2025.09.16

13조

System Verilog : 10000 COUNTER with Uart

황석현 | 이윤지

CONTENTS

01 Introduction

02 10000진 Counter

03 RX

04 TX

05 FIFO

06 Uart TOP

07 고찰

08 Trouble shooting

CHAPTER 01

Introduction

프로젝트 목표

- 10000진 카운터 시스템의 UART 경로의 신뢰성을 위해 UART의 RX·TX·FIFO를 각각/통합으로 검증하고자 함

System Verilog 선택 이유

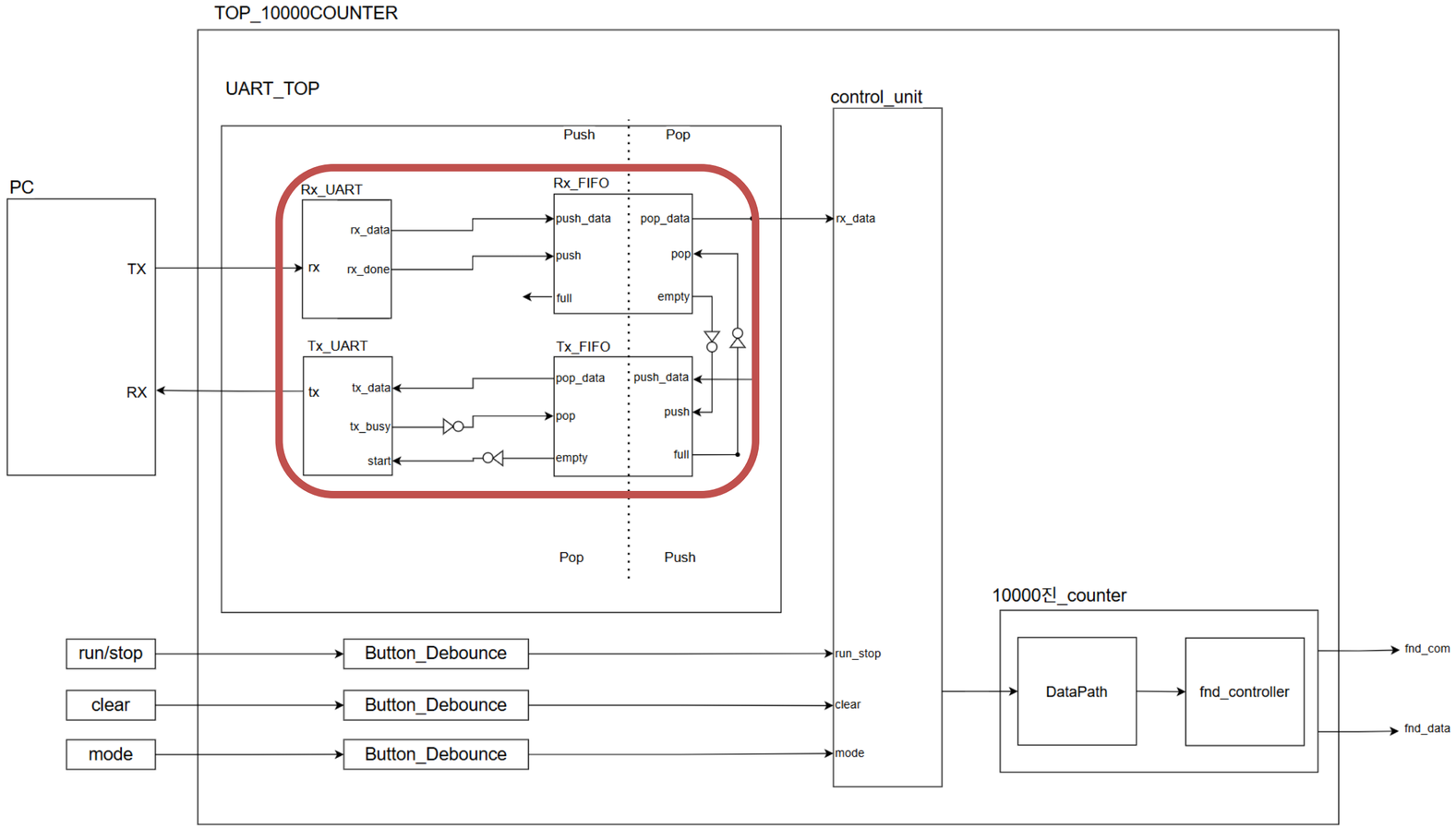
- 인터페이스로 DUT 연결 관리에 용이
- Report 체계로 테스트 흐름/원인 추적 가능

설계 환경

- Xilinx Vivado
- System Verilg
- ComPortMaster

CHAPTER 02

10000진 Counter

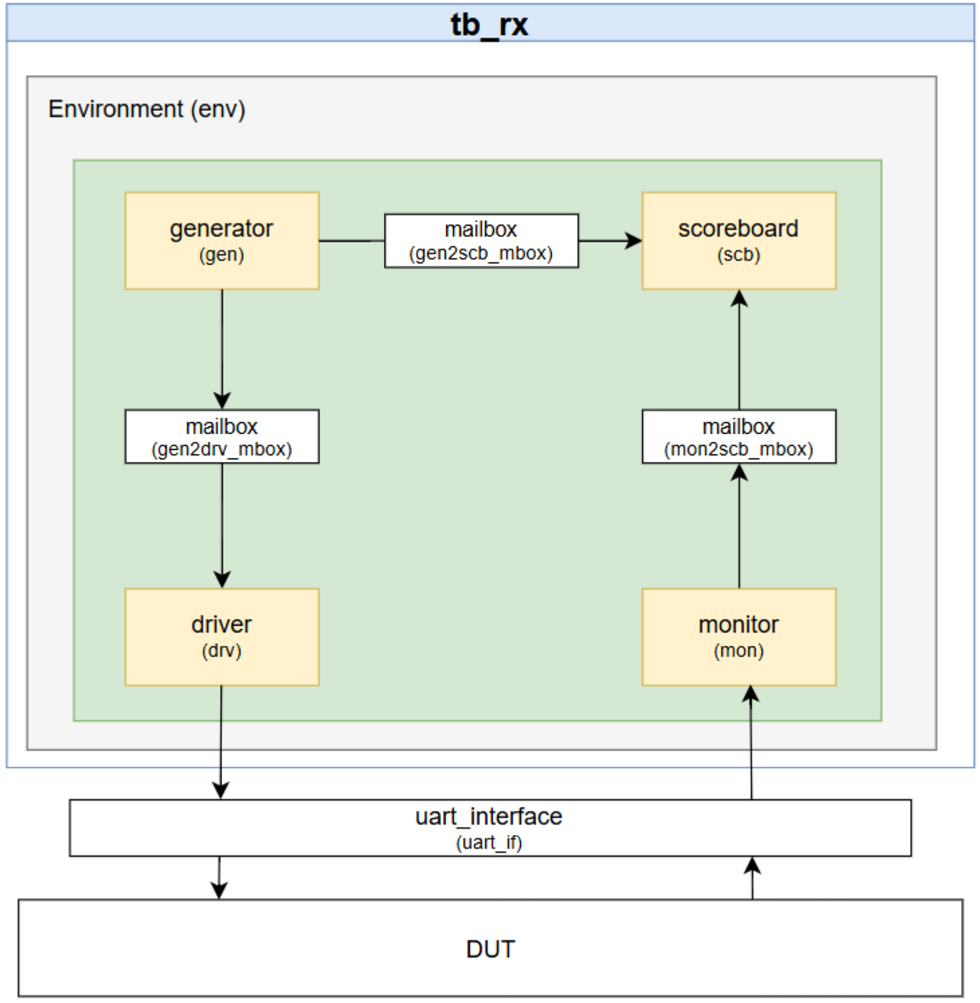


10000진 counter

- 역할 : FPGA 보드의 **4자리 10000진 카운터(0~9999)** 시스템
- 구현된 시스템을 FPGA 보드와 UART에서 자유자재로 동작하고자 함
- PC와 카운터 간 통신을 위한 UART & FIFO 모듈을 검증

CHAPTER 03

RX



SV Testbench 구조

- Generator : DUT에 주입할 랜덤 데이터 생성
- Driver : Generator로부터 받은 데이터를 DUT의 입력 핀에 맞춰 구동(Drive)
- Monitor : DUT의 출력 핀을 감시, 출력되는 데이터를 캡처
- Scoreboard : Generator로부터 받은 예상 데이터, Monitor로부터 받은 실제 데이터 비교
- Mailbox: 각 class간 메시지를 주고받는 역할

CHAPTER 03

RX

```
interface uart_rx_interface;
    logic       clk;
    logic       rst;
    logic       rx;
    logic       b_tick;
    logic [7:0] rx_data;
    logic       rx_done;
endinterface
```

```
module tb_uart_rx ();

    uart_rx_interface uart_rx_tb ();
    environment env;
    uart_rx dut (
        .clk(uart_rx_tb.clk),
        .rst(uart_rx_tb.rst),
        .rx(uart_rx_tb.rx),
        .b_tick(uart_rx_tb.b_tick),
        .rx_data(uart_rx_tb.rx_data),
        .rx_done(uart_rx_tb.rx_done)
    );

    baud_tick_gen dut_tick (
        .clk(uart_rx_tb.clk),
        .rst(uart_rx_tb.rst),
        .b_tick(uart_rx_tb.b_tick)
    );

    always #5 uart_rx_tb.clk = ~uart_rx_tb.clk;
    initial begin
        uart_rx_tb.clk = 0;
        env = new(uart_rx_tb);
        env.run();
    end

endmodule
```

