

UVM VERIFICATION LINE BUFFER

EDA Playgroud: <https://edaplayground.com/x/bnEr>

Contents

Part 1. Introduction

- Design Spec
- Design Verification

Part 2. Structure

- Verification Scenario
- UVM Verification Environment

Part 3. Data Flow

- Stimulus Generation
- Data Flow
- Port

Part 4. Validation Logic

- Scoreboard
- Waveform

Part 5. Conclusion

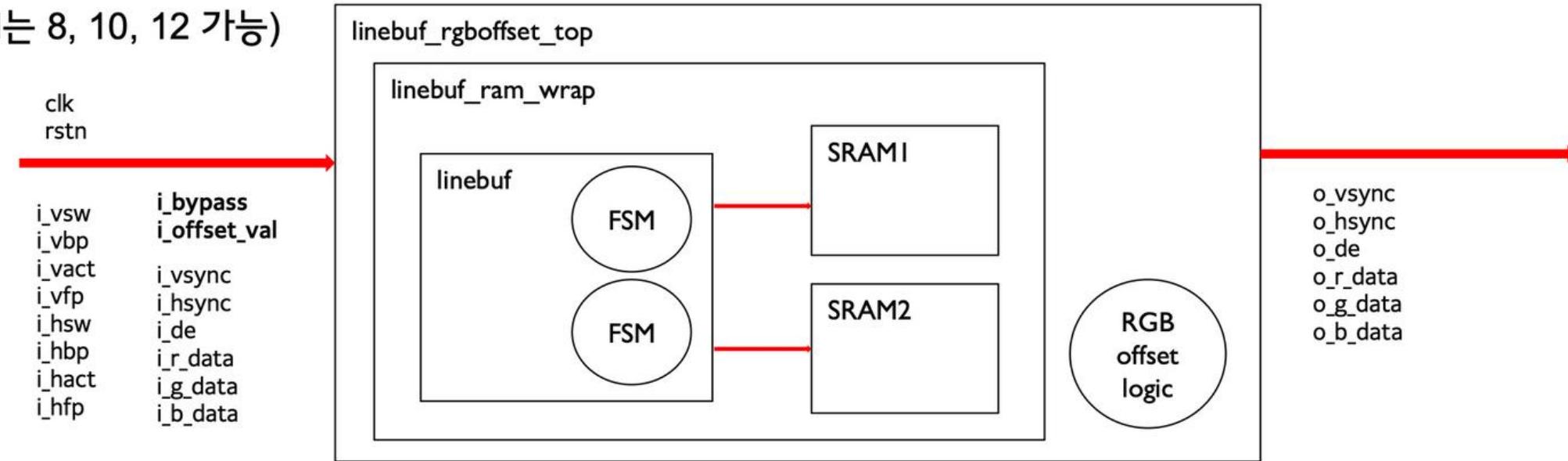
- Trouble Shooting
- Result

01 Introduction

Design Spec

Design Spec

(RGB_WIDTH는 8, 10, 12 가능)



✓ Target Design

입력된 Video Signal을 function 설정 값에 따라 Bypass 또는 offset 처리 후 출력하는 IP

✓ Function

- Bypass Mode
- Offset Mode
- 초기 세팅 후 Mode 선택값 변동없음

Design Spec

Function

✓ Bypass Mode

- 첫 번째 i_vsync가 동작하기 전에 i_bypass = HIGH
- Line buffer 및 offset 처리없이 Video Signal 입력을 1 Clock delay 후 그대로 출력

✓ Offset Mode

- 첫 번째 i_vsync가 동작하기 전에 i_bypass = LOW
- Line buffer(1 line + 1 clock delay)
- 각 R, G, B data에 i_offset_val(offset value)를 더해서 출력
- 더한 값이 bit size 최대값을 초과하면 최대값으로 출력

✓ 제한사항

- i_bypass와 i_offset_val은 동시에 세팅되며 처음 세팅 이후 변동되지 않는다.
- (i_vsw, i_vbp, …도 이 신호들과 동시에 세팅해야 하며 변동되지 않음)

Design Verification

✓ 검증 요구사항

✓ Random Frame Size Test

- 다양한 Frame Size(VSW, VBP, ...)에서 DUT가 정상 동작하는지 확인

✓ Random RGB Data Test

- 다양한 RGB Data가 입력되었을 때 DUT가 정상 동작하는지 확인 (RGB_WIDTH 고려)
- Random Data : 매 clock마다 R, G, B data에 Random한 값이 입력될 경우
- Fix Data : R, G, B data에 고정된 값이 입력될 경우
- Increasing Data : 한 frame 동안 매 clock마다 R, G, B data가 0부터 bit size 최대값까지 증가하며, 최대 치를 넘을 경우 다시 0으로 순환 반복되는 값이 입력될 경우 (매 frame마다 0으로 다시 시작)

✓ Mode Test

- DUT의 bypass, offset 기능이 정상 동작하는지 확인한다.

✓ 참고사항

*3가지 검증 요구사항에서 hsync, vsync, de 신호의 delay가 올바른지 check할 필요는 없으며, Output data만 Input data 대비 잘 출력되었는지 확인하면 된다.

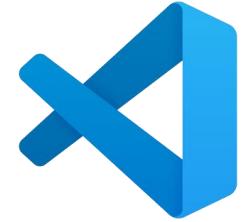
구현 환경

✓ Language



- System Verilog

✓ Tool



synopsys®

cadence™

- VS CODE
- Synopsys VCS
- EDAPlayground
- Cadence xcelium 23.09

02 TEST Structure

Verification Scenario

✓ Scenario 1 : Bypass Mode 검증

- 입력 데이터가 Line Buffer의 Offset 연산 로직을 거치지 않고, 입력 값 그대로 변화 없이 출력되는지 검증한다.

✓ Scenario 2 : Offset Mode 검증

- 입력 데이터에 대한 연산 결과가 최댓값을 초과하는 경우, 오버플로우 없이 최댓값으로 고정되는지 검증한다.
- 공통 시나리오
 1. Random RGB Data: 매 클럭마다 무작위 값을 입력.
 1. Fix RGB Data: 프레임 전체 기간 동안 고정된 단일 값을 입력.
 1. Increasing RGB Data: 0부터 최대값까지 1씩 증가하며, 최대값 초과 시 0으로 순환하는 데이터 입력

Verification Scenario

Bypass Mode

- 검증 test case :
1. 다양한 Frame size에서 DUT가 정상 동작 확인(Frame size Random 생성).
 2. Bypass 모드 활성화 시, 설정된 Offset 값이 연산에 반영되지 않고 무시됨을 확인한다.

	Random Data	Fix Data	Increasing Data
data_mode	RANDOM	FIX	INCREASE
user_bypass	1	1	1
offset_value	user_offset_val	user_offset_val	user_offset_val
i_data	rand_data	fix_data	count
expected_data	rand_data	fix_data	count

Verification Scenario

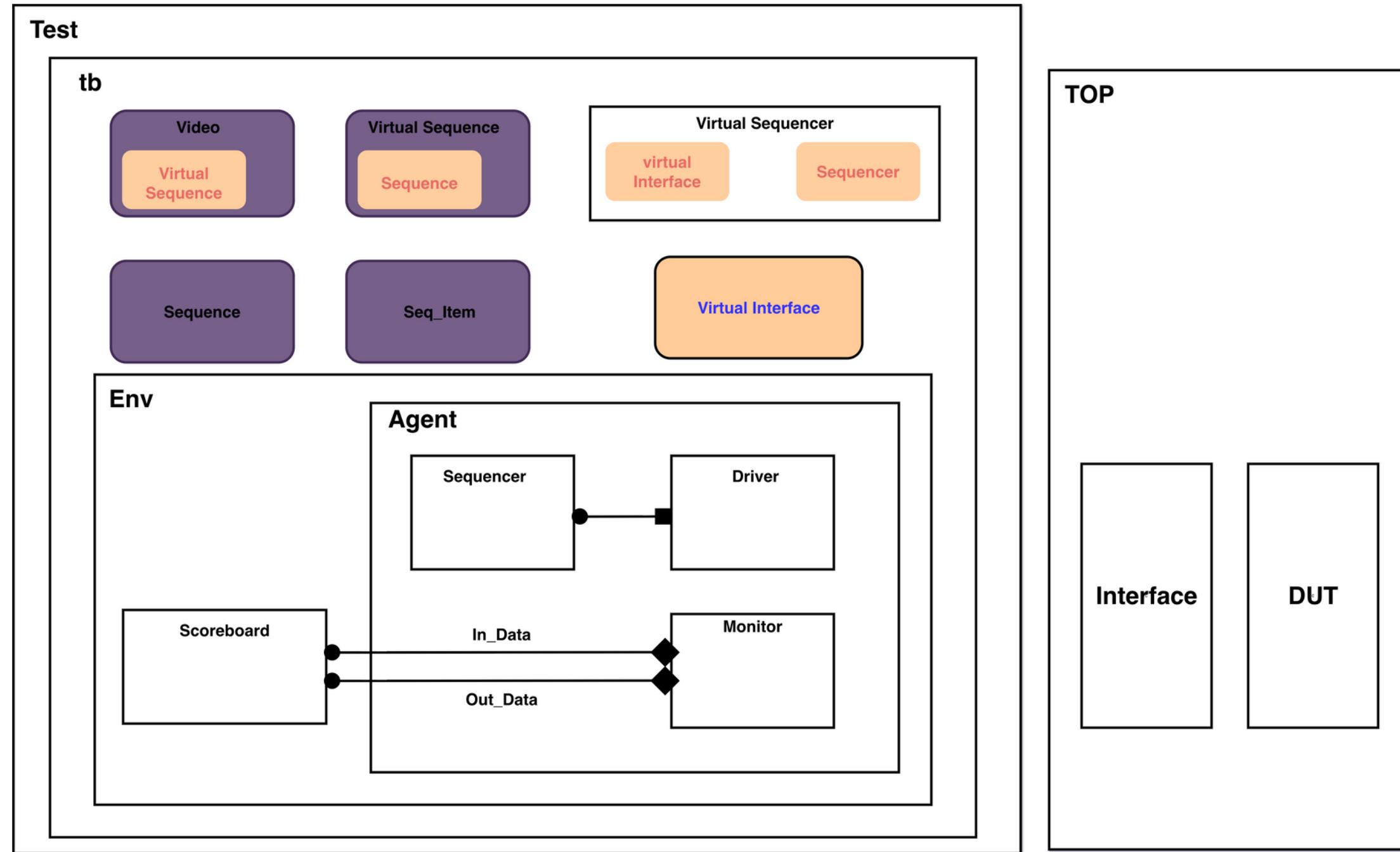
Offset Mode

- 검증 test case :
 1. 다양한 Frame size에서 DUT가 정상 동작하는지 확인(Frame size Random 생성).
 2. Line Buffer 사용 시 발생하는 delay가 설계 사양과 일치하는지 검증한다.
 3. 연산 결과가 최댓값을 초과하는 경우, 오버플로우 없이 최댓값으로 고정되는지 검증한다

