Line Buffer 기반 video Down-Scaler 설계 및 검증

[HARMAN 세미콘 아카데미]
황석현

2025.12.07.

프로젝트 개요

- **목표**
  - **DownScaler 구현**: Line Buffer를 응용하여 입력 해상도를 가변적으로 축소(1/N)하는 Scaler 설계.
  - **검증** : Verilog HDL 설계 및 Testbench를 통한 Timing / Functional Verify.

- **개발 환경**
  - Language : SystemVerilog
  - Tool : EDA PlayGround (Cadence Xcelium)

특징 및 기능

- **특징**

1. Ping-Pong Buffering : 실시간 영상 처리를 위해 읽기(Read)와 쓰기(Write)를 교차 수행.
2. Zero Latency Output : SRAM의 Async Read 방식과 Look-ahead 제어를 통해 입력과 동시에 출력되는 Fast Response 구현.
3. Flexible Scaling : parameter인 SCALE_RATIO 변경만으로 1/2, 1/3 등 다양한 비율 지원.

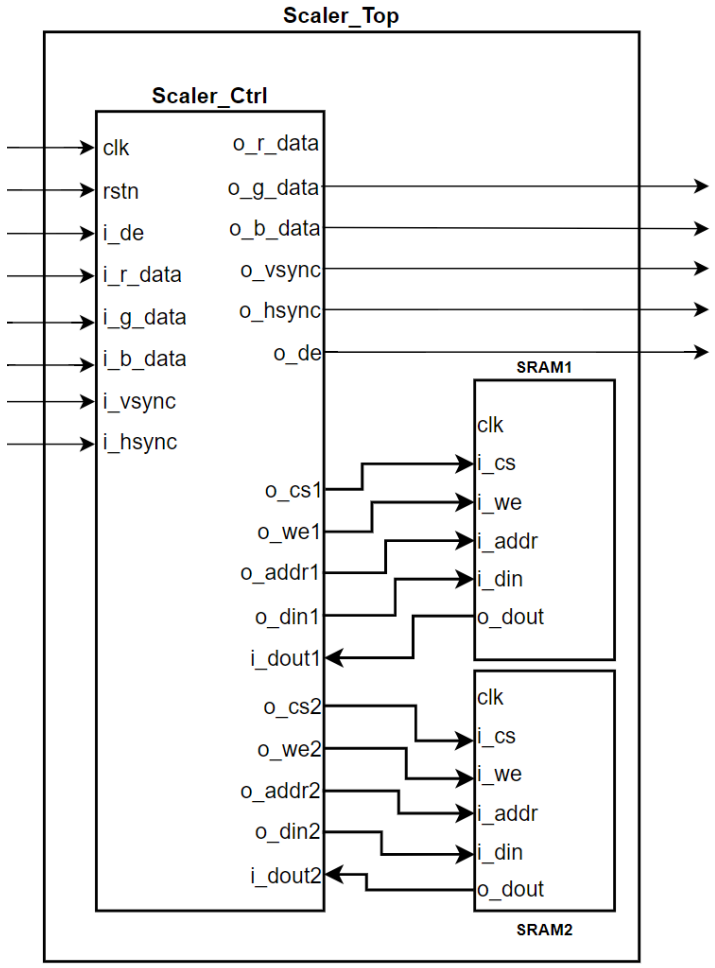
- **기능**

1. SRAM 2개 사용하여 Scaler 설계
2. Scale Function으로 Bypass, 1/2 or 1/3 downscale 구현
   - Sampling, Average, Cross 방식 사용
3. 1/n downscale 처리에서 Input Data가 n으로 나누어 떨어지지 않는 경우, padding하여 Output 출력

Signal	I/O	Bit Width	Description
Clk	I	1	Pixel Clock (1 pixel per 1clk)
Resetn	I	1	Reset (Active low)
i_vsync	I	1	Input Vertical Sync
i_hsync	I	1	Input Vertical Sync
i_den	I	1	Input Data Enable
i_r_data	I	10	Input Data Pixel (R)
i_g_data	I	10	Input Data Pixel (G)
i_b_data	I	10	Input Data Pixel (B)
o_vsync	O	1	Output Vertical Sync
o_hsync	O	1	Output Vertical Sync
o_den	O	1	Output Data Enable
o_r_data	O	10	Output Data Pixel (R)
o_g_data	O	10	Output Data Pixel (G)
o_b_data	O	10	Output Data Pixel (B)

I/O Description

Block Diagram



- **Scaler_Top**

  : 전체 시스템을 통합하는 최상위 module.

- **Scaler_Ctrl**

  : FSM 및 카운터를 포함한 핵심 제어 로직.

  - FSM: 전체 동작 상태 관리 (6-State).
  - Counter : Write(cnt_w)와 Read(cnt_r) 카운터를 분리해 서로 다른 속도로 동작.
  - Read_Addr = cnt_r * RATIO 수식을 통해 Sampling 주소 생성.