Interface & DUT

- Interface : DUT와 Environment를 연결하는 역할
- DUT : 검증이 되는 실제 하드웨어 설계
- 100MHz 기반 dut_tick의 출력 b_tick과 uart_rx dut의 입력 b_tick을 연결
  > 실제 동작과 동일한 타이밍 구성
- Dut_tick에서 출력된 b_tick은 dut와 driver, moniter에 공통으로 사용

CHAPTER 03

RX

```
class transaction;

    randc bit [7:0] send_data;
    bit rx;
    bit b_tick;
    bit [7:0] rx_data;
    bit rx_done;

    task display(string name_s);
        $display("%t : [%s] : send_data= %d ", $time, name_s, send_data);
    endtask
endclass
```

Random-cyclic

```
task run();
    drv.reset();
    fork
        gen.run(256);
        drv.run();
        mon.run();
        scb.run();
    join_any
```

```
class generator;
    transaction tr;
    mailbox #(transaction) gen2drv_mbox;
    mailbox #(transaction) gen2scb_mbox;
    event gen_next_event;

    int total_count = 0;

    function new(mailbox#(transaction) gen2drv_mbox,
                 mailbox#(transaction) gen2scb_mbox, event gen_next_event);
        this.gen2drv_mbox   = gen2drv_mbox;
        this.gen2scb_mbox   = gen2scb_mbox;
        this.gen_next_event = gen_next_event;
    endfunction  //new()

    task run(int count);
        tr = new();
        repeat (count) begin
            total_count++;
            assert (tr.randomize())
            else $display("[GEN] tr.randomize () error!!!!!");
            tr.display("GEN");
            gen2drv_mbox.put(tr);
            gen2scb_mbox.put(tr);  //expected data
            #1;
            @(gen_next_event);  //scb끝날때까지 기다림, 받음
        end
    endtask
endclass
```

Repeat 바깥에 선언하여 repeat 할 때마다 값이 중복 생성되지 않도록 함

Transaction & environment & Generator

- Transaction : random_cyclic을 선언 > 모든 값이 중복없이 생성됨.
- [7:0] send_data 경우의 수 : 256개 > environment 의 gen.run(256) 으로 선언
- Generator : tr = new()를 repeat 바깥에 > 매 반복마다 new()를 하면 값 중복 생성
- Transaction : display를 통해 각 class가 처리될 때 "시간 : class 명 : send_data"를 출력

CHAPTER 03

RX

```
class environment;

    mailbox #(transaction) gen2drv_mbox;
    mailbox #(transaction) mon2scb_mbox;
    mailbox #(transaction) gen2scb_mbox;
    generator gen;
    driver drv;
    monitor mon;
    scoreboard scb;

    event gen_next_event;
    event mon_next_event;

    function new(virtual uart_rx_interface uart_rx_tb);
        gen2drv_mbox = new();
        mon2scb_mbox = new();
        gen2scb_mbox = new();
        gen = new(gen2drv_mbox, gen2scb_mbox, gen_next_event);
        drv = new(gen2drv_mbox, uart_rx_tb, mon_next_event);
        mon = new(mon2scb_mbox, uart_rx_tb, mon_next_event);
        scb = new(mon2scb_mbox, gen2scb_mbox, gen_next_event);
    endfunction
```

```
task report();
    $display("============================");
    $display("======= test report =======");
    $display("============================");
    $display("==     Total Test : %4d    ==", gen.total_count);
    $display("==     Pass Test : %4d     ==", scb.Pass_count);
    $display("==     Fail Test : %4d     ==", scb.Fail_count);
    $display("============================");
    $display("==== Test bench is finish ===");
    $display("============================");
endtask


task run();
    drv.reset();
    fork
        gen.run(256);
        drv.run();
        mon.run();
        scb.run();
    join_any
    #10;
    report();
    $display("finished");
    $stop;
endtask
```

environment

- Environment : new() 를 통해 각 class 생성
- 각 class 간 통신에 필요한 mailbox를 생성
- 각 class간 동기화 신호를 위한 event 생성
- Report : task run() 을 통해 각 class의 동작이 모두 끝난 후 test의 결과값 출력
- fork-join_any : 각 class를 동시 실행, 하나라도 task 종료 시 fork문 종료

CHAPTER 03

RX

```
class generator;
    transaction tr;
    mailbox #(transaction) gen2drv_mbox;
    mailbox #(transaction) gen2scb_mbox;
    event gen_next_event;

    int total_count = 0;

    function new(mailbox#(transaction) gen2drv_mbox,
                 mailbox#(transaction) gen2scb_mbox, event gen_next_event);
        this.gen2drv_mbox   = gen2drv_mbox;
        this.gen2scb_mbox   = gen2scb_mbox;
        this.gen_next_event = gen_next_event;
    endfunction  //new()

    task run(int count);
        tr = new();
        repeat (count) begin
            total_count++;
            assert (tr.randomize())
            else $display("[GEN] tr.randomize () error!!!!!");
            tr.display("GEN");
            gen2drv_mbox.put(tr);
            gen2scb_mbox.put(tr);  //expected data
            #1;
            @(gen_next_event);  //scb끝날때까지 기다림, 받음
        end
    endtask
endclass
```

Generator

- Generator : DUT에 주입할 transaction 객체 생성
- transaction의 randc : 0 – 255까지 모든 8bit 데이터를 중복없이 생성
- Mailbox : 데이터를 dut로 입력할 driver, dut의 출력과 비교할 scoreboard로 동시 전송
- Event : scoreboard로 부터 신호를 받을 때까지 대기 > 데이터 밀림 방지

CHAPTER 03

RX

```
class driver;
    transaction tr;
    mailbox #(transaction) gen2drv_mbox;
    virtual uart_rx_interface uart_rx_tb;
    event mon_next_event;

    int t = 0;
    int i = 0;

    function new(mailbox#(transaction) gen2drv_mbox,
                 virtual uart_rx_interface uart_rx_tb, event mon_next_event);
        this.gen2drv_mbox = gen2drv_mbox;
        this.uart_rx_tb = uart_rx_tb;
        this.mon_next_event = mon_next_event;
    endfunction  //new()

    task reset();
        //uart_rx_tb.clk = 0;
        uart_rx_tb.rst = 1;
        uart_rx_tb.rx  = 1;
        repeat (2) @(posedge uart_rx_tb.clk);
        uart_rx_tb.rst = 0;
        repeat (2) @(posedge uart_rx_tb.clk);
        $display("[DRV] reset done");
    endtask
```

B_tick을 repeat하여 dut의 비트 타이밍과 일치

```
task run();
    forever begin
        // tr = new;
        gen2drv_mbox.get(tr);  //send_data
        tr.display("DRV");
        uart_rx_tb.rx = 0;
        // --->> b_tick_cnt_next = 0;

        //start
        repeat (16) @(posedge uart_rx_tb.b_tick);

        //data
        for (i = 0; i < 8; i++) begin
            uart_rx_tb.rx = tr.send_data[i];
            repeat (16) @(posedge uart_rx_tb.b_tick);
        end

        uart_rx_tb.rx = 1;
        @(posedge uart_rx_tb.rx_done);
        ->mon_next_event;
        repeat (16) @(posedge uart_rx_tb.b_tick);
    end
endtask
```

Driver

- Driver : mailbox를 통해 generator 에서 받은 데이터를 dut 에 맞춰 rx 입력 핀 구동
- 각 비트의 타이밍은 b_tick 신호에 동기화 : dut의 각 비트 타이밍과 일치
- 스타트비트(16틱) → 데이터 8비트(비트당 16틱) → 스탑비트(16틱)를 생성
- 수신 완료(rx_done) 대기 후 mon_next_event로 monitor에 신호 보냄

CHAPTER 03

RX

```
//dut의 출력
class monitor;
    transaction real_tr;
    mailbox #(transaction) mon2scb_mbox;
    virtual uart_rx_interface uart_rx_tb;
    event mon_next_event;

    function new(mailbox#(transaction) mon2scb_mbox,
                 virtual uart_rx_interface uart_rx_tb, event mon_next_event);
        this.mon2scb_mbox = mon2scb_mbox;
        this.uart_rx_tb = uart_rx_tb;
        this.mon_next_event = mon_next_event;
    endfunction  //new()

    task run();
        forever begin
            @(mon_next_event);  //이벤트 받음
            //@(posedge uart_rx_tb.rx_done);
            real_tr = new();  // 새로운 tr 객체 생성
            real_tr.send_data = uart_rx_tb.rx_data;
            real_tr.rx      = uart_rx_tb.rx;   //각 count마다 dut에서 관찰되는 값을 복사
            real_tr.b_tick = uart_rx_tb.b_tick;
            real_tr.rx_data = uart_rx_tb.rx_data;
            real_tr.display("MON");

            //@(posedge uart_rx_tb.clk);
            mon2scb_mbox.put(real_tr);
        end
    endtask
endclass
```

Monitor

- Monitor : event를 통해 driver에서 신호를 받으면 Dut 출력 캡처
- Dut의 출력을 real_tr이라는 객체를 생성해 받음
- mailbox를 통해 Scoreboard로 전달
- Real_tr : generator 에서 받은 데이터와 monitor로 관찰한 데이터 구분