	Random Data	Fix Data
data_mode	RANDOM	
user_bypass	0	
offset_value	user_offset_val	user_fix_val
i_data	rand_data	fix_data
expected_data	rand_data + user_offset_val	fix_data + i_data

UVM Structure

Block Diagram



Check List

Component

- Test
- tb
- Virtual Sequencer
- Env(Scoreboard)
- Agent(Sequencer, Driver, Monitor)

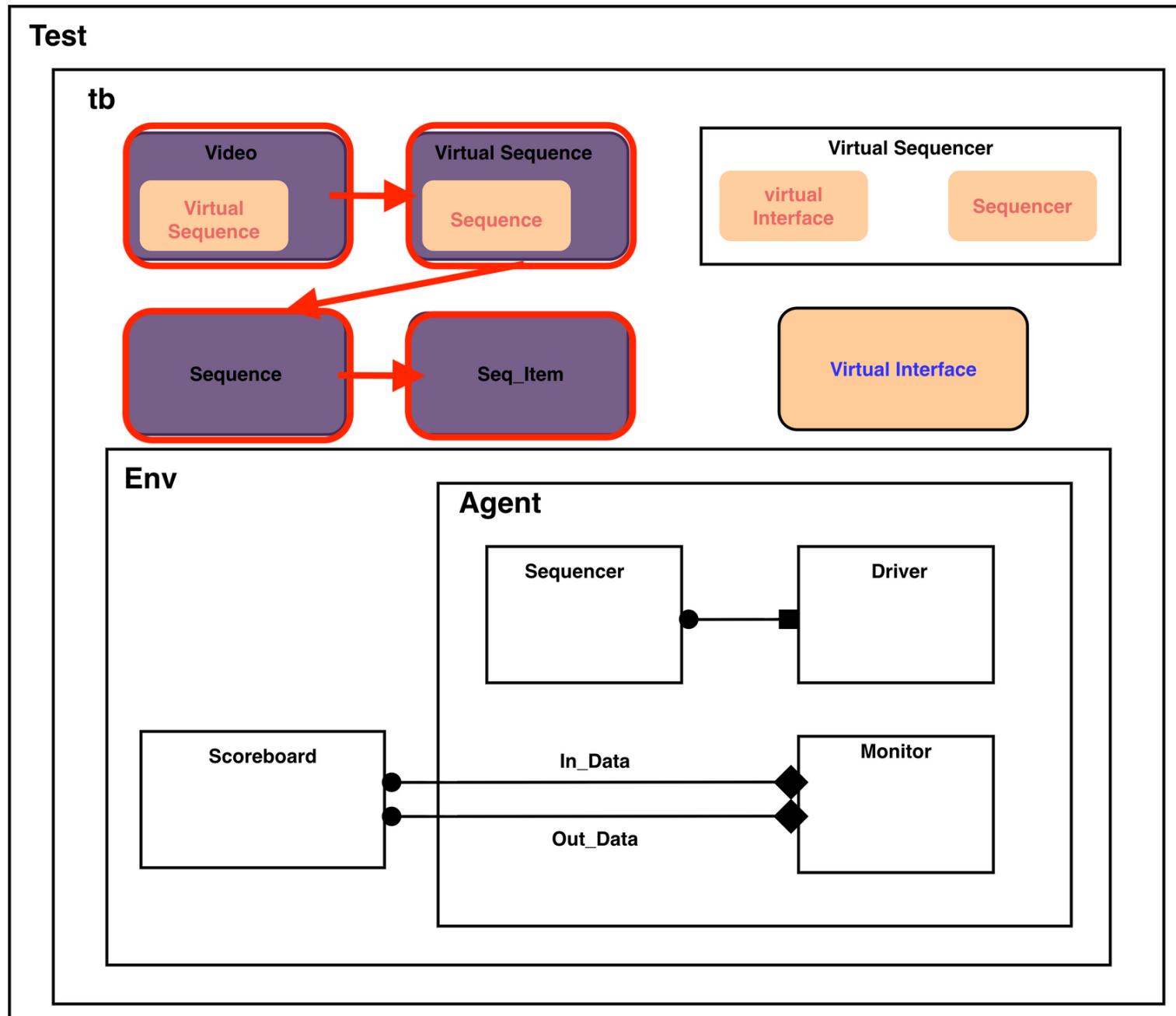
Object

- Virtual Sequence
- sequence
- seq_item

03 Data Flow

Stimulus Generation

✓ Stimulus Generation



✓ Video

- Video에서 Fixed data와 같은 설정값들(mode, vsw, vbp, ...)을 사용자가 원하는 시나리오 맞춰서 설정한다.

✓ Virtual Sequence

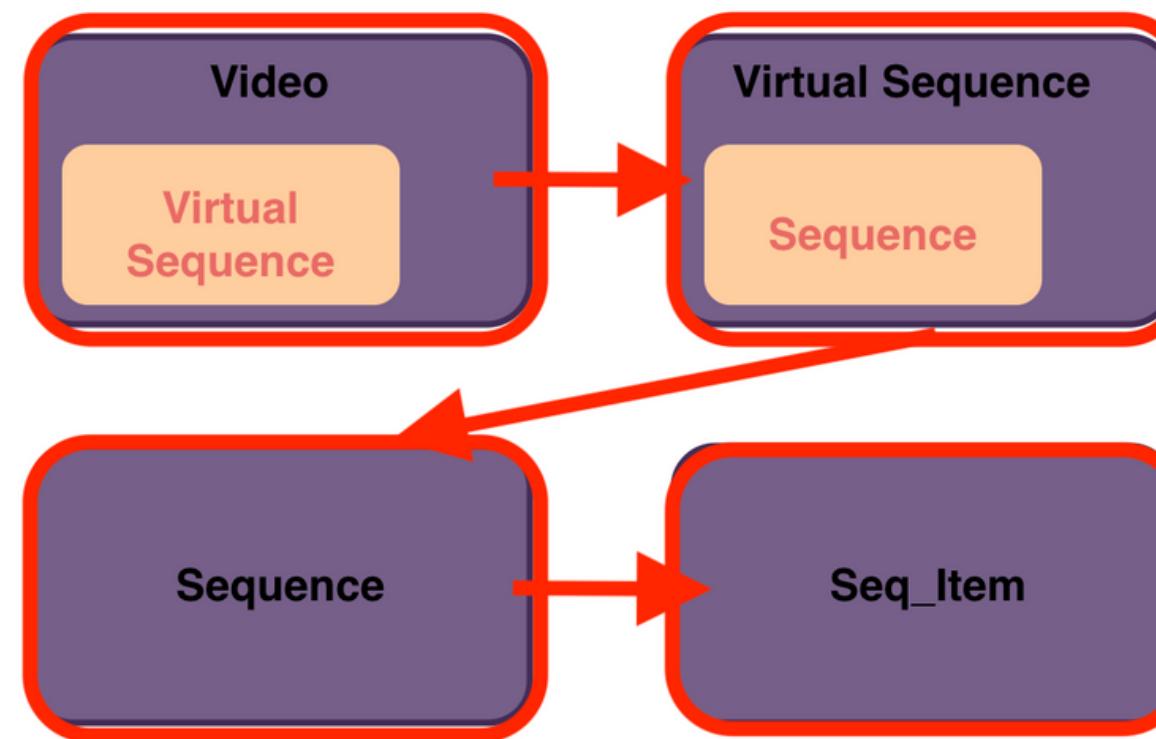
- Video에서 설정한 값들이 Virtual Sequence에서 handler를 통해 Sequence에 전달한다.
- 이때 전달된 값을 바탕으로 Mode1(Bypass, Offset)과 Mode2(Random, Fixed, Increase)를 결정한다.

✓ Sequence

- Video에서 온 데이터에 맞는 Mode1, 2를 결정짓고 데이터를 준비시키고 Driver와 Hand Shake를 통해 전달 한다.