- **SRAM 1,2**

  : Line Data 저장을 위한 메모리.

  1 line - 10 pixels

설계 내용

- **Line Buffer Control (Ping-Pong)**

  동작 원리 : 연속된 데이터 흐름을 끊지 않기 위해 두 개의 메모리를 번갈아 사용하여 읽기/쓰기 동시 수행.

  Even Line : SRAM1에 저장(Write) / SRAM2에서 출력(Read)

  Odd Line : SRAM2에 저장(Write) / SRAM1에서 출력(Read)

  → 현재 줄을 저장하는 동안 이전 줄 데이터를 처리하여 1 Line Delay 후 안정적인 데이터 출력.

- **Down-Scaling Algorithm**

  : 전체 pixel 중 일정 간격(SCALE_RATIO)마다 pixel을 선택하여 해상도 축소.

  → Read_Addr = cnt_r x SCALE_RATIO

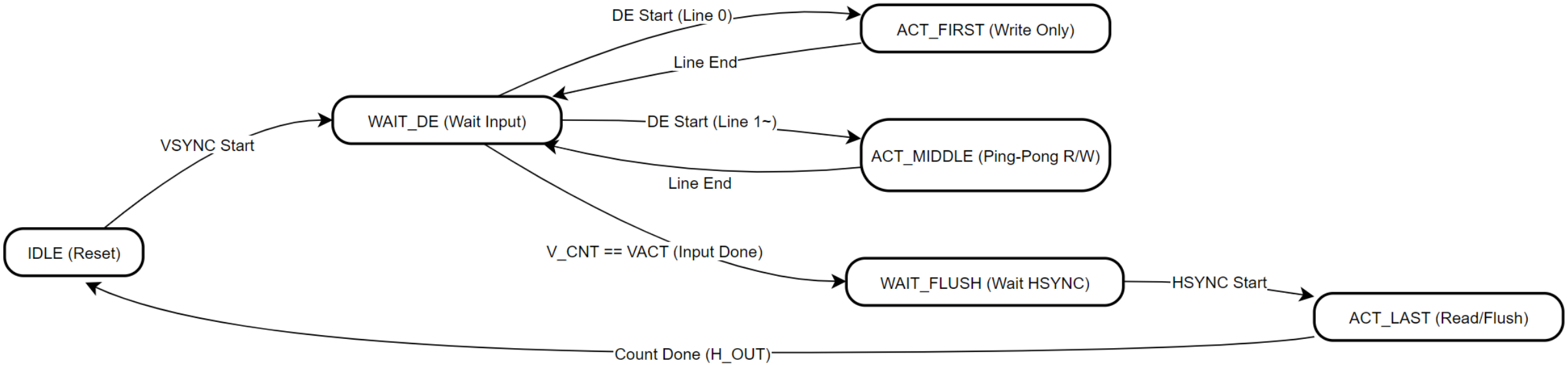
  읽기 카운터(cnt_r)는 0부터 1씩 증가하지만, 실제 SRAM에 요청하는 주소(o_addr)는 비율만큼 곱해져 건너뛰며 접근.

  Write Address : cnt_w (0, 1, 2, 3...) → 모든 입력 pixel 순차 저장.

  Read Address : cnt_r x SCALE_RATIO (0, 2, 4...) → 비율만큼 건너뛰며 읽기.

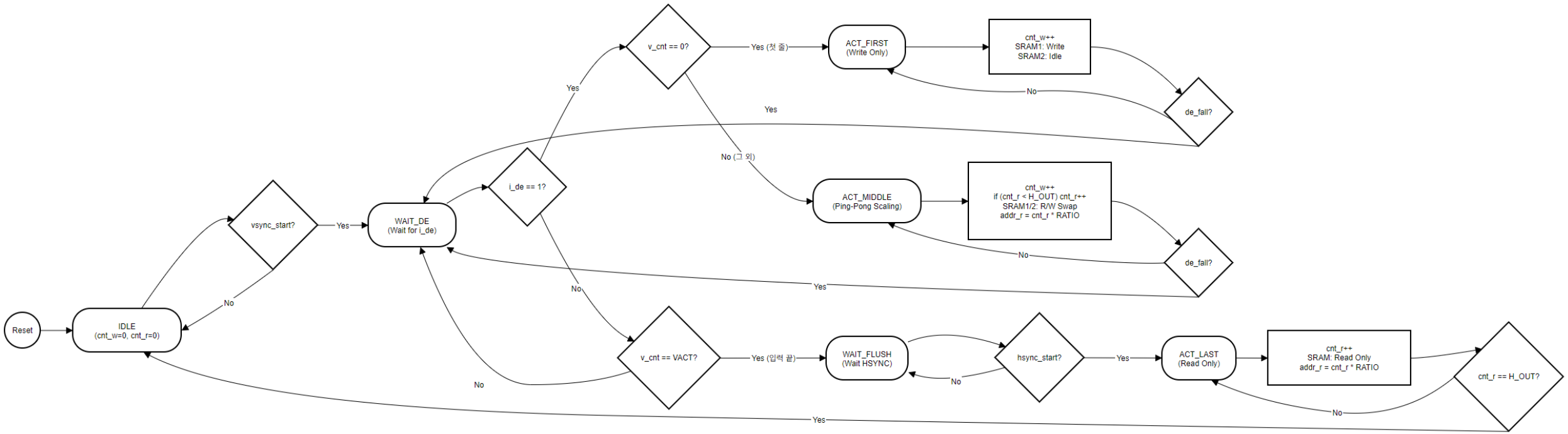
  ex) SCALE_RATIO = 2일 경우, 홀수 번째 pixel을 버리고 짝수 번째만 취함.