CHAPTER 03

RX

```
task run();
    forever begin

        gen2scb_mbox.get(tr);
        mon2scb_mbox.get(real_tr);
        tr.display("SCB");
        real_data = real_tr.send_data;

        expected_data = tr.send_data;

        if (expected_data == real_data) begin
            $display("[SCB] : Data matched : %d", real_data);
            $display(
                "expected_data = %d, real_data = %d",
                expected_data, real_data);
            Pass_count++;
        end else begin
            $display("[SCB] : Data mis-matched : %d, %d", real_data,
                     expected_data);
            $display(
                "expected_data = %d, real_data = %d",
                expected_data, real_data);
            Fail_count++;
        end

        ->gen_next_event;
    end
endtask
```

```
[SCB] : Data matched :  91
expected_data =  91, real_data =  91
```

env.run에서 scb.run까지 동작되면 출력되는 report

```
==============================
======== test report ========
==============================
==     Total Test :  256     ==
==     Pass Test :  256      ==
==     Fail Test :    0      ==
==============================
==== Test bench is finish ===
==============================
finished
```

Scoreboard

- Scoreboard : generator에서 send_data를 받아 expected_data에 저장
- Monitor에서 dut 결과를 real_data로 저장
- 256개의 경우마다 data_matched 혹은 dis-matched 출력 >
  각 케이스의 비교 메시지 출력
- 한 번의 비교가 끝나면 event로 다음 generator에 신호 보냄
- 256개 각각의 경우마다 pass / fail 집계
  > 마지막에 report가 집계 결과 출력

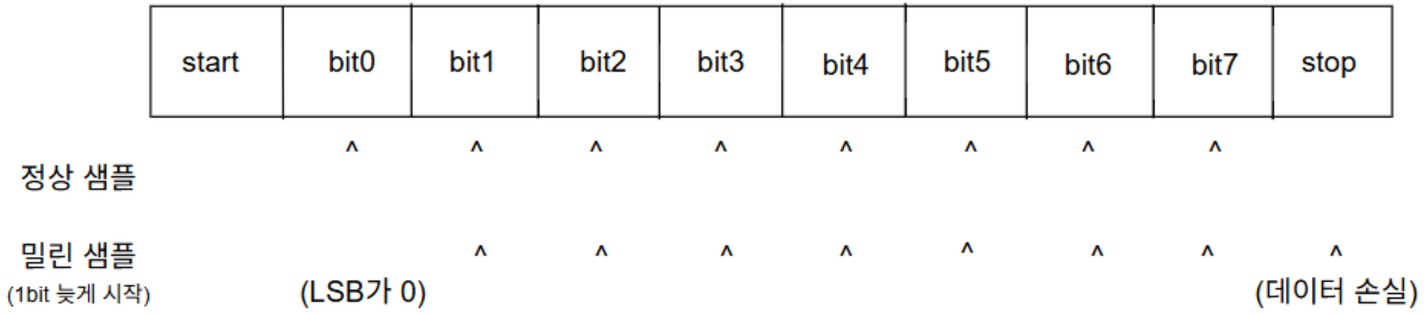
CHAPTER 03

RX

```
//data
for (i = 1; i < 9; i++) begin
    uart_rx_tb.rx = tr.send_data[i];
    repeat (16) @(posedge uart_rx_tb.b_tick);
end
```

```
==============================
======== test report ========
==============================
==     Total Test :  256     ==
==     Pass Test :    1      ==
==     Fail Test :  255      ==
==============================
==== Test bench is finish ===
==============================
```

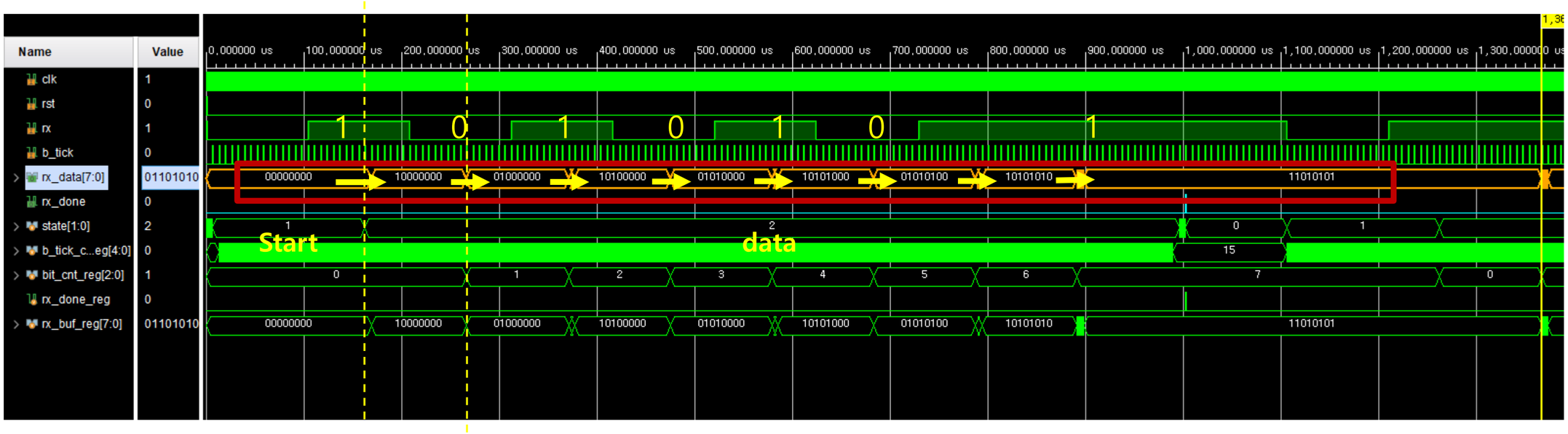
```
        38409015000 : [GEN] : send_data=   0
        38513165000 : [DRV] : send_data=   0
        39509205000 : [MON] : send_data=   0
        39509205000 : [SCB] : send_data=   0
[SCB] : Data matched :   0
expected_data =   0, real_data =   0
```



- **Why?** Expected_data 와 real_data (비교 데이터) 가 맞게 비교되는지
- 1bit씩 미뤄서 데이터를 넣어주면 맨 앞 bit0은 자연히 0으로 채워짐
- 마지막 진짜 데이터는 날아감 > 대부분 Fail
- 0x00 (0000_0000) 인 경우 : Pass

CHAPTER 03

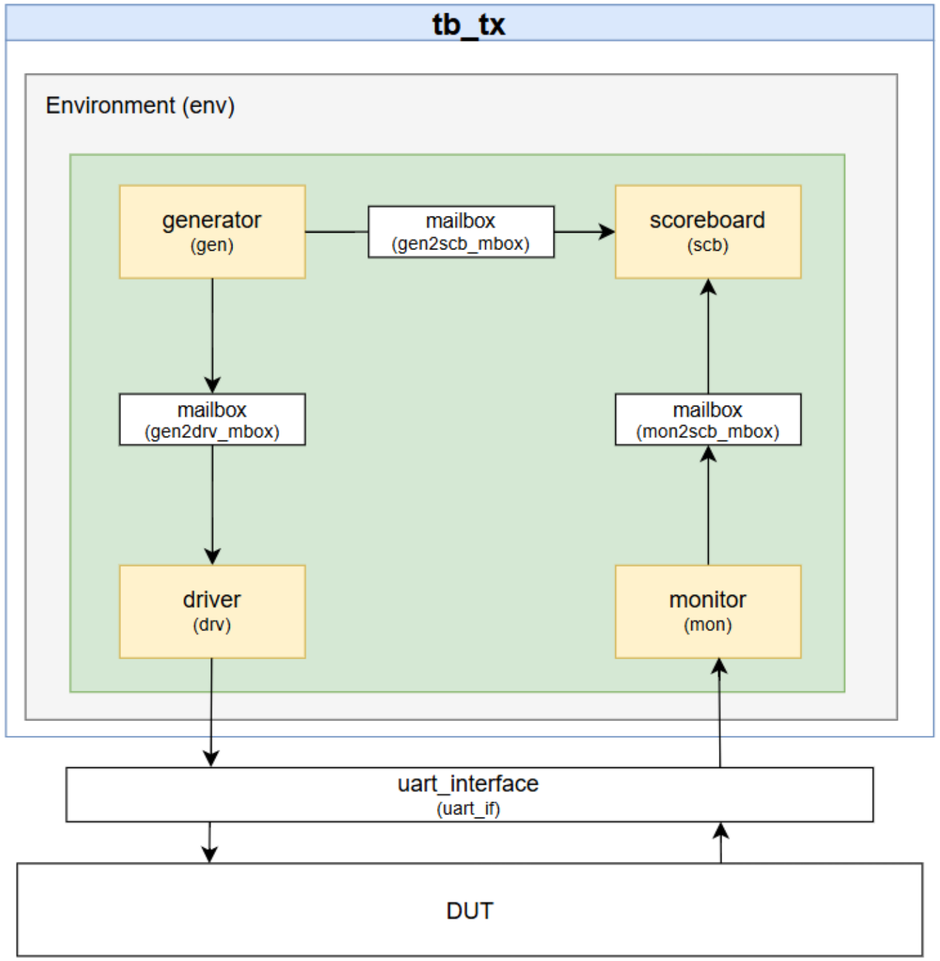
RX



- 드라이버의 tr.send_data 값 -> dut.Rx = 1 : rx_data에 1 들어옴 / Rx = 0 : rx_data 에 0 들어옴
- Rx_data : LSB부터 MSB까지 들어옴
- (MSB) 1101_0101 (LSB) : 1 -> 0 -> 1 -> 0 -> 1 -> 0 -> 1 -> 1 순

CHAPTER 04

TX



UART_TX

- Interface로 DUT와 Environment 연결
- Environment가 generator -> driver -> monitor -> scoreboard 구조 생성
- 4개의 class는 mailbox로 소통