Stimulus Generation

Sequence Logic



✓ Base Sequence (Timing Control)

- Generate Video Timing
 - h_cnt와 v_cnt(Counter-based Logic)를 활용하여 Sync signal 생성
- Check data enable
 - h_cnt와 v_cnt를 활용하여 Active Region Control
- Send Signal
 - Data를 담아서 전송

✓ User Sequence (Pattern Generator)

- Send Frame
 - Data Mode Select (Random, Fix, Increase)해서 해당 데이터와 send signal을 통해 데이터 생성

Stimulus Generation

Sequence Logic

```
virtual task generate_video_timing(lb_ro_drv_pkt_c pkt);
    int v_total = pkt.i_vsw + pkt.i_vbp + pkt.i_vact + pkt.i_vfp;
    int h_total = pkt.i_hsw + pkt.i_hbp + pkt.i_hact + pkt.i_hfp;

    if (h_total == 0 || v_total == 0) begin
        `uvm_error(get_type_name(), "invalid timing parameters: zero total")
        return;
    end

    pkt.i_vsync = (v_cnt < pkt.i_vsw);
    pkt.i_hsync = (h_cnt < pkt.i_hsw);

    pkt.i_de = check_data_enable(rnd_item);

    h_cnt++;
    if (h_cnt >= h_total) begin
        h_cnt = 0;
        v_cnt++;
        if (v_cnt >= v_total) begin
            v_cnt = 0;
        end
    end
endtask

virtual function bit check_data_enable(lb_ro_seq_item_c item);
    return ((v_cnt >= (item.i_vsw + item.i_vbp)) &&
            (v_cnt < (item.i_vsw + item.i_vbp + item.i_vact)) &&
            (h_cnt >= (item.i_hsw + item.i_hbp)) &&
            (h_cnt < (item.i_hsw + item.i_hbp + item.i_hact)));
endfunction
```

✓ Base Sequence (Timing Control)

- Generate Video Timing
 - h_cnt와 v_cnt(Counter-based Logic)를 활용하여 Sync signal 생성
- Check data enable
 - h_cnt와 v_cnt를 활용하여 Active Region Control
- Send Signal
 - Data를 담아서 전송

Stimulus Generation

Sequence Logic

```
task send_signal();
    `uvm_info(get_type_name(), $sformatf("send_signal starts.."),
    `uvm_create(req);
    req.i_bypass = rnd_item.i_bypass;
    req.i_offset_val = rnd_item.i_offset_val;
    req.i_vsw = rnd_item.i_vsw;
    req.i_vbp = rnd_item.i_vbp;
    req.i_vact = rnd_item.i_vact;
    req.i_vfp = rnd_item.i_vfp;
    req.i_hsw = rnd_item.i_hsw;
    req.i_hbp = rnd_item.i_hbp;
    req.i_hact = rnd_item.i_hact;
    req.i_hfp = rnd_item.i_hfp;
    generate_video_timing(req);
    if (req.i_de) begin
        req.i_r_data = rnd_item.i_r_data;
        req.i_g_data = rnd_item.i_g_data;
        req.i_b_data = rnd_item.i_b_data;
    end else begin
        req.i_r_data = 0;
        req.i_g_data = 0;
        req.i_b_data = 0;
    end
    `uvm_send(req);
    `uvm_info(get_type_name(), $sformatf("send_signal ends.."), U
endtask
```

✓ Base Sequence (Timing Control)

- Send Signal
 - Video에서 설정한 값이 Virtual Sequence를 통해서 넣어준 값 할당
 - User sequence에서 rnd_item에 넣어준 RGBMode에 따른 Data를 req Data에 할당
 - 이를 마치고 난 후에 마지막 uvm_send를 통해서 Driver에게 전송하는 로직 구현

Stimulus Generation

Sequence Logic

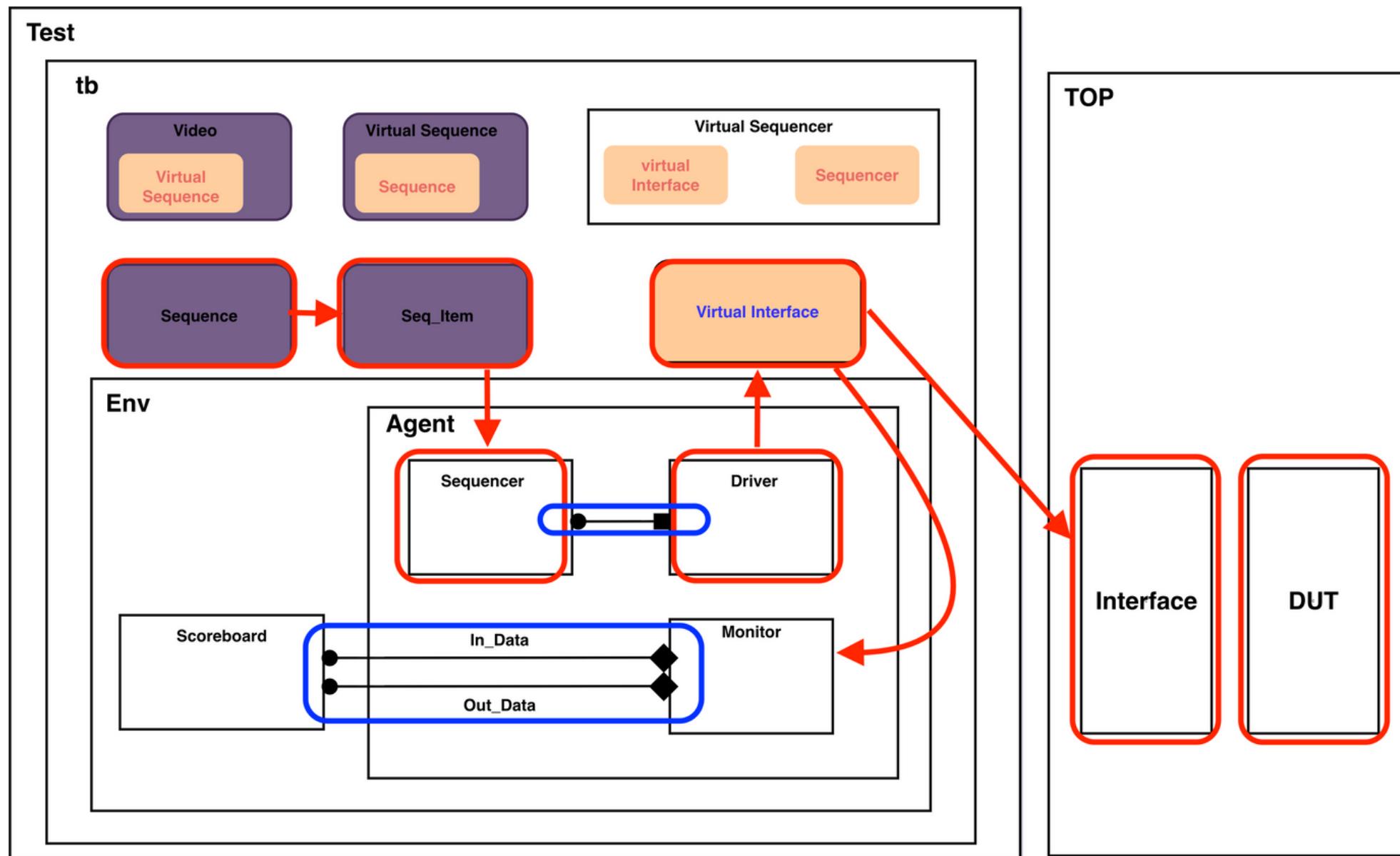
```
task send_frame(input int num_frames = 1);
    int h_total = rnd_item.i_hsw + rnd_item.i_hbp + rnd_item.i_hact + rnd_item.i_hfp;
    int v_total = rnd_item.i_vsw + rnd_item.i_vbp + rnd_item.i_vact + rnd_item.i_vfp;
    int total_pixels = h_total * v_total;
    for (int f = 0; f < num_frames; f++) begin
        v_cnt = 0;
        h_cnt = 0;
        for (int i = 0; i < total_pixels; i++) begin
            if (check_data_enable(rnd_item)) begin
                case (data_mode)
                    FIX: begin
                        rnd_item.i_r_data = fix_r_data;
                        rnd_item.i_g_data = fix_g_data;
                        rnd_item.i_b_data = fix_b_data;
                    end
                    INCREASE: begin
                        rnd_item.i_r_data = r_cnt;
                        rnd_item.i_g_data = g_cnt;
                        rnd_item.i_b_data = b_cnt;
                        r_cnt = (r_cnt == 1023) ? 0 : r_cnt + 1;
                        g_cnt = (g_cnt == 1023) ? 0 : g_cnt + 1;
                        b_cnt = (b_cnt == 1023) ? 0 : b_cnt + 1;
                    end
                    RANDOM: begin
                        void'(rnd_item.randomize() with {
                            i_r_data inside {[0 : 1023]};
                            i_g_data inside {[0 : 1023]};
                            i_b_data inside {[0 : 1023]};});
                    end
                endcase
            end else begin
                rnd_item.i_r_data = 0;
                rnd_item.i_g_data = 0;
                rnd_item.i_b_data = 0;
            end
            send_signal();
        end
    end
endtask
```

✓ User Sequence (Pattern Generator)

- Send Frame
 - Check data enable이 1일 때 RGB데이터 생성하고 아니면 RGB Data 0으로 할당
 - Data Mode Select (Random, Fix, Increase)해서 Mode에 맞는 RGB데이터 생성 후 rnd_item에 할당
 - Data 모두 생성한 이후에 send_signal을 통해서 Video에서 설정한 값과 생성한 RGB Data를 Driver로 전송

Data Flow

✓ Data Flow



✓ Sequence와 Driver

- Handshake(`uvm_do_on_with`)을 통해서 sequence의 Data가 Driver로 전달.

