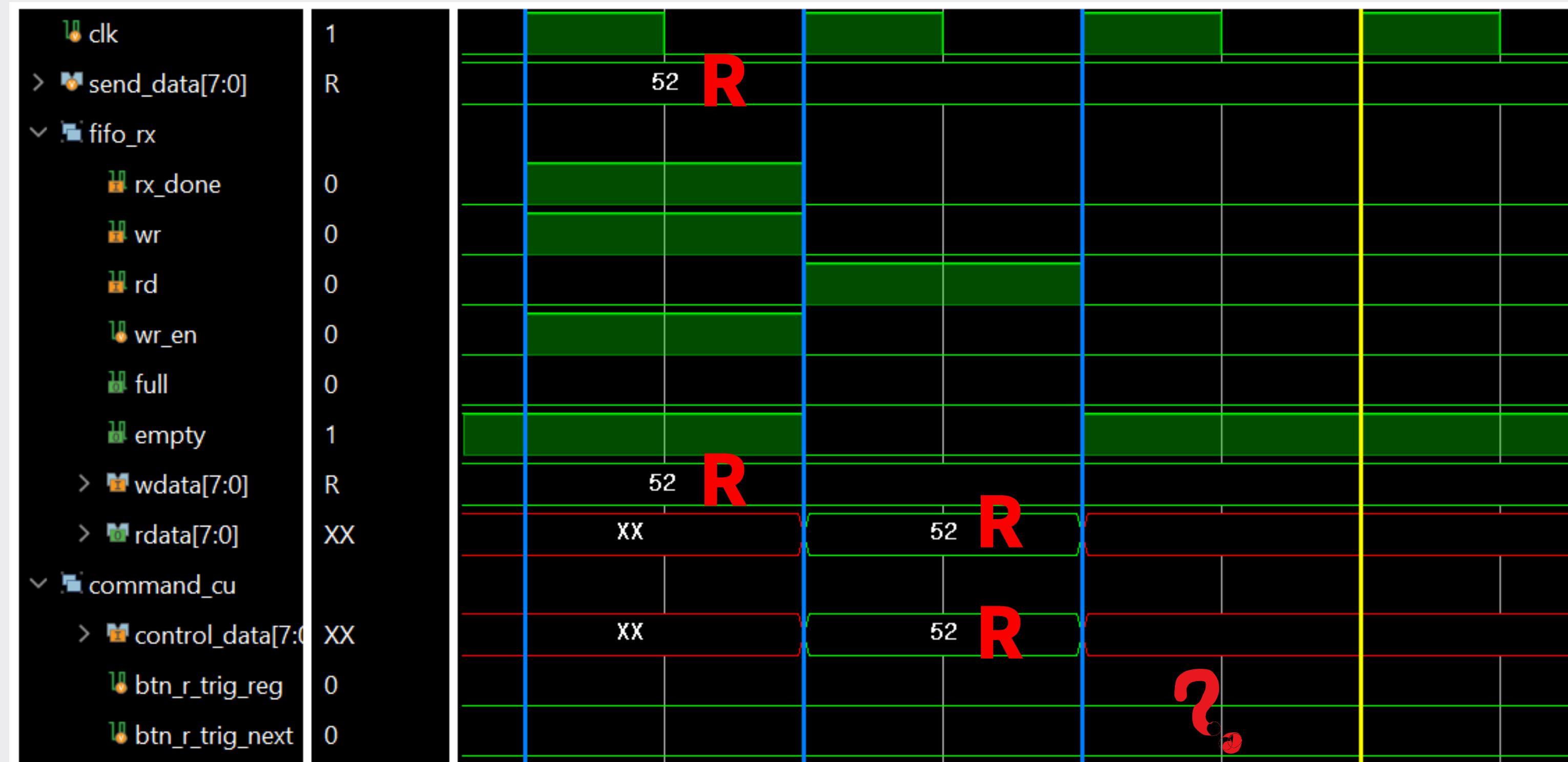
# Counter\_uart\_트러블슈팅

문제 발생



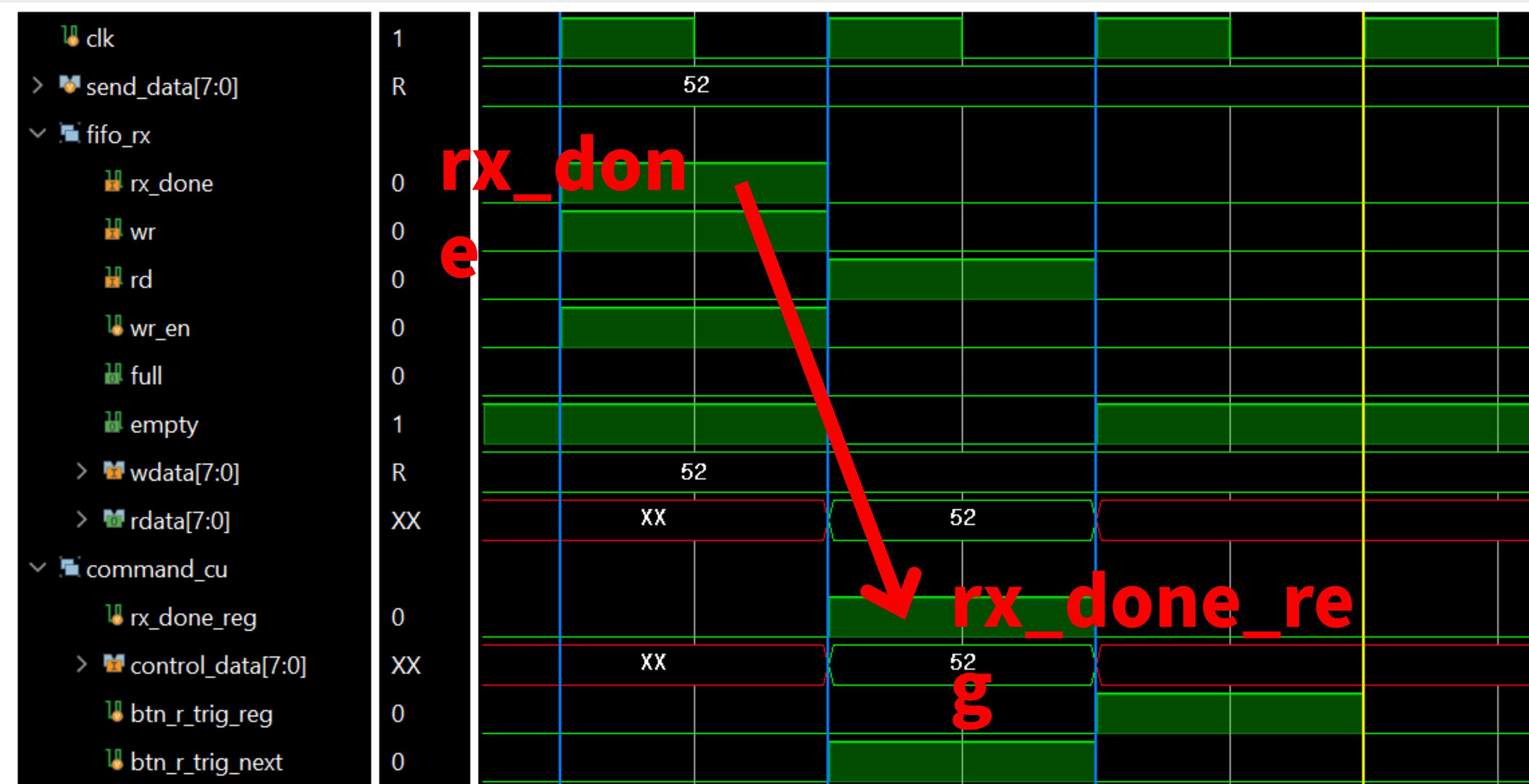
# Counter\_uart Block\_트러블슈팅

## 문제 발생

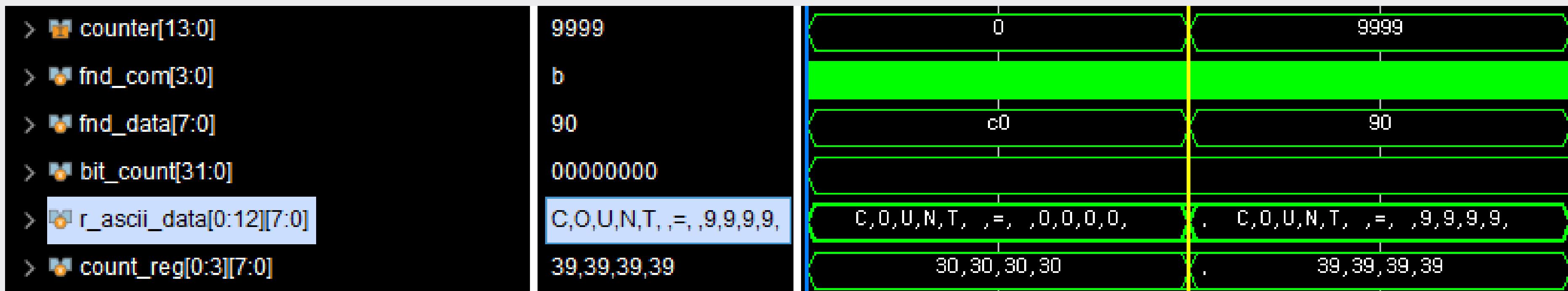


# Counter\_uart Block\_트러블슈팅

해결 성공



# Counter\_ascii 시뮬레이션





# 프로젝트 팀 구성

이름

유지훈

신상혁

역할

uart\_tx, uart\_rx 검증  
counter\_10000 RTL

fifo, uart\_fifo 검증  
counter\_10000 RTL 테스트벤치

# 마침 인사

프로젝트에 대한 프레젠테이션을 마치겠습니다.

프로젝트와 관련한 질문을 받겠습니다.