CHAPTER 04

TX

```
class driver;
    transaction tr;
    mailbox #(transaction) gen2drv_mbox;
    virtual uart_tx_interface uart_tx_tb;
    event mon_next_event;

    function new(mailbox #(transaction) gen2drv_mbox,
                 virtual uart_tx_interface uart_tx_tb, event mon_next_event);
        this.gen2drv_mbox = gen2drv_mbox;
        this.uart_tx_tb = uart_tx_tb;
        this.mon_next_event = mon_next_event;
    endfunction //new()

    task reset();
        uart_tx_tb.rst = 1;
        uart_tx_tb.start_trigger = 0;
        repeat (2) @(posedge uart_tx_tb.clk);
        uart_tx_tb.rst = 0;
        repeat (2) @(posedge uart_tx_tb.clk);
        $display("[DRV] reset done");
    endtask

    task run();
        forever begin
            gen2drv_mbox.get(tr);
            tr.display("DRV");
            uart_tx_tb.start_trigger = 1;
            @(posedge uart_tx_tb.clk);
            uart_tx_tb.start_trigger = 0;
            -> mon_next_event;
            @(posedge uart_tx_tb.clk);
            uart_tx_tb.tx_data = tr.send_data;
            //@(posedge uart_tx_tb.tx_busy);
        end
    endtask
endclass
```

Driver

- driver : generator로 부터 mailbox를 통해 랜덤 데이터 받음
- dut의 start_trigger를 1clk 활성화 : 데이터 전송 시작하도록 dut에 신호
- transaction으로 생성한 랜덤 데이터를 dut의 tx_data 핀에 전달
- 다음 이벤트 처리를 위해 monitor에 event로 신호 전달

CHAPTER 04

TX

```
class monitor;
    transaction tr;
    mailbox #(transaction) mon2scb_mbox;
    virtual uart_tx_interface uart_tx_tb;
    event mon_next_event;

    function new(mailbox#(transaction) mon2scb_mbox,
                 virtual uart_tx_interface uart_tx_tb, event mon_next_event);
        this.mon2scb_mbox = mon2scb_mbox;
        this.uart_tx_tb = uart_tx_tb;
        this.mon_next_event = mon_next_event;
    endfunction  //new()

    task run();
        forever begin
            @(mon_next_event);
            tr = new();
            repeat (24) @(posedge uart_tx_tb.b_tick);
            for (int i = 0; i < 8; i++) begin
                tr.recieve_data[i] = uart_tx_tb.tx;
                repeat (16) @(posedge uart_tx_tb.b_tick);
            end
            tr.display("MON");
            repeat (16) @(posedge uart_tx_tb.b_tick);
            mon2scb_mbox.put(tr);
            //@(uart_tx_tb.tx_busy == 1'b0);
        end
    endtask
endclass
```

Monitor

- Monitor : driver에서 동작이 완료했다는 event를 기다림
- b_tick 의 repeat : dut의 각 비트 타이밍과 일치
- dut의 비트 타이밍에 맞춰 dut의 tx 데이터를 receive_data로 캡처
- mailbox를 통해 캡처한 data를 scoreboard로 보냄

CHAPTER 04

TX

```
class scoreboard;
    transaction tr, real_tr;
    mailbox #(transaction) mon2scb_mbox;
    mailbox #(transaction) gen2scb_mbox;
    event gen_next_event;

    logic [7:0] expected_data;
    logic [7:0] real_data;

    int Pass_count = 0;
    int Fail_count = 0;

    function new(mailbox#(transaction) mon2scb_mbox,
                 mailbox#(transaction) gen2scb_mbox, event gen_next_event);
        this.mon2scb_mbox   = mon2scb_mbox;
        this.gen2scb_mbox   = gen2scb_mbox;
        this.gen_next_event = gen_next_event;
    endfunction  //new()

    task run();
        forever begin
            mon2scb_mbox.get(tr);
            real_data = tr.recieve_data;
            gen2scb_mbox.get(real_tr);
            expected_data = real_tr.send_data;
            tr.display("SCB");

            if (expected_data == real_data) begin
                $display("[SCB] : Data matched : %d", real_data);
                $display("expected_data = %d, real_data = %d", expected_data,
                         real_data);
                Pass_count++;
            end else begin
                $display("[SCB] : Data mis-matched : %d, %d", real_data,
                         expected_data);
                $display("expected_data = %d, real_data = %d", expected_data,
                         real_data);
                Fail_count++;
            end

            ->gen_next_event;
        end
    endtask

endclass
```

```
[SCB] : Data matched :  91
expected_data =  91, real_data =  91
```

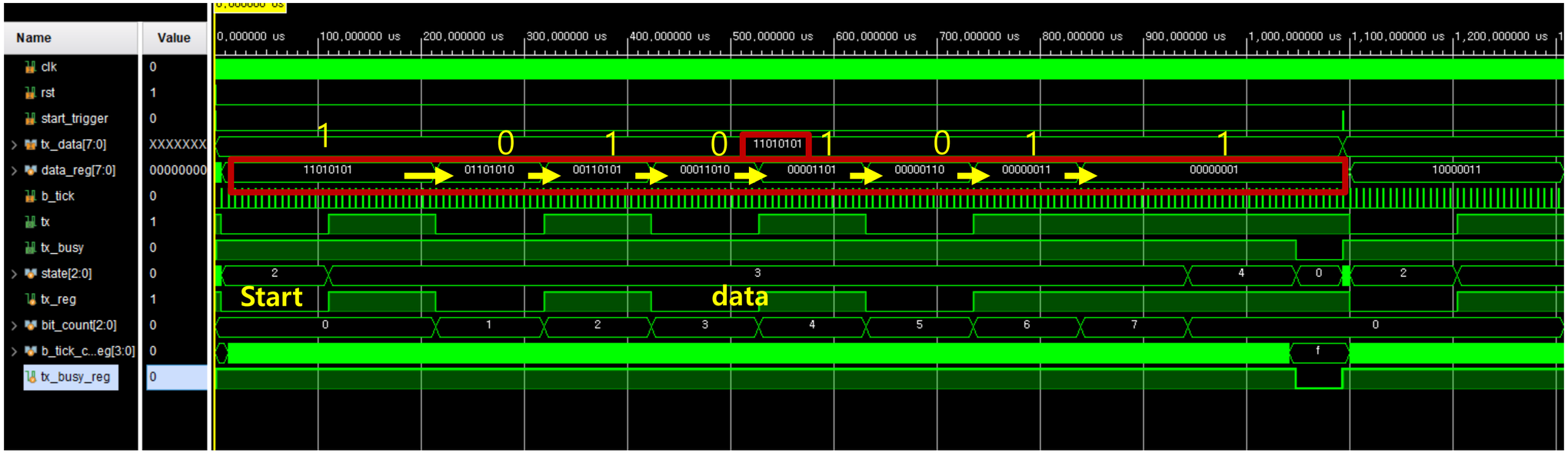
```
============================
======== test report ========
============================
==     Total Test :  256    ==
==     Pass Test :  256     ==
==     Fail Test :    0     ==
============================
==== Test bench is finish ===
============================
finished
```

Scoreboard

- Scoreboard : mailbox를 통해 monitor 와 generator 로부터 데이터를 받음
- generator 로부터 생성된 예상 데이터와 monitor 가 캡처한 실제 데이터를 비교
- 일치하면 data_matched 불일치 하면 dis-matched 라는 display 메시지를 출력
- 각각의 경우마다 Pass 혹은 Fail 카운트하여 리포트

CHAPTER 04

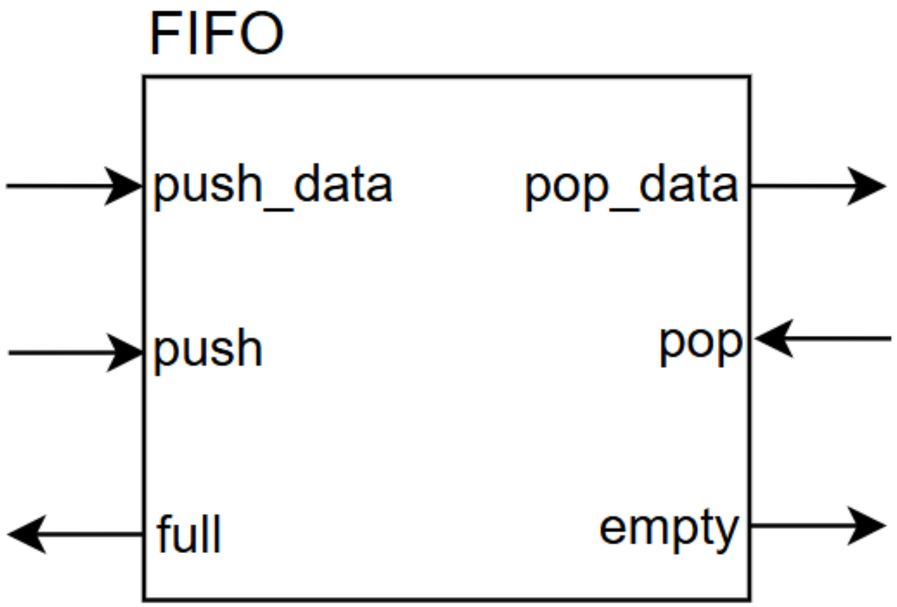
TX



- Tx_data 역시 LSB-first 로 들어옴 : (MSB) 1101_0101 (LSB)
- 전송 순서 : 1 -> 0 -> 1 -> 0 -> 1 -> 0 -> 1 -> 1 순 (순서대로 tx 핀을 통해 전송)
- dut -> data_next = data_reg >> 1 : 오른쪽으로 한 비트씩 이동 (shift)