✓ Driver와 Interface

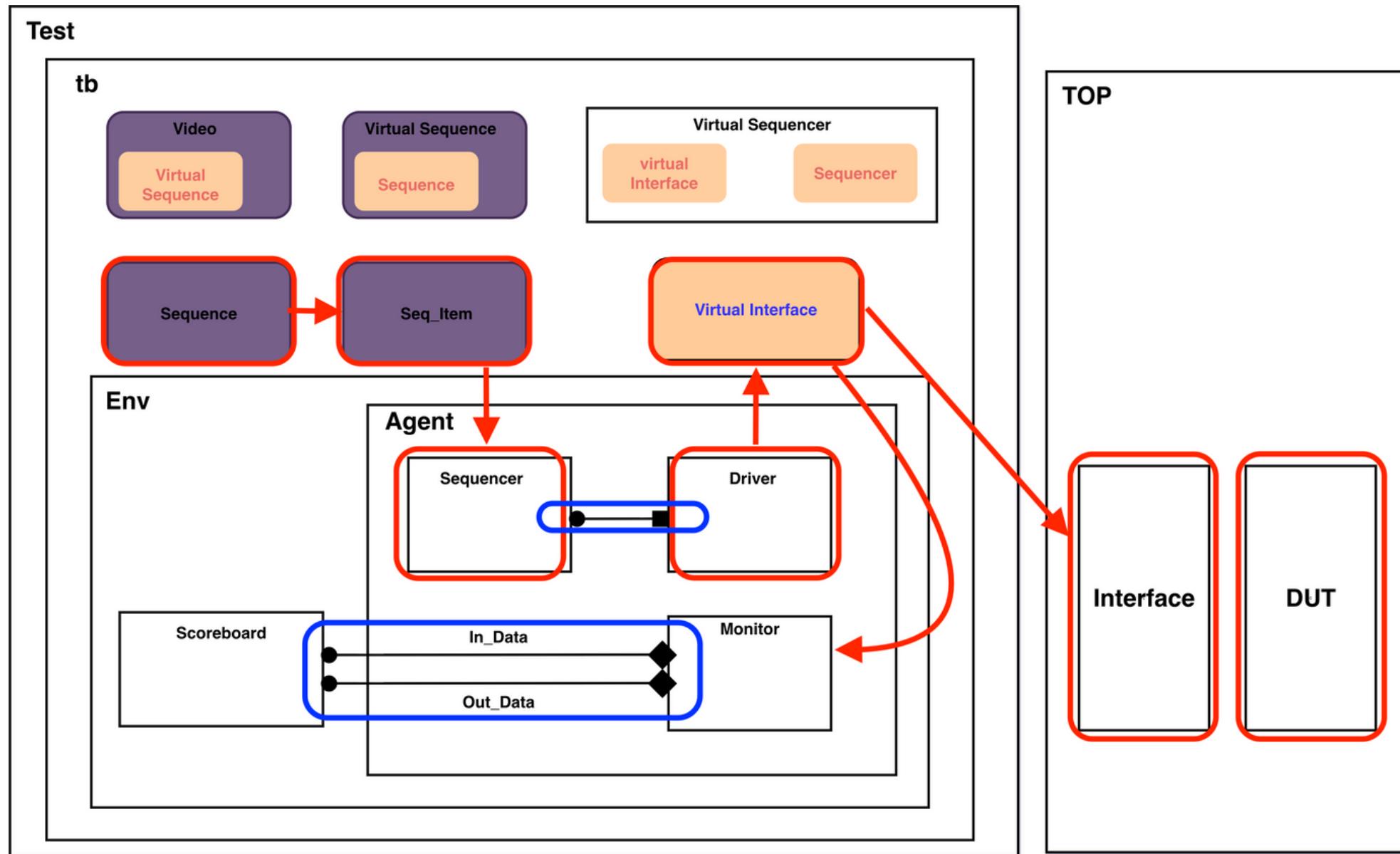
- 전달하는 Data는 Virtual Interface를 통해서 Interface로 Data를 전달.

✓ Monitor와 Scoreboard

- Virtual Interface를 통해서 Interface의 Input Data, Output Data를 Monitor로 받아 와 해당 Data를 Scoreboard로 전달

Port

✓ Port



✓ TLM (Pull Mode)

- Sequencer와 Driver를 연결해주는 물리적 Port는 Pull Mode Port이다.
- Producer는 Driver이고 Consumer은 Sequencer이다.
- 이때 Handshak에 맞춰서 데이터가 전달된다.

✓ TLM (Broadcast Mode)

- Monitor와 Scoreboard를 연결시켜주는 포트는 Analysis Port이다
- Producer는 Monitor이고 Consumer는 Scoreboard이다.

04 Validation Logic

Scoreboard

✓ 비교 검증의 목적

✓ 무엇을 검증해야 하는가?

1. Pixel 검증
 - a. RGB 각 채널 값이 정확한가?
 - b. Offset 연산이 올바른가?
 - c. Saturation이 제대로 동작하는가? ($255 + \text{offset} > 255 \rightarrow 255$ 로 제한)
2. Line 검증
 - a. 라인당 픽셀 개수가 맞는가?
 - b. 라인 내 모든 픽셀이 정확한가?
 - c. 라인 경계 검출이 올바른가? (DE falling edge 타이밍)
3. Frame 검증
 - a. 프레임당 라인 개수가 맞는가?
 - b. 프레임 내 모든 라인이 정확한가?
 - c. 프레임 경계 검출이 올바른가? (VSYNC falling edge 타이밍)

Scoreboard

✓ 비교 검증의 목적

✓ 어떻게 검증할 것인가? → 3-Tier 검증 아키텍처

: 계층적 검증으로 빠진 에러 없이 완벽한 검증

Pixel Level → 개별 데이터 정확도 검증



Line Level → 구조적 무결성 검증



Frame Level → 전체 시스템 검증

✓ 검증 방식

병렬 실행: 3개 레벨 동시 비교

독립성: 각 레벨이 독립적인 Queue 사용

완전성: 모든 픽셀을 3가지 관점에서 검증

Scoreboard

Pixel

```
virtual task compare_pixels();
    rgb_pixel_t c_pixel, dut_pixel;
    int pixel_num = 0;
    forever begin
        wait (c_pixel_q.size() > 0 && dut_pixel_q.size() > 0);
        c_pixel = c_pixel_q.pop_front();
        dut_pixel = dut_pixel_q.pop_front();
        pixel_num++;
        compare_pixel_data(c_pixel, dut_pixel, pixel_num);
    end
endtask
```

검증 방식

Pixel

- Expected data Queue와 Dut Data Queue가 있을 때까지 대기
- Pop Front를 통해서 데이터 꺼낸 이후에 해당 데이터 비교
- Pixel_num Counter값 증가시키고 다음 pixel값 비교 시작

Compare Timing

- 수집 조건: DE=1 (Level Check)
- 비교 시점: Queue에 데이터가 있으면 즉시

Scoreboard

✓ Line

```
virtual task compare_lines();
    rgb_line_t c_line, dut_line;
    int line_num = 0;
    forever begin
        wait (c_line_q.size() > 0 && dut_line_q.size() > 0);
        c_line = c_line_q.pop_front();
        dut_line = dut_line_q.pop_front();
        line_num++;
        compare_line_data(c_line, dut_line, line_num);
    end
endtask
```

✓ 검증 방식

✓ Line

- Expected data Queue와 Dut Data Queue가 있을 때까지 대기
- Pop Front를 통해서 Line데이터 꺼낸 이후에 pixel 개수, 각 pixel 값 비교(2단계 검증)
- Error 추적 : Line LH error pixel 위치 report

✓ Compare Timing

- 수집 과정: DE=1일 때 버퍼에 축적
- 완성 조건: DE Falling Edge
- 비교 시점: Queue에 라인이 추가되면 즉시

Scoreboard

Frame 비교

```
virtual task compare_frames();
    rgb_frame_t c_frame, dut_frame;
    int frame_num = 0;
    forever begin
        wait (c_frame_q.size() > 0 && dut_frame_q.size() > 0);
        c_frame = c_frame_q.pop_front();
        dut_frame = dut_frame_q.pop_front();
        frame_num++;
        compare_frame_data(c_frame, dut_frame, frame_num);
    end
endtask
```

검증 방식

Frame

- Expected data Queue와 Dut Data Queue가 있을 때까지 대기
- Pop Front를 통해서 데이터 꺼낸 이후에 해당 데이터 비교
- Frame Size(Line 개수), Line Size(Pixel 개수), pixel value 비교

Compare Timing

- 수집 과정: DE Falling Edge마다 라인을 프레임 버퍼에 추가
- 완성 조건: VSYNC Falling Edge
- 비교 시점: Queue에 프레임이 추가되면 즉시

Scoreboard

✓ Queue 관리

✓ 3-Level Queue Architecture

	Pixel	Line	Frame
Queue 차원	1D	2D	3D
중간 버퍼	없음	current_c_line	current_c_frame
Push 시점	DE=1 (즉시)	DE falling	VSYNC falling
Pop data	픽셀 1개	라인 1개 (픽셀 배열)	프레임 1개 (라인 배열)
사이즈 체크	불필요	필수 (픽셀 개수)	필수 (라인/픽셀 개수)

- Pixel Queue: Pixel 단위 비교 (C Module 과 DUT Pixel 저장)
- Line Queue: Line 단위 비교 (완성된 라인들을 저장)
- Frame Queue: Frame 단위 비교 (완성된 프레임들을 저장)

Scoreboard

✓ Error Handling

✓ 에러 감지 방식

1. Pixel Error
 - a. 픽셀 2개를 비교하여 RGB 값이 하나라도 다르면 에러 감지
2. Line Error (3단계)
 - a. 사이즈 체크: C_Model과 DUT의 픽셀 개수 불일치 → 즉시 에러 출력 후 리턴
 - b. 픽셀별 비교: 라인 내 모든 픽셀을 순회하며 비교, 에러 발견해도 계속 진행 (모든 에러 검출)
 - c. 결과 판정: 에러가 하나라도 있으면 라인 FAILED, 에러 개수 요약 출력
3. Frame Error (3단계)
 - a. 사이즈 체크: C_Model과 DUT의 라인 개수 불일치 → 즉시 에러 출력 후 리턴
 - b. 라인별 순회: 각 라인의 사이즈 체크 후 사이즈 다르면 에러 출력 후 다음 라인, 같으면 픽셀 비교
 - c. 결과 판정: 에러가 하나라도 있으면 프레임 FAILED, 에러 개수 요약 출력

Scoreboard

✓ Error Handling

✓ 에러 출력 정보

1. Pixel Error
 - a. 픽셀 번호, C Model 값, DUT 값
2. Line Error
 - a. **라인 번호**, 픽셀 번호, C Model 값, DUT 값, **에러 개수**
3. Frame Error (3단계)
 - a. **프레임 번호**, 라인 번호, 픽셀 번호, C Model 값, DUT 값, 에러 개수

Scoreboard

Error Handling

핵심 강점

1. 정확한 위치 식별
 - a. 프레임/라인/픽셀 번호로 에러 위치 정확히 식별
 - b. 디버깅 시 바로 해당 위치로 이동 가능
2. 상세한 값 비교
 - a. C Model과 DUT 값을 모두 출력
 - b. 어느 채널(R/G/B)에서 에러인지 명확히 표시
3. 즉시 보고
 - a. 에러 발견 즉시 로그 출력
 - b. 실시간으로 문제 파악 가능
4. 계층적 추적
 - a. 하나의 에러를 Pixel/Line/Frame 3가지 레벨에서 동시 추적하여 전체적인 에러 패턴 파악 가능
 - b. 에러 개수를 계층별로 집계하여 전체 시뮬레이션 품질 평가 가능

Scoreboard

✓ Error Handling

✓ 실제 에러 케이스

Scenario: Pixel 10 → R+1 에러

```
if (pixel_count == 10) begin
    out_pkt.o_r_data = out_pkt.o_r_data + 1;
    `uvm_info("ERROR_INJECTION",
        $sformatf("Pixel %0d: R channel +1 (R=%0d + %0d)",
            pixel_count, 1b_ro_vif.o_r_data, out_pkt.o_r_data),
        UVM_LOW)
end

UVM_INFO 1b_ro_monitor.sv(85) @ 3050: uvm_test_top.tb.1b_ro_env.1b_ro_agent.1b_ro_monitor [ERROR_INJECTION] Pixel 10: R channel +1 (R=109 110)
UVM_ERROR 1b_ro_sb.sv(311) @ 3050: uvm_test_top.tb.1b_ro_env.1b_ro_sb [1b_ro_sb_c] Pixel 10 MISMATCH - C(R=109 G=24 B=481) DUT(R=110 G=24 B=481)
UVM_ERROR 1b_ro_sb.sv(337) @ 3270: uvm_test_top.tb.1b_ro_env.1b_ro_sb [1b_ro_sb_c] Line 1 Pixel 9 MISMATCH - C: R=109 G=24 B=481 | DUT: R=110 G=24 B=481
UVM_ERROR 1b_ro_sb.sv(349) @ 3270: uvm_test_top.tb.1b_ro_env.1b_ro_sb [1b_ro_sb_c] Line 1 FAILED with 1 pixel errors
UVM_INFO 1b_ro_sb.sv(344) @ 3810: uvm_test_top.tb.1b_ro_env.1b_ro_sb [1b_ro_sb_c] Line 2 MATCH (20 pixels)
UVM_INFO 1b_ro_sb.sv(344) @ 4350: uvm_test_top.tb.1b_ro_env.1b_ro_sb [1b_ro_sb_c] Line 3 MATCH (20 pixels)
UVM_INFO 1b_ro_sb.sv(344) @ 4890: uvm_test_top.tb.1b_ro_env.1b_ro_sb [1b_ro_sb_c] Line 4 MATCH (20 pixels)
UVM_INFO 1b_ro_sb.sv(344) @ 5430: uvm_test_top.tb.1b_ro_env.1b_ro_sb [1b_ro_sb_c] Line 5 MATCH (20 pixels)
UVM_INFO 1b_ro_sb.sv(344) @ 5970: uvm_test_top.tb.1b_ro_env.1b_ro_sb [1b_ro_sb_c] Line 6 MATCH (20 pixels)
UVM_INFO 1b_ro_sb.sv(344) @ 6510: uvm_test_top.tb.1b_ro_env.1b_ro_sb [1b_ro_sb_c] Line 7 MATCH (20 pixels)
UVM_INFO 1b_ro_sb.sv(344) @ 7050: uvm_test_top.tb.1b_ro_env.1b_ro_sb [1b_ro_sb_c] Line 8 MATCH (20 pixels)
UVM_INFO 1b_ro_sb.sv(344) @ 7590: uvm_test_top.tb.1b_ro_env.1b_ro_sb [1b_ro_sb_c] Line 9 MATCH (20 pixels)
UVM_INFO 1b_ro_sb.sv(172) @ 10870: uvm_test_top.tb.1b_ro_env.1b_ro_sb [1b_ro_sb_c] C Model: Frame completed with 9 lines
UVM_INFO 1b_ro_sb.sv(202) @ 10890: uvm_test_top.tb.1b_ro_env.1b_ro_sb [1b_ro_sb_c] DUT: Frame completed with 9 lines
UVM_ERROR 1b_ro_sb.sv(389) @ 10890: uvm_test_top.tb.1b_ro_env.1b_ro_sb [1b_ro_sb_c] Frame 1 Line 0 Pixel 9 MISMATCH - C: R=109 G=24 B=481 | DUT: R=110 G=24 B=481
UVM_ERROR 1b_ro_sb.sv(402) @ 10890: uvm_test_top.tb.1b_ro_env.1b_ro_sb [1b_ro_sb_c] Frame 1 FAILED with 1 errors
```

Scoreboard

✓ Error Handling

✓ 실제 에러 케이스

Video Timing

```
=====
[Video Timing Parameters - Randomized values]
=====

Data Mode      : RANDOM
Bypass         : 1
offset val    : 1023

=====
[Horizontal Timing]
HSW   = 1
HBP   = 3
HACT  = 20
HFP   = 3
H_TOTAL = 27

=====
[Vertical Timing]
VSW   = 3
VBP   = 2
VACT = 9
VFP   = 3
V_TOTAL = 17

=====
[Summary]
Active Area   : 20 x 9 = 180 pixels
Total Frame   : 27 x 17 = 459 pixels
=====
```

Report

```
[PIXEL] Match: 1439, Mismatch: 1
***** PIXEL TEST FAILED *****

=====
===== LINE COMPARISON RESULT =====
=====

MATCHED LINES      = 71
MISMATCHED LINES = 1
PIXEL ERRORS      = 1

*****
LINE TEST FAILED *****

=====
===== FRAME COMPARISON RESULT =====
=====

MATCHED FRAMES     = 7
MISMATCHED FRAMES= 1
LINE ERRORS        = 0
PIXEL ERRORS       = 1

*****
FRAME TEST FAILED *****
```