CHAPTER 05

FIFO



- **FIFO (First In First Out)**
  : 새로운 데이터가 저장되거나 입력되면 오래된 데이터를 내보내고 새로운 데이터가 저장되는 방식

CHAPTER 05

FIFO

- **Interface**
- **Transaction**

```
3   interface fifo_interface;
4       logic       clk;
5       logic       rst;
6       logic       wr;
7       logic       rd;
8       logic [7:0] wdata;
9       logic [7:0] rdata;
10      logic       full;
11      logic       empty;
12  endinterface
13
14
15  class transaction;
16      // random stimulus
17      rand logic       wr;
18      rand logic       rd;
19      rand logic [7:0] wdata;
20
21      // for scoreboard
22      logic      [7:0] rdata;
23      logic            full;
24      logic            empty;
```

- DUT의 모든 입출력 포트를 하나의 logic 타입 interface로 정의.
- virtual interface로 Driver와 Monitor에서 DUT의 데이터(wr, rd, wdata, rdata, full, empty) 제어.
- rand 변수 (wr, rd, wdata)
  : Generator에서 생성하고
  Driver를 통해 DUT의 입력으로 인가.
- logic 변수 (rdata, full, empty)
  : DUT 출력을 Monitor가 받아 저장

CHAPTER 05

FIFO

- **Generator**

```
41  class generator;
42      transaction trans;
43      mailbox #(transaction) gen2drv_mbox;
44
45      event gen_next_event;
46
47      int total_count = 0;
48
49      function new(mailbox#(transaction) gen2drv_mbox, event gen_next_event);
50          this.gen2drv_mbox    = gen2drv_mbox;
51          this.gen_next_event = gen_next_event;
52      endfunction
53
54      task run(int count);
55          repeat (count) begin
56              total_count++;
57
58              trans = new();
59              assert (trans.randomize())
60              else $error("[GEN] trans.randomize() error !!! ");
61
62              gen2drv_mbox.put(trans);
63
64              trans.display("GEN");
65              // Receive event
66              @(gen_next_event);
67          end
68      endtask
69  endclass
```

```
class transaction;
    // random stimulus
    rand logic       wr;
    rand logic       rd;
    rand logic [7:0] wdata;
```

- generator에서 driver로 데이터를 보낼 mailbox 생성.
- Trans 데이터를 랜덤화하고 mailbox를 통해 랜덤 데이터 전송.
- scoreboard 동작이 마쳤음을 알리는 gen_next_event 신호 받음. Scoreboard가 이전 테스트에 대한 검증을 완전히 마치고 ->gen_next_event 신호를 보내줄 때까지 generator가 다음 transaction을 생성하지 못하게 함.

CHAPTER 05

FIFO

```
73  class driver;
74      transaction trans;
75      mailbox #(transaction) gen2drv_mbox;
76      virtual fifo_interface fifo_if;
77
78      event mon_next_event;
79
80      int i;
81
82      function new(mailbox#(transaction) gen2drv_mbox,
83                   virtual fifo_interface fifo_if, event mon_next_event);
84          this.gen2drv_mbox    = gen2drv_mbox;
85          this.fifo_if         = fifo_if;
86          this.mon_next_event = mon_next_event;
87      endfunction
```

- **Driver**

```
89      task reset();
90          fifo_if.clk   = 0;
91          fifo_if.rst   = 1;
92          fifo_if.wr    = 0;
93          fifo_if.rd    = 0;
94          fifo_if.wdata = 0;
95
96          repeat (2) @(posedge fifo_if.clk);
97          fifo_if.rst = 0;
98          repeat (2) @(posedge fifo_if.clk);
99          $display("[DRV] reset done!");
100         //#1;
101     endtask
102
103     task run();
104         forever begin
105             #1
106             gen2drv_mbox.get(trans);
107
108             fifo_if.wr    = trans.wr;
109             fifo_if.rd    = trans.rd;
110             fifo_if.wdata = trans.wdata;
111
112             trans.display("DRV");
113             #2;
114             ->mon_next_event;
115             @(posedge fifo_if.clk);
116         end
117     endtask
118 endclass
```

- mailbox를 통해 trans 데이터 받음.
  받은 데이터를 interface의 데이터에 대입.
- Driver 동작이 마쳤음을 monitor에게 알림.

CHAPTER 05

FIFO

- **Monitor**

```
121  class monitor;
122      transaction trans;
123      virtual fifo_interface fifo_if;
124      mailbox #(transaction) mon2scb_mbox;
125      event mon_next_event;
126
127      function new(mailbox#(transaction) mon2scb_mbox,
128                   virtual fifo_interface fifo_if, event mon_next_event);
129          this.mon2scb_mbox    = mon2scb_mbox;
130          this.fifo_if         = fifo_if;
131          this.mon_next_event = mon_next_event;
132      endfunction
133
134      task run();
135          forever begin
136              @(mon_next_event);
137              trans     = new;
138              trans.wr    = fifo_if.wr;
139              trans.rd    = fifo_if.rd;
140              trans.wdata = fifo_if.wdata;
141              trans.rdata = fifo_if.rdata;
142              trans.full  = fifo_if.full;
143              trans.empty = fifo_if.empty;
144              trans.display("MON");
145              mon2scb_mbox.put(trans);
146              @(posedge fifo_if.clk);
147          end
148      endtask
149  endclass
```

- Driver가 보낸 mon_next_event가 발생할 때만 동작.
- DUT로부터 출력된 결과값을 trans 변수에 대입 후, mailbox를 통해 scoreboard로 전송.

CHAPTER 05

FIFO

- **Scoreboard**

```
152 class scoreboard;
153     transaction trans;
154     mailbox #(transaction) mon2scb_mbox;
155
156     event gen_next_event;
157
158     logic [7:0] fifo_queue[$:15];
159     logic [7:0] expected_data;
160
161     int pass_count = 0, fail_count = 0;
162
163     function new(mailbox#(transaction) mon2scb_mbox, event gen_next_event);
164         this.mon2scb_mbox    = mon2scb_mbox;
165         this.gen_next_event = gen_next_event;
166     endfunction
```

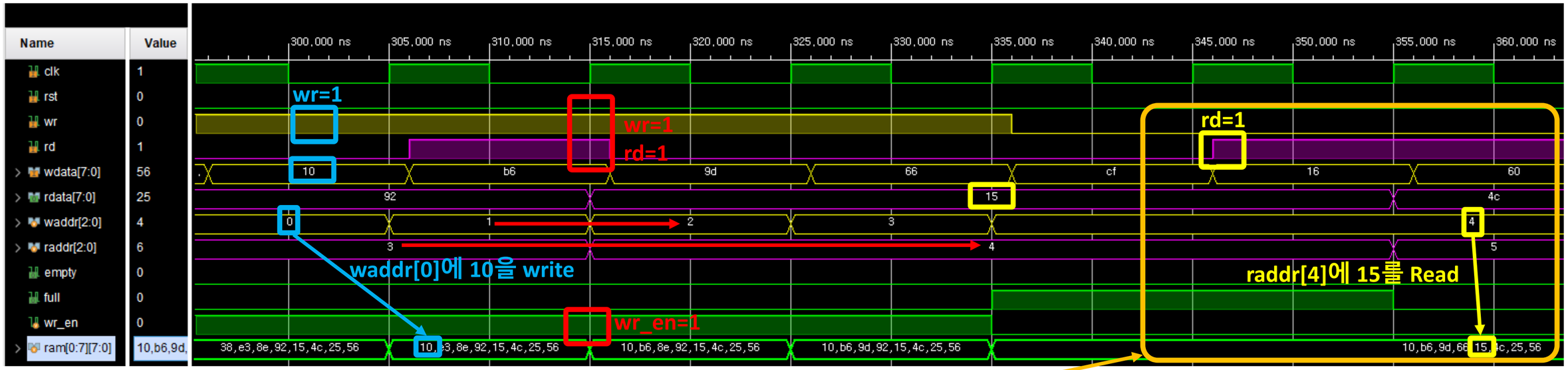
```
168     task run();
169         forever begin
170             mon2scb_mbox.get(trans)
171
172             trans.display("SCB");
173             if (trans.wr) begin
174                 if (!trans.full) begin
175                     fifo_queue.push_back(trans.wdata);
176                     $display("[SCB] : Data stored in Queue : %d, size :%d",
177                              trans.wdata, fifo_queue.size());
178                 end else begin
179                     $display("[SCB] : Queue is full : %d", fifo_queue.size());
180                 end
181             end
```

```
183             if (trans.rd) begin
184                 if (!trans.empty) begin
185                     expected_data = fifo_queue.pop_front();
186                     if (trans.rdata == expected_data) begin
187                         pass_count++;
188                         $display("[SCB] : Data matched : %d", trans.rdata);
189                     end else begin
190                         fail_count++;
191                         $display(
192                             "[SCB] : Data mis-matched : rdata=%d, expected data=%d",
193                             trans.rdata, expected_data);
194                     end
195                 end else begin
196                     $display("[SCB] FIFO is Empty");
197                 end
198             end
199
200             $display("--------------------------------");
201             $display("%p", fifo_queue);
202             $display("--------------------------------");
203
204             ->gen_next_event;
205         end
206     endtask
207 endclass
```

- Mailbox로 전송되는 데이터를 받음.
- FIFO에 넣은 값(push_back)과 FIFO로부터 나온 값(pop_front)을 비교하여 PASS/FAIL을 판정.
- 작업이 모두 끝나고 generator에게 gen_next_event 신호를 보내 다음 작업을 수행하게 함.