1개의 픽셀 에러가 3가지 레벨에서 감지됨

Pixel 레벨: 1개 불일치

Line 레벨: 1개 라인 불일치

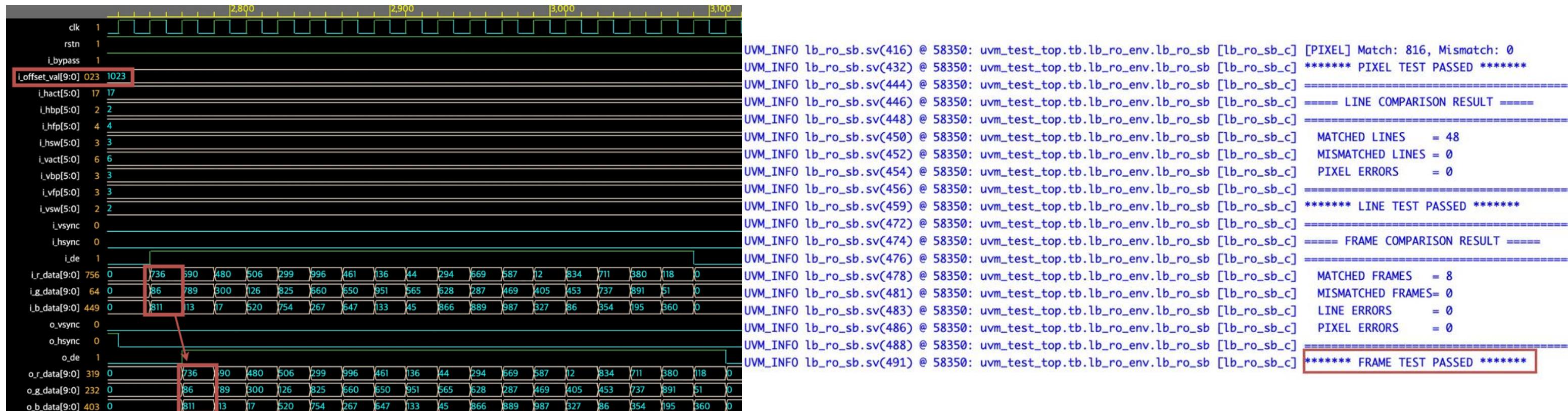
Frame 레벨: 1개 프레임 불일치

⇒ 3-Tier 검증 성공

모든 계층에서 동일한 에러를 독립적으로
검출

Waveform

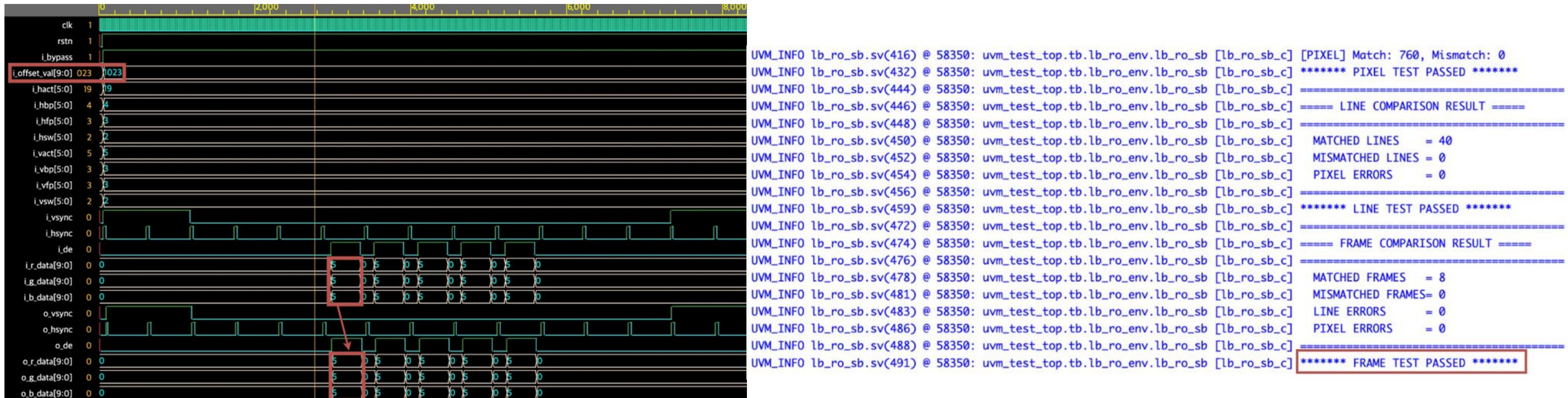
✓ Bypass 시나리오 - RANDOM



- i_bypass = 1
- i_offset_val = 1023
- i_data = RANDOM
- i_offset_val 값과 상관없이 랜덤 생성된 i_RGBdata가 o_RGBdata로 출력

Waveform

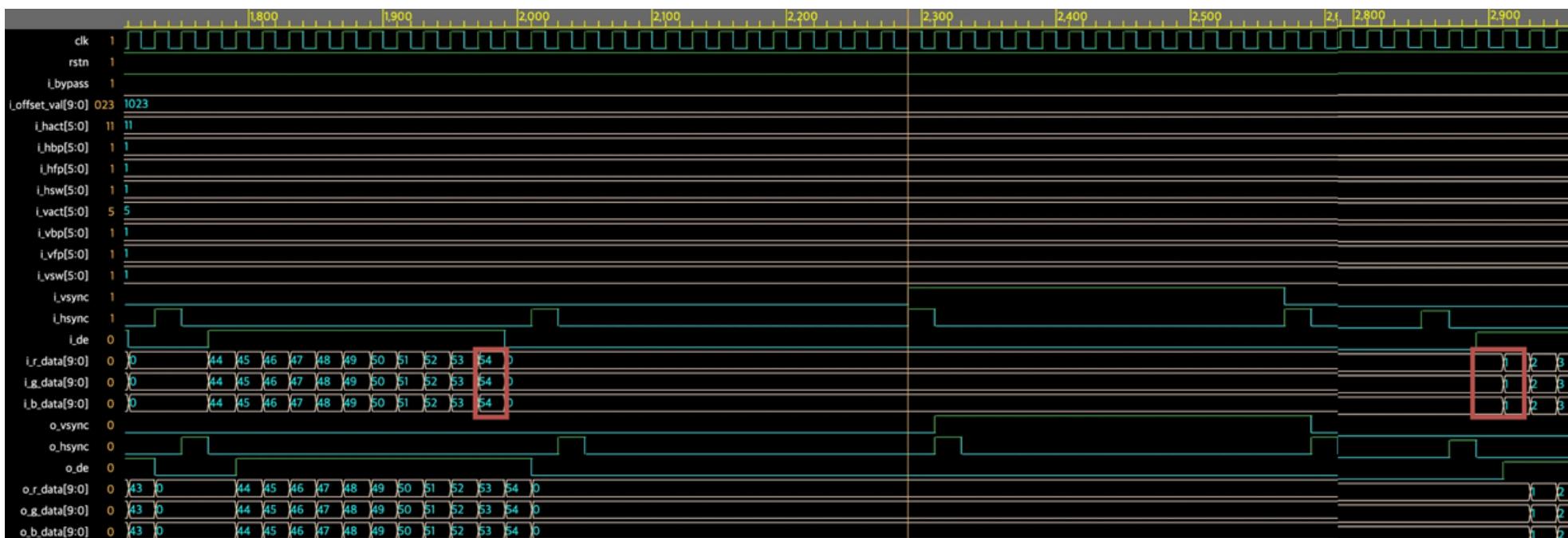
✓ Bypass 시나리오 - FIX



- i_bypass = 1
 - i_offset_val = 1023
 - i_data = FIX
-
- i_offset_val 값과 상관없이 고정된 i_RGBdata가 o_RGBdata로 출력

Waveform

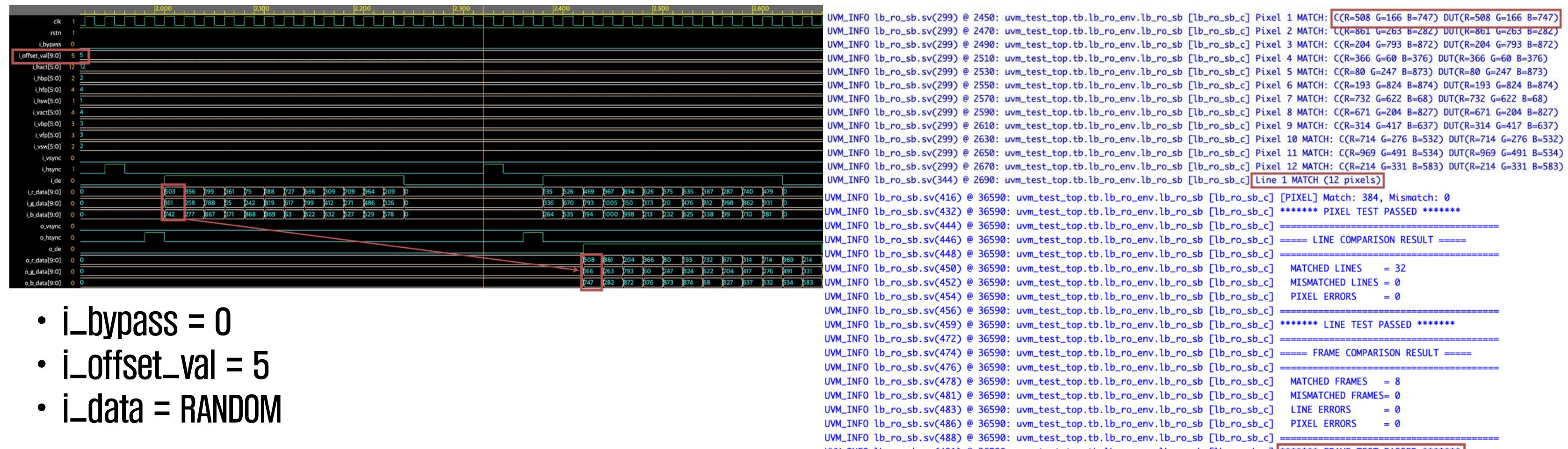
✓ Bypass 시나리오 - INCREASE



- $i_bypass = 1$
- $i_offset_val = 1023$
- $i_RGBdata$ 로 0~1023까지 입력하고
최댓값 이후에는 다시 0으로 순환
- 한 프레임이 끝난 이후에는 다시 0
부터 증가