CHAPTER 05

FIFO

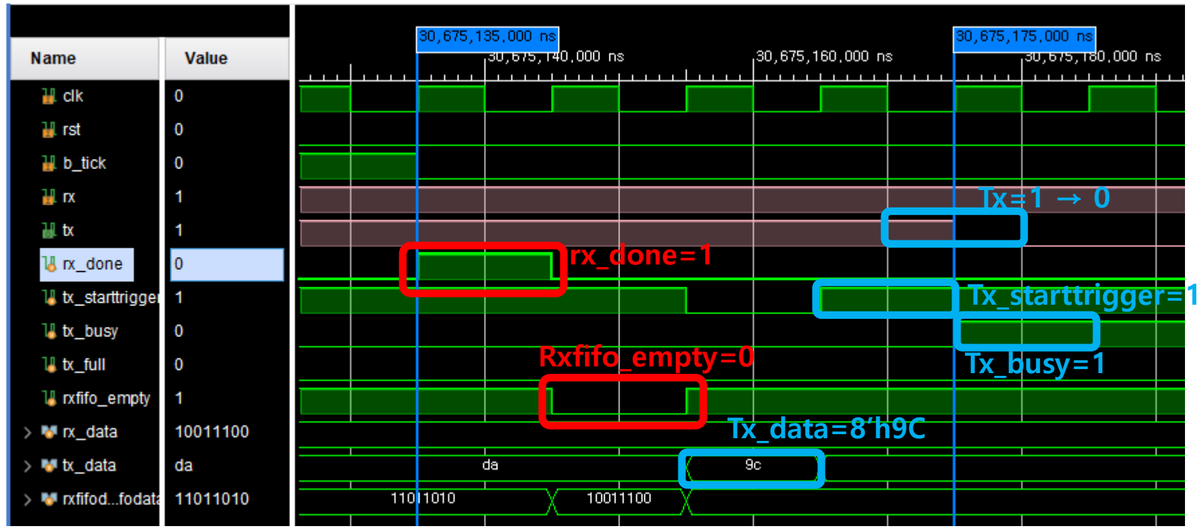


- wdata 8'h10를 waddr[0]에 write.
- Rdata 8'h15를 rdata[4]에서 read.

- wr_en = 0인 상태에서 write는 되지 않지만, read는 가능.
- wr_en = 1인 상태(wr = 1)에서 rd가 1일 때, waddr와 raddr을 +1 하고 wdata값이 ram에 저장

CHAPTER 05

FIFO



- rx_done이 1 clk 동안 high로 뛰는 펄스는 uart_rx 모듈이 외부 1바이트의 rx 데이터 수신 마쳤음을 알림.
  또, RX_FIFO의 push로 작동해 high이던 rxfifo_empty 신호가 low가 되는데, 받은 데이터가 RX_FIFO에 저장되어 비어 있지 않음을 의미.

- tx_data가 준비되면 tx_start trigger 신호를 high로 만들어 uart_tx 모듈에게 데이터를 보내라고 명령. 이에 uart_tx 모듈은 두 가지 동작을 수행.
  1. tx_busy 신호를 high로 만들어 현재 송신 중임을 알림.
  2. tx 출력 핀을 high(IDLE)에서 low로 떨어뜨려, 데이터 전송의 시작을 알리는 Start Bit 생성.

CHAPTER 05

FIFO

wr=1, rd=1

```
                368000, [GEN] : wr = 1, rd = 1, wdata = 189, rdata =   x, full = x, empty = x
                376000, [DRV] : wr = 1, rd = 1, wdata = 189, rdata =   x, full = x, empty = x
                378000, [MON] : wr = 1, rd = 1, wdata = 189, rdata =  86, full = 0, empty = 0
                378000, [SCB] : wr = 1, rd = 1, wdata = 189, rdata =  86, full = 0, empty = 0
[SCB] : Data stored in Queue : 189, size :         6
[SCB] : Data matched :  86
--------------------------------
'{16,182,157,102,189}
--------------------------------
```

wr=0, rd=1

```
                378000, [GEN] : wr = 0, rd = 1, wdata =  92, rdata =   x, full = x, empty = x
                386000, [DRV] : wr = 0, rd = 1, wdata =  92, rdata =   x, full = x, empty = x
                388000, [MON] : wr = 0, rd = 1, wdata =  92, rdata =  16, full = 0, empty = 0
                388000, [SCB] : wr = 0, rd = 1, wdata =  92, rdata =  16, full = 0, empty = 0
[SCB] : Data matched :  16
--------------------------------
'{182,157,102,189}
--------------------------------
```

wr=1, rd=0

```
                388000, [GEN] : wr = 1, rd = 0, wdata = 106, rdata =   x, full = x, empty = x
                396000, [DRV] : wr = 1, rd = 0, wdata = 106, rdata =   x, full = x, empty = x
                398000, [MON] : wr = 1, rd = 0, wdata = 106, rdata = 182, full = 0, empty = 0
                398000, [SCB] : wr = 1, rd = 0, wdata = 106, rdata = 182, full = 0, empty = 0
[SCB] : Data stored in Queue : 106, size :         5
--------------------------------
'{182,157,102,189,106}
--------------------------------
```

wr=1, rd=1

```
                398000, [GEN] : wr = 1, rd = 1, wdata = 174, rdata =   x, full = x, empty = x
                406000, [DRV] : wr = 1, rd = 1, wdata = 174, rdata =   x, full = x, empty = x
                408000, [MON] : wr = 1, rd = 1, wdata = 174, rdata = 182, full = 0, empty = 0
                408000, [SCB] : wr = 1, rd = 1, wdata = 174, rdata = 182, full = 0, empty = 0
[SCB] : Data stored in Queue : 174, size :         6
[SCB] : Data matched : 182
--------------------------------
'{157,102,189,106,174}
--------------------------------
```

wr=0, rd=0

```
                408000, [GEN] : wr = 0, rd = 0, wdata = 141, rdata =   x, full = x, empty = x
                416000, [DRV] : wr = 0, rd = 0, wdata = 141, rdata =   x, full = x, empty = x
                418000, [MON] : wr = 0, rd = 0, wdata = 141, rdata = 157, full = 0, empty = 0
                418000, [SCB] : wr = 0, rd = 0, wdata = 141, rdata = 157, full = 0, empty = 0
--------------------------------
'{157,102,189,106,174}
--------------------------------
```

wr=1, rd=0

```
                418000, [GEN] : wr = 1, rd = 0, wdata = 170, rdata =   x, full = x, empty = x
                426000, [DRV] : wr = 1, rd = 0, wdata = 170, rdata =   x, full = x, empty = x
                428000, [MON] : wr = 1, rd = 0, wdata = 170, rdata = 157, full = 0, empty = 0
                428000, [SCB] : wr = 1, rd = 0, wdata = 170, rdata = 157, full = 0, empty = 0
[SCB] : Data stored in Queue : 170, size :         6
--------------------------------
'{157,102,189,106,174,170}
--------------------------------
```

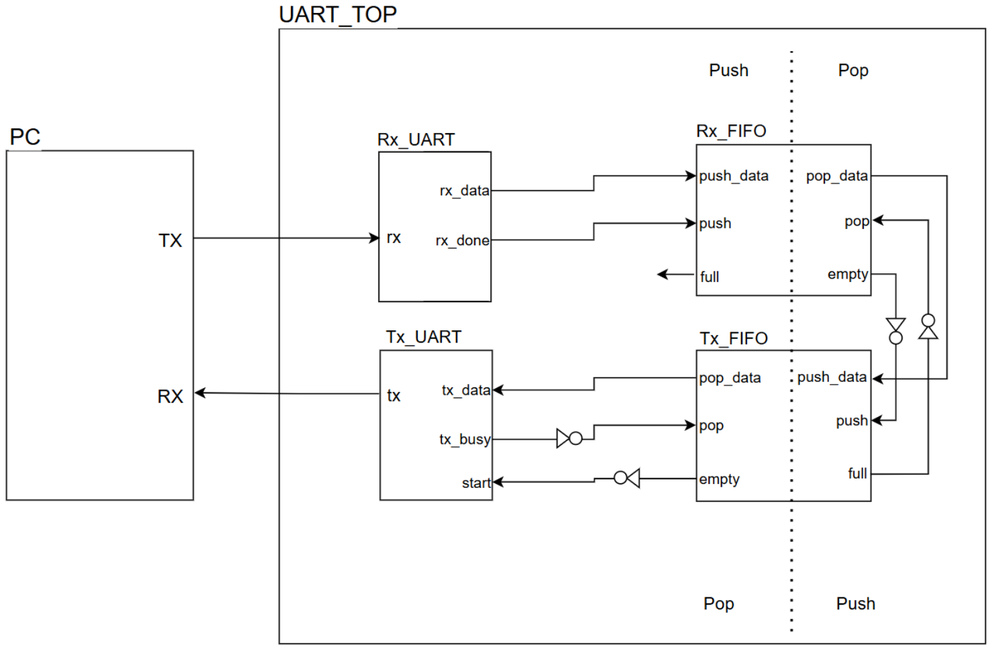
```
==================================
========== test report ==========
==================================
==    Total Test :          50   ==
==    Pass Test  :          26   ==
==    Fail Test  :           0   ==
==================================
==     Testbench is finished    ==
==================================
Finished
```