Waveform

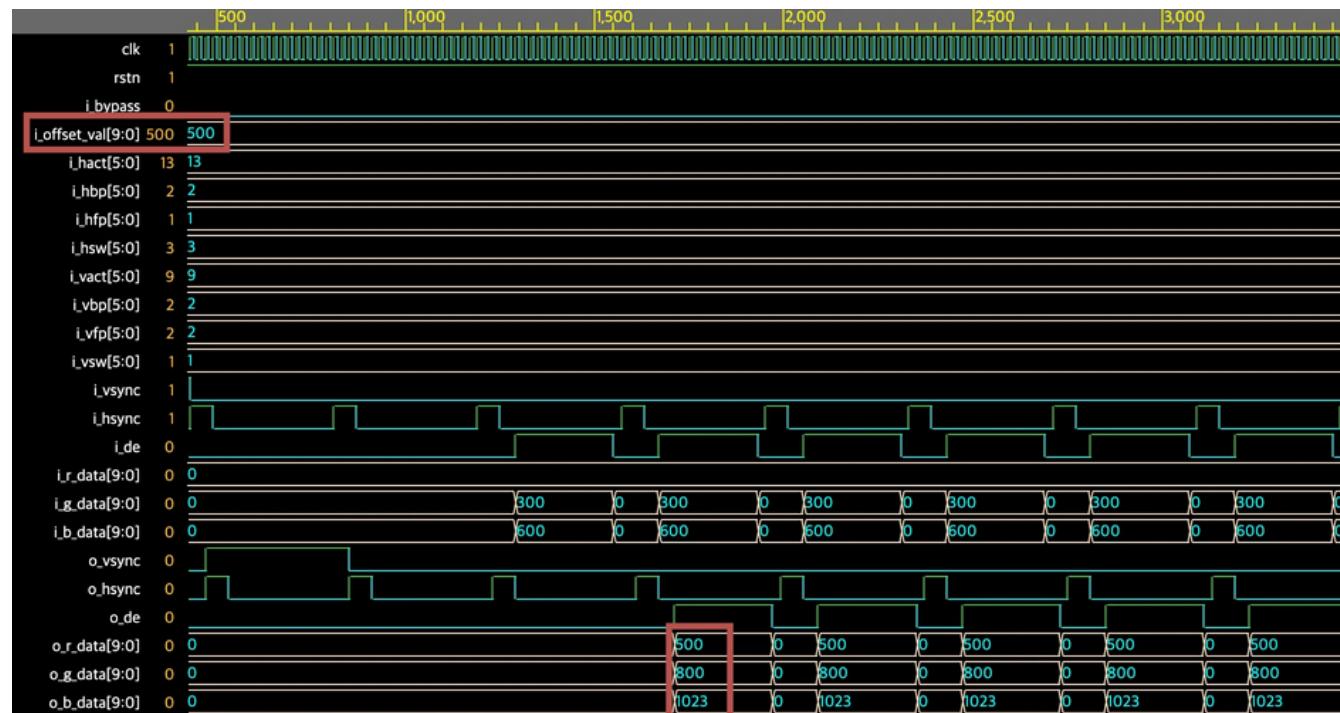
Offset 시나리오 - RANDOM



- i_offset_val 값과 랜덤 생성된 i_RGBdata를 더해서 o_RGBdata 출력

Waveform

Offset 시나리오 - FIX



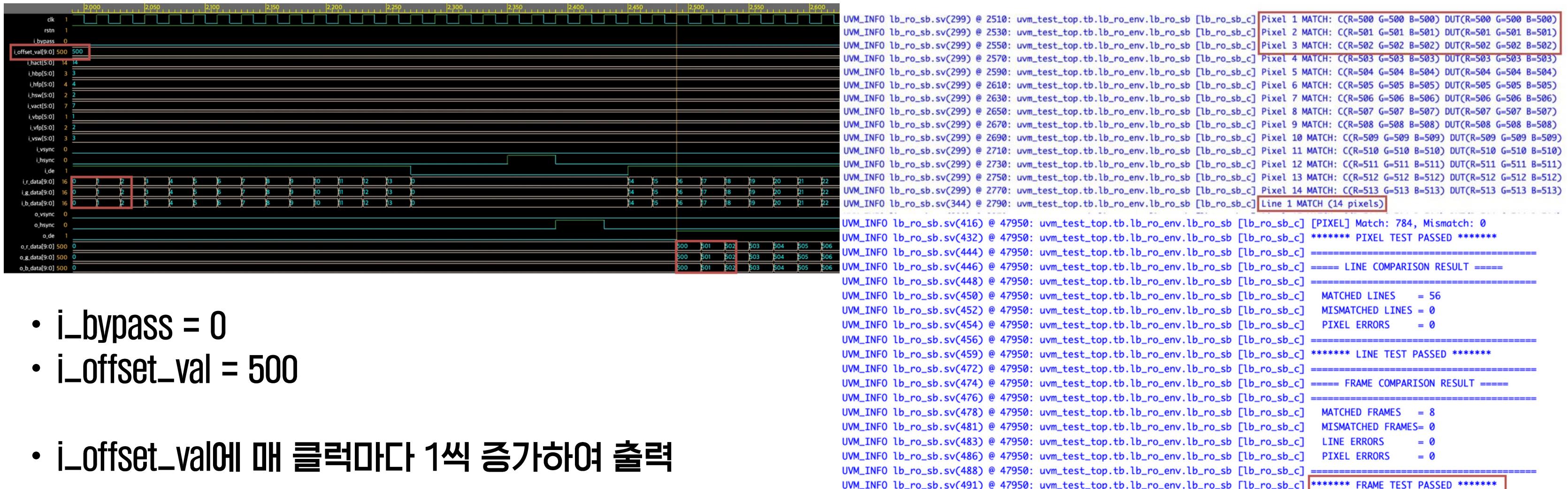
```
UVM_INFO lb_ro_sb.sv(299) @ 1730: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] Pixel 1 MATCH: C(R=500 G=800 B=1023) DUT(R=500 G=800 B=1023)
UVM_INFO lb_ro_sb.sv(299) @ 1750: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] Pixel 2 MATCH: C(R=500 G=800 B=1023) DUT(R=500 G=800 B=1023)
UVM_INFO lb_ro_sb.sv(299) @ 1770: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] Pixel 3 MATCH: C(R=500 G=800 B=1023) DUT(R=500 G=800 B=1023)
UVM_INFO lb_ro_sb.sv(299) @ 1790: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] Pixel 4 MATCH: C(R=500 G=800 B=1023) DUT(R=500 G=800 B=1023)
UVM_INFO lb_ro_sb.sv(299) @ 1810: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] Pixel 5 MATCH: C(R=500 G=800 B=1023) DUT(R=500 G=800 B=1023)
UVM_INFO lb_ro_sb.sv(299) @ 1830: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] Pixel 6 MATCH: C(R=500 G=800 B=1023) DUT(R=500 G=800 B=1023)
UVM_INFO lb_ro_sb.sv(299) @ 1850: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] Pixel 7 MATCH: C(R=500 G=800 B=1023) DUT(R=500 G=800 B=1023)
UVM_INFO lb_ro_sb.sv(299) @ 1870: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] Pixel 8 MATCH: C(R=500 G=800 B=1023) DUT(R=500 G=800 B=1023)
UVM_INFO lb_ro_sb.sv(299) @ 1890: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] Pixel 9 MATCH: C(R=500 G=800 B=1023) DUT(R=500 G=800 B=1023)
UVM_INFO lb_ro_sb.sv(299) @ 1910: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] Pixel 10 MATCH: C(R=500 G=800 B=1023) DUT(R=500 G=800 B=1023)
UVM_INFO lb_ro_sb.sv(299) @ 1930: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] Pixel 11 MATCH: C(R=500 G=800 B=1023) DUT(R=500 G=800 B=1023)
UVM_INFO lb_ro_sb.sv(299) @ 1950: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] Pixel 12 MATCH: C(R=500 G=800 B=1023) DUT(R=500 G=800 B=1023)
UVM_INFO lb_ro_sb.sv(299) @ 1970: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] Pixel 13 MATCH: C(R=500 G=800 B=1023) DUT(R=500 G=800 B=1023)
UVM_INFO lb_ro_sb.sv(344) @ 1990: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] Line 1 MATCH (13 pixels)

UVM_INFO lb_ro_sb.sv(416) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] [PIXEL] Match: 936, Mismatch: 0
UVM_INFO lb_ro_sb.sv(432) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] ***** PIXEL TEST PASSED *****
UVM_INFO lb_ro_sb.sv(444) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] =====
UVM_INFO lb_ro_sb.sv(446) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] ===== LINE COMPARISON RESULT =====
UVM_INFO lb_ro_sb.sv(448) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] =====
UVM_INFO lb_ro_sb.sv(450) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] MATCHED LINES = 72
UVM_INFO lb_ro_sb.sv(452) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] MISMATCHED LINES = 0
UVM_INFO lb_ro_sb.sv(454) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] PIXEL ERRORS = 0
UVM_INFO lb_ro_sb.sv(456) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] =====
UVM_INFO lb_ro_sb.sv(459) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] ***** LINE TEST PASSED *****
UVM_INFO lb_ro_sb.sv(472) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] =====
UVM_INFO lb_ro_sb.sv(474) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] ===== FRAME COMPARISON RESULT =====
UVM_INFO lb_ro_sb.sv(476) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] =====
UVM_INFO lb_ro_sb.sv(478) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] MATCHED FRAMES = 8
UVM_INFO lb_ro_sb.sv(481) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] MISMATCHED FRAMES= 0
UVM_INFO lb_ro_sb.sv(483) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] LINE ERRORS = 0
UVM_INFO lb_ro_sb.sv(486) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] PIXEL ERRORS = 0
UVM_INFO lb_ro_sb.sv(488) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] =====
UVM_INFO lb_ro_sb.sv(491) @ 42670: uvm_test_top.tb.lb_ro_env.lb_ro_sb [lb_ro_sb_c] ***** FRAME TEST PASSED *****
```

- $i_bypass = 0$
- $i_offset_val = 500$
- $fix_rdata = 0$
- $fix_gdata = 300$
- $fix_bdata = 600$
- i_offset_val 값과 fix_data 가 더해져 출력

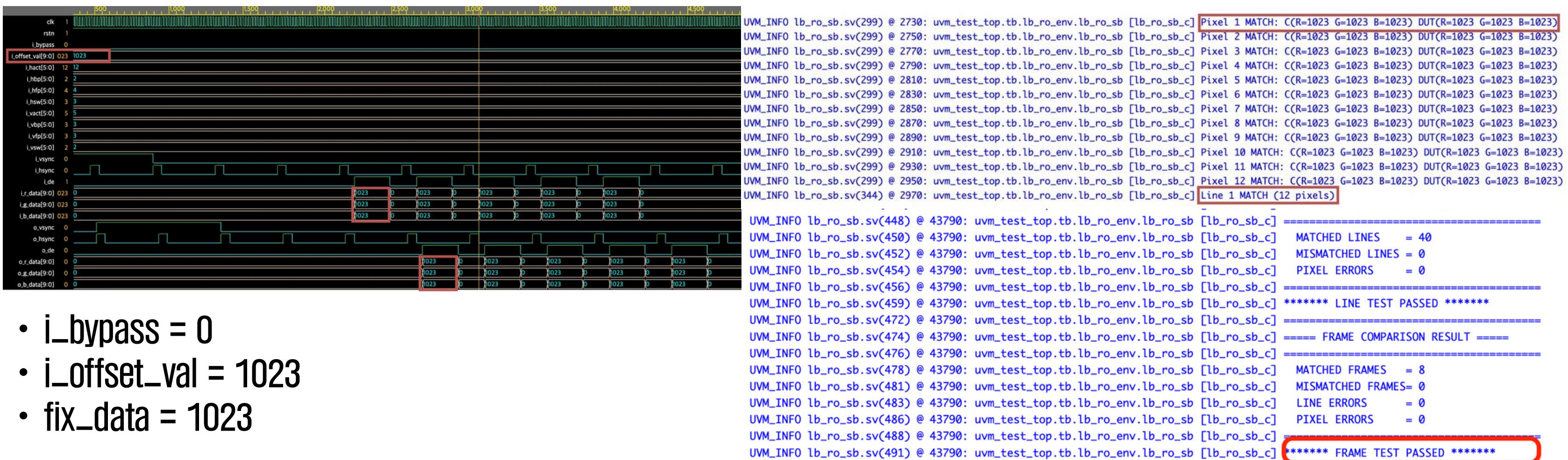
Waveform

Offset 시나리오 - INCREASE



Waveform

Offset 시나리오 - FIX corner case



- i_bypass = 0
- i_offset_val = 1023
- fix_data = 1023
- i_offset_val 값과 fix_data를 모두 최댓값으로 설정하여 출력

06 Conclusion

Trouble Shooting

✓ Edge Detection이 왜 필요한가?

✓ Pixel은 Level, Line/Frame은 Edge?

1. Pixel 비교

a. 개별 데이터 수집: 픽셀은 하나씩 독립적으로 처리하기 때문에 시작/끝 구분이 불필요

2. Line/Frame 비교 (Edge Detection)

a. 그룹 단위 데이터 수집: 여러 픽셀/라인을 모아서 처리하기 때문에 완성 시점을 명확히 알아야 함

✓ Level Check의 문제점

Level Check로 Line/Frame을 처리하면 매 클락마다 같은 라인이 반복돼서 추가됨

→ Queue 오염, 비교 결과 엄망!!

✓ 해결: Edge Detection

De/VSYNC 1 → 0 순간만 1회 실행하여 완성 시점을 명확하게 지정

De falling edge → Line 완성 시점, VSYNC falling edge → Frame 완성 시점

Trouble Shooting

✓ 마지막 프레임 누락 문제

✓ 문제 상황

Scoreboard에서 프레임 단위 비교 시 마지막 프레임이 누락되는 현상 발생

✓ 원인 분석

VSYNC는 프레임 시작 신호이므로, VSYNC falling edge를 이전 프레임의 완성 시점으로 사용

→ 마지막 프레임은 다음 VSYNC가 없어 Queue에 Push되지 않음

→ run_phase 종료 시 current_frame 버퍼에만 남아있음

→ 비교되지 않고 시뮬레이션 종료

Trouble Shooting

✓ 마지막 프레임 누락 문제

✓ 해결 방법

1. 남은 프레임 데이터를 Queue에 추가
 - current_c_frame → c_frame_q.push_back()
2. Queue 소진 및 비교
 - 남은 Line 데이터 모두 비교
 - 남은 Frame 데이터 모두 비교
3. Queue 정합성 체크
 - 모든 Queue가 비워졌는지 확인

⇒ 마지막 프레임을 포함한 모든 데이터 비교 완료

Trouble Shooting

Sequence 계층 간 rand 중복 선언 문제

문제 상황

lb_ro_user_seq_c에서 fix_r_data, fix_g_data, fix_b_data를 rand로 선언했으나, 상위 Sequence에서 직접 값을 할당받기 때문에 rand 선언이 무의미. 코드 의도가 불명확해지는 문제 발생

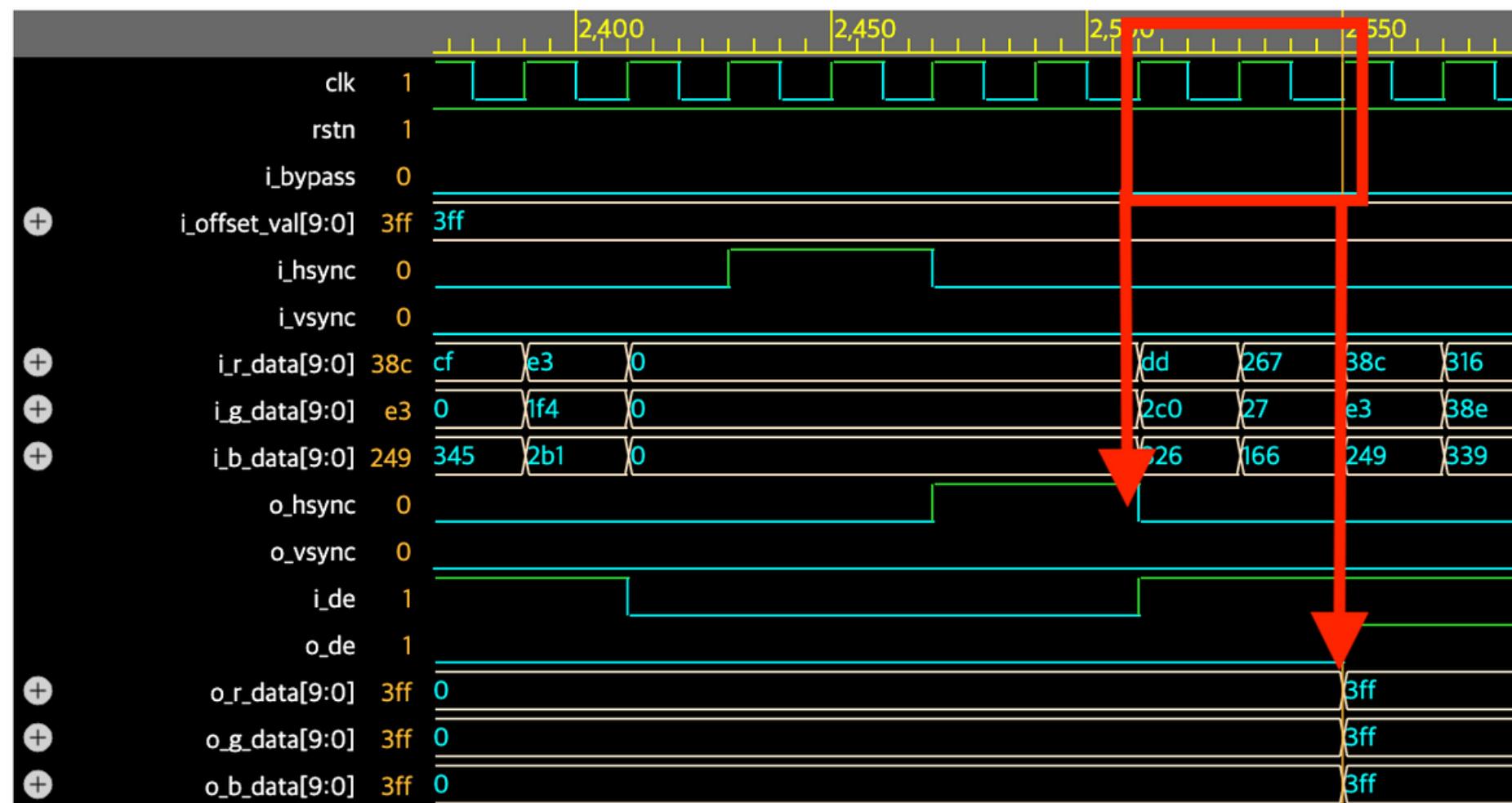
해결 방법

Sequence 계층 구조에서 변수의 randomization 주체가 어느 계층에 있는지 명확히 설계해야 함.

Discussion

✓ Delay Error : 1Line + 2 Clock Delay

문제 상황 : Design Spec에서는 Offset 일 때는 1line + 1clk delay라고 명시되어있지만 실제로는 2 Clock Delay 발생



Discussion

✓ Delay Error : 1Line + 2 Clock Delay

원인 : linebuf_rgboffset_top에서 Output data clock 동기화 및 Line Buffer에서 din clock 동기화 문제

✓ Line Buffer에서 din clock 동기화

```
// ***** din reg *****
always @(posedge clk or negedge rstn) begin
    if (!rstn) r_din <= {DATA_WIDTH{1'b0}};
    else r_din <= {i_red, i_green, i_blue};
end
```

✓ Output data clock 동기화

```
// ===== Output Pipeline =====
always @(posedge clk or negedge rstn) begin
    if (!rstn) begin
        o_vsync <= 1'b0;
        o_hsync <= 1'b0;
        o_de <= 1'b0;
        o_r_data <= {RGB_WIDTH{1'b0}};
        o_g_data <= {RGB_WIDTH{1'b0}};
        o_b_data <= {RGB_WIDTH{1'b0}};
    end else begin
        o_vsync <= w_src_vsync ;
        o_hsync <= w_src_hsync ;
        o_de <= w_src_de ;
        o_r_data <= w_clamp_r ;
        o_g_data <= w_clamp_g ;
        o_b_data <= w_clamp_b ;
    end
end
```

Result

✓ 결론

- 핵심 구현 내용
 - Sequence
 - 3가지 Data Mode (RANDOM/FIX/INCREASE) 구현
 - Constraint를 활용한 시나리오 기반 검증
 - Scoreboard
 - Pixel/Line/Frame 3-Tier 검증 아키텍처
 - C Model 기반 Golden Reference 비교
 - Edge Detection을 통한 정확한 비교 시점 제어

✓ 배운 점

- UVM 검증 환경을 직접 구축하면서 Scoreboard 설계, Queue 관리, Phase 기반 처리 방식 습득
- Agent, Monitor, Scoreboard를 구현하며 각 컴포넌트의 유기적 연결과 Data Flow에 대한 깊은 이해
- 프레임 누락 문제를 해결하는 과정에서 UVM Phase의 실제 활용법과 시뮬레이션 전체 타이밍 중요성에 대한 이해도 향상

감사합니다