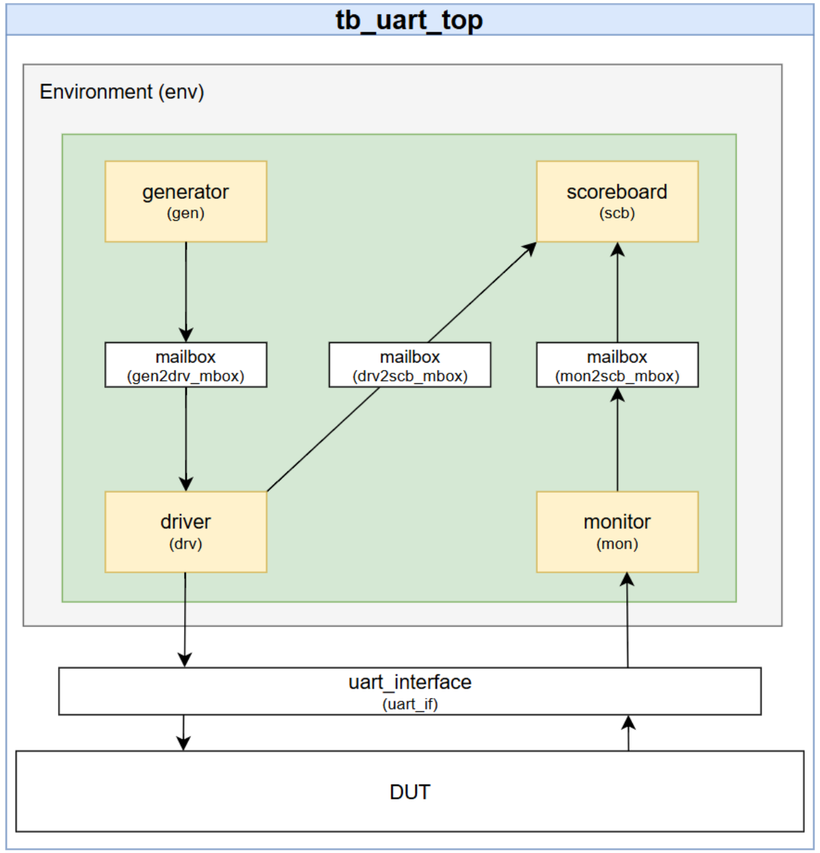
```
if (trans.rd) begin
    if (!trans.empty) begin
        expected_data = fifo_queue.pop_front();
        if (trans.rdata == expected_data) begin
            pass_count++;
            $display("[SCB] : Data matched : %d", trans.rdata);
        end else begin
            fail_count++;
            $display(
                "[SCB] : Data mis-matched : rdata=%d, expected data=%d",
                trans.rdata, expected_data);
        end
    end else begin
        $display("[SCB] FIFO is Empty");
    end
end
```

- rd=1 이고 empty=0일 때만 고려하여 Pass도 Fail도 아닌 상태가 발생

CHAPTER 06

UART TOP





CHAPTER 06

UART TOP

- **Interface**
- **Transaction**

```
3   interface uart_interface;
4       logic clk;
5       logic rst;
6       logic rx;
7       logic tx;
8   endinterface
9
10
11  class transaction;
12      rand logic [7:0] send_data;
13
14      task display(string name_s);
15          $display("%t, [%s] : data = %b", $time, name_s, send_data);
16      endtask
17  endclass
```

- DUT의 모든 입출력 포트를 interface에 logic 변수들로 정의.
- virtual interface로 Driver와 Monitor에서 DUT의 데이터(rx, tx) 제어.
- transaction 클래스에서 랜덤 8bit send_data를 정의했는데, DUT를 통해 처리된 작업을 확인할 원본 데이터임.
- 추후 원본 데이터와 비교할 tx_data 를 정의하기 위해서도 사용.

CHAPTER 06

UART TOP

- **Generator**

```
20  class generator;
21      transaction trans;
22      mailbox #(transaction) gen2drv_mbox;
23
24      event gen_next_event;
25
26      function new(mailbox#(transaction) gen2drv_mbox, event gen_next_event);
27          this.gen2drv_mbox   = gen2drv_mbox;
28          this.gen_next_event = gen_next_event;
29      endfunction
30
31      task run(int count);
32          repeat (count) begin
33              trans = new();
34              assert (trans.randomize())
35              else $error("[GEN] trans.randomize() error !!! ");
36
37              gen2drv_mbox.put(trans);
38
39              trans.display("GEN");
40
41              @(gen_next_event);
42          end
43      endtask
44  endclass
```

- Random 데이터를 가진 transaction 객체를 생성하여 gen2drv_mbox라는 mailbox에 담아 Driver에게 전달.
- Scoreboard에서 작업이 끝나고 generator에게 gen_next_event 신호를 보내는데 이 신호를 받음. scoreboard에서 작업이 모두 끝나고 generator가 시작하도록 일러줌.

CHAPTER 06

UART TOP

- **Driver**

```
47  class driver;
48      transaction trans;
49      mailbox #(transaction) gen2drv_mbox;
50      mailbox #(transaction) drv2scb_mbox;
51      virtual uart_interface uart_if;
52
53      parameter BIT_PERIOD = 104160;
54
55      function new(mailbox#(transaction) gen2drv_mbox,
56                   virtual uart_interface uart_if, mailbox#(transaction) drv2scb_mbox);
57          this.gen2drv_mbox = gen2drv_mbox;
58          this.uart_if      = uart_if;
59          this.drv2scb_mbox = drv2scb_mbox;
60      endfunction
61
62
63      task reset();
64          uart_if.clk = 0;
65          uart_if.rst = 1;
66          uart_if.rx  = 1;
67
68          repeat (2) @(posedge uart_if.clk);
69          uart_if.rst = 0;
70          repeat (2) @(posedge uart_if.clk);
71          $display("[DRV] reset done!");
72      endtask
73
74      task run();
75          forever begin
76              #1 gen2drv_mbox.get(trans);
77
78              drv2scb_mbox.put(trans);
79              trans.display("DRV");
80
81              send(trans.send_data);
82          end
83      endtask
```

```
85      task send(input [7:0] send_data);
86          // start bit
87          uart_if.rx = 1'b0;
88          #(BIT_PERIOD);
89
90          // data bit
91          for (int k = 0; k < 8; k++) begin
92              uart_if.rx = send_data[k];
93              #(BIT_PERIOD);
94          end
95
96          // Stop Bit
97          uart_if.rx = 1'b1;
98          #(BIT_PERIOD);
99      endtask
100 endclass
```

- Mailbox를 통해 Generator로부터 데이터를 받음.
- 1.
  받은 데이터를 Scoreboard에 전달하여 원본 데이터(=보낼 데이터) 값을 알려줌
- 2.
  send task로 rx_data를 interface에 입력할 rx로 만듦.

CHAPTER 06

UART TOP

- **monitor**

```
104  class monitor;
105      transaction trans;
106      virtual uart_interface uart_if;
107      mailbox #(transaction) mon2scb_mbox;
108
109      parameter BIT_PERIOD = 104160;
110
111      function new(mailbox#(transaction) mon2scb_mbox,
112                   virtual uart_interface uart_if);
113          this.mon2scb_mbox = mon2scb_mbox;
114          this.uart_if      = uart_if;
115      endfunction
116
117      task run();
118          forever begin
119              @(negedge uart_if.tx);
120
121              receive();
122          end
123      endtask
```

```
125      task receive();
126          trans = new;
127
128          $display("          receive start");
129          #(BIT_PERIOD / 2);  // middle of start bit
130
131          // start bit pass/fail
132          if (!uart_if.tx) begin
133              for (int j = 0; j < 8; j++) begin
134                  #(BIT_PERIOD);
135                  trans.send_data[j] = uart_if.tx;
136              end
137
138              trans.display("MON");
139
140              // check stop bit
141              #(BIT_PERIOD);
142              if (uart_if.tx) begin
143                  mon2scb_mbox.put(trans);
144              end
145          end
146      endtask
147  endclass
```

- DUT의 tx를 보다가 receive task를 통해 start 신호(tx=0)가 들어오면 tx 신호 해석을 시작 -> tx를 8bit tx_data로 복원.
- 8bit까지 모두 받고고 stop 신호(tx=1)를 받으면 그 결과를 mon2scb_mbox를 통해 Scoreboard에 전달.

CHAPTER 06

UART TOP

- **Scoreboard**

```
150  class scoreboard;
151      transaction trans;
152      transaction tr;
153      mailbox #(transaction) mon2scb_mbox;
154      mailbox #(transaction) drv2scb_mbox;
155      event gen_next_event;
156
157      int pass_count = 0, fail_count = 0, total_count = 0;
158
159      function new(mailbox#(transaction) mon2scb_mbox, event gen_next_event,
160                   mailbox#(transaction) drv2scb_mbox);
161          this.mon2scb_mbox   = mon2scb_mbox;
162          this.gen_next_event = gen_next_event;
163          this.drv2scb_mbox   = drv2scb_mbox;
164      endfunction
```

```
167      task run();
168          forever begin
169              drv2scb_mbox.get(tr);  // from driver
170              mon2scb_mbox.get(trans); //  from monitor
171              trans.display("SCB");
172
173              if (trans.send_data == tr.send_data) begin
174                  pass_count++;
175                  $display("[SCB] data matched! rx_data : %b == tx_data : %b",
176                           tr.send_data, trans.send_data);
177                  $display(
178                      "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
179              end else begin
180                  fail_count++;
181                  $display(
182                      "[SCB] data mis-matched... rx_data : %b != tx_data : %b",
183                      tr.send_data, trans.send_data);
184                  $display(
185                      "~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
186              end
187
188              total_count++;
189              ->gen_next_event;
190          end
191      endtask
192  endclass
```

- 두 Mailbox로부터 데이터가 도착하면 Driver로부터 받은 '예상값'(=보낸 값)과 Monitor로부터 받은 '실제 결과값'을 비교하여 PASS/FAIL을 판정.
- 작업이 모두 끝나고 generator에게 gen_next_event 신호를 보낸다.

CHAPTER 06

UART TOP

- **Environment**

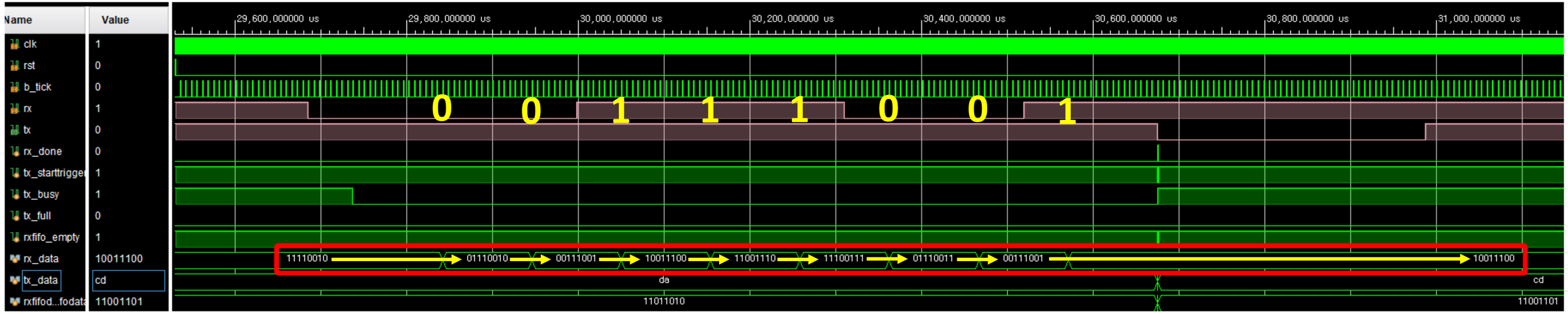
```
195  class environment;
196      mailbox #(transaction) gen2drv_mbox;
197      mailbox #(transaction) mon2scb_mbox;
198      mailbox #(transaction) drv2scb_mbox;
199      event gen_next_event;
200
201      generator gen;
202      driver drv;
203      monitor mon;
204      scoreboard scb;
205
206      function new(virtual uart_interface uart_if);
207          gen2drv_mbox = new();
208          mon2scb_mbox = new();
209          drv2scb_mbox = new();
210
211          gen = new(gen2drv_mbox, gen_next_event);
212          drv = new(gen2drv_mbox, uart_if, drv2scb_mbox);
213          mon = new(mon2scb_mbox, uart_if);
214          scb = new(mon2scb_mbox, gen_next_event, drv2scb_mbox);
215      endfunction
```

```
219      task report();
220          $display("====================================");
221          $display("=========== test report ===========");
222          $display("====================================");
223          $display("==    Total Test : %d   ==", scb.total_count);
224          $display("==    Pass Test  : %d   ==", scb.pass_count);
225          $display("==    Fail Test  : %d   ==", scb.fail_count);
226          $display("====================================");
227          $display("==     Testbench is finished     ==");
228          $display("====================================");
229      endtask
230
231      task run();
232          drv.reset();
233          fork
234              gen.run(256);
235              drv.run();
236              mon.run();
237              scb.run();
238          join_any
239          #10;
240          report();
241          $display("Finished");
242          $stop;
243      endtask
244  endclass
```

- 지금까지 설명한 모든 컴포넌트와 Mailbox들을 생성하고, 서로 연결하여 전체 테스트 환경을 구축하고 실행.
- run task에서 fork - join을 통해 모든 컴포넌트들이 동시에 각자의 역할을 수행하도록 하여 전체 검증 시나리오를 실행시킴.

CHAPTER 06

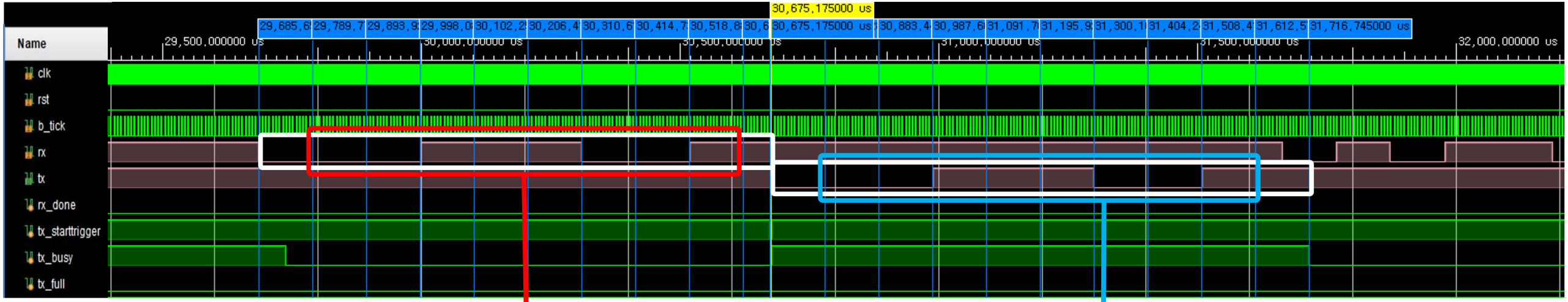
UART TOP

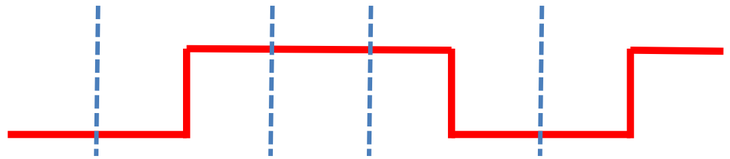


- Rx 신호가 1bit씩 push되어 rx_data를 구성하며 bit shift가 발생.

CHAPTER 06

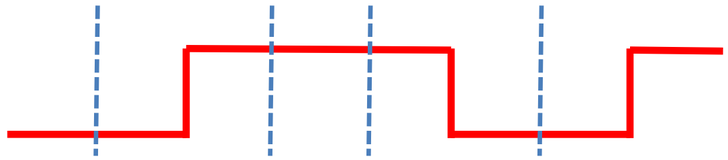
UART TOP





0 0 1 1 1 0 0 1

(LSB) → 1100(=C) 1001(=9) (MSB)



0 0 1 1 1 0 0 1

(LSB) → 1100(=C) 1001(=9) (MSB)

CHAPTER 06

UART TOP

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        411640375000, [GEN] : data = 00111001
        411640375000, [DRV] : data = 00111001
          receive start
        413515255000, [MON] : data = 00111001
        413619415000, [SCB] : data = 00111001
[SCB] data matched! rx_data : 00111001 == tx_data : 00111001
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        413619415000, [GEN] : data = 01001100
        413619415000, [DRV] : data = 01001100
          receive start
        415494295000, [MON] : data = 01001100
        415598455000, [SCB] : data = 01001100
[SCB] data matched! rx_data : 01001100 == tx_data : 01001100
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        415598455000, [GEN] : data = 11110100
        415598455000, [DRV] : data = 11110100
          receive start
        417473335000, [MON] : data = 11110100
        417577495000, [SCB] : data = 11110100
[SCB] data matched! rx_data : 11110100 == tx_data : 11110100
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        417577495000, [GEN] : data = 10111010
        417577495000, [DRV] : data = 10111010
          receive start
        419452375000, [MON] : data = 10111010
        419556535000, [SCB] : data = 10111010
[SCB] data matched! rx_data : 10111010 == tx_data : 10111010
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        419556535000, [GEN] : data = 00101111
        419556535000, [DRV] : data = 00101111
          receive start
        421431415000, [MON] : data = 00101111
        421535575000, [SCB] : data = 00101111
[SCB] data matched! rx_data : 00101111 == tx_data : 00101111
```

```
====================================
========== test report ===========
====================================
==    Total Test :         256   ==
==    Pass Test  :         256   ==
==    Fail Test  :           0   ==
====================================
==     Testbench is finished     ==
====================================
Finished
```

CHAPTER 07

고찰

Event 제어

- Monitor는 Driver로부터의 이벤트 없이 DUT의 tx 출력을 확인하는 방식으로 (@(negedge tx)) 구현.

- 반면, Generator와 Scoreboard 사이에는 이벤트가 필요했음.
- Scoreboard가 하나의 케이스 비교를 완료한 후에만 Generator에게 다음 데이터를 생성하라는 신호(->gen_next_event)를 보내는 Feedback Loop를 구현.

Transaction의 변수 간소화

- Mailbox와 Transaction을 2개 설정해 Transaction의 send_data를 예상값과 실제값에 모두 사용할 수 있었음.

CHAPTER 08

Trouble Shooting

문제 발생

```
task run();
    forever begin

        gen2scb_mbox.get(tr);
        tr.display("SCB");
        mon2scb_mbox.get(real_tr);
```

- Scoreboard 에서 generator에서 보내는 mailbox를 받은 후 바로 display를 하게 되면 monitor에서 “put” 하는 데이터를 받기 전에 SCB 로그가 먼저 찍힘 > generator 에서 바로 scoreboard 로 흘러가는 것 처럼 보임.
- 로그 착시로 검증 흐름을 오해 → 디버깅 시간 증가

문제 해결

```
task run();
    forever begin

        mon2scb_mbox.get(real_tr);
        gen2scb_mbox.get(tr);
        tr.display("SCB");
```

- Scoreboard 에서 display는 두 개의 get이 모두 끝난 후 수행
- 검증 흐름 명확화 → 디버깅 시간 감소
- 로그가 찍은 결과와 동일한 라운드를 출력하여 착시 제거