FSM 설계



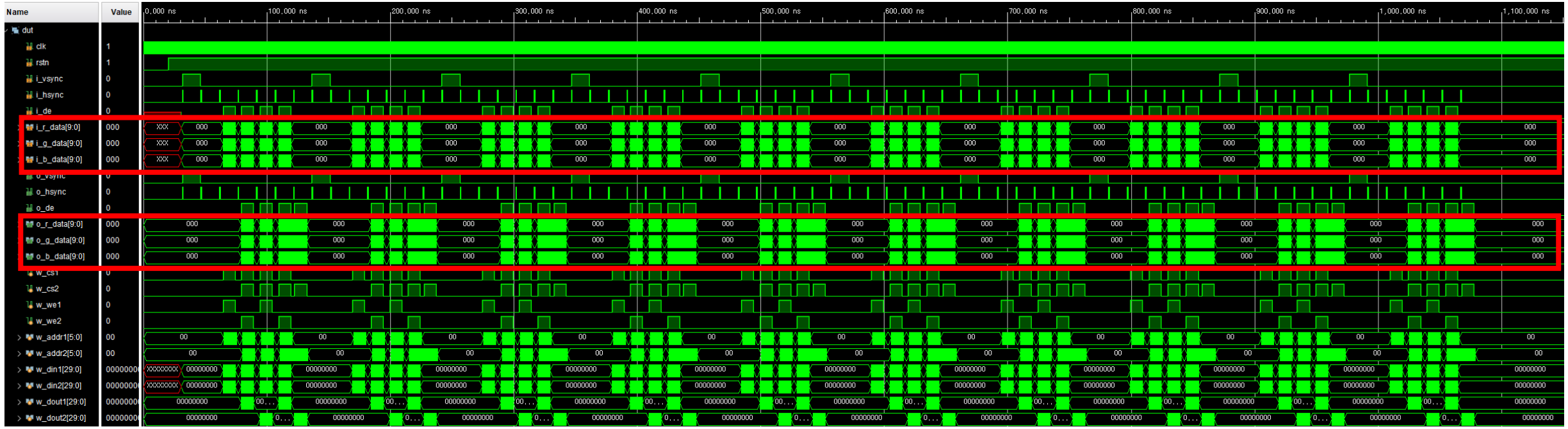
IDLE : 초기화 및 대기 상태.
WAIT_DE : 라인 시작 전 H-Blank 대기.
ACT_FIRST : 첫 번째 라인. 읽을 데이터가 없으므로 Write Only.
ACT_MIDDLE : 중간 라인. Read/Write (Ping-Pong) 동시 수행.
WAIT_FLUSH : 입력 종료 후 마지막 라인 출력 대기.
ACT_LAST : 마지막 라인. 입력이 없어도 Read Only 수행 (Flush).

ASM 설계


Reset
IDLE
(cnt_w=0, cnt_r=0)
vsync_start?
Yes
No
WAIT_DE
(Wait for i_de)
i_de == 1?
Yes
No
v_cnt == 0?
Yes (첫 줄)
No (그 외)
ACT_FIRST
(Write Only)
cnt_w++
SRAM1: Write
SRAM2: Idle
No
de_fall?
Yes
ACT_MIDDLE
(Ping-Pong Scaling)
cnt_w++
if (cnt_r < H_OUT) cnt_r++
SRAM1/2: R/W Swap
addr_r = cnt_r * RATIO
No
de_fall?
Yes
v_cnt == VACT?
Yes (입력 끝)
No
WAIT_FLUSH
(Wait HSYNC)
No
hsync_start?
Yes
ACT_LAST
(Read Only)
cnt_r++
SRAM: Read Only
addr_r = cnt_r * RATIO
No
cnt_r == H_OUT?
Yes

Scaling 파형

Bypass (Ratio = 1)



4줄의 데이터가 총 10번 출력되고 있음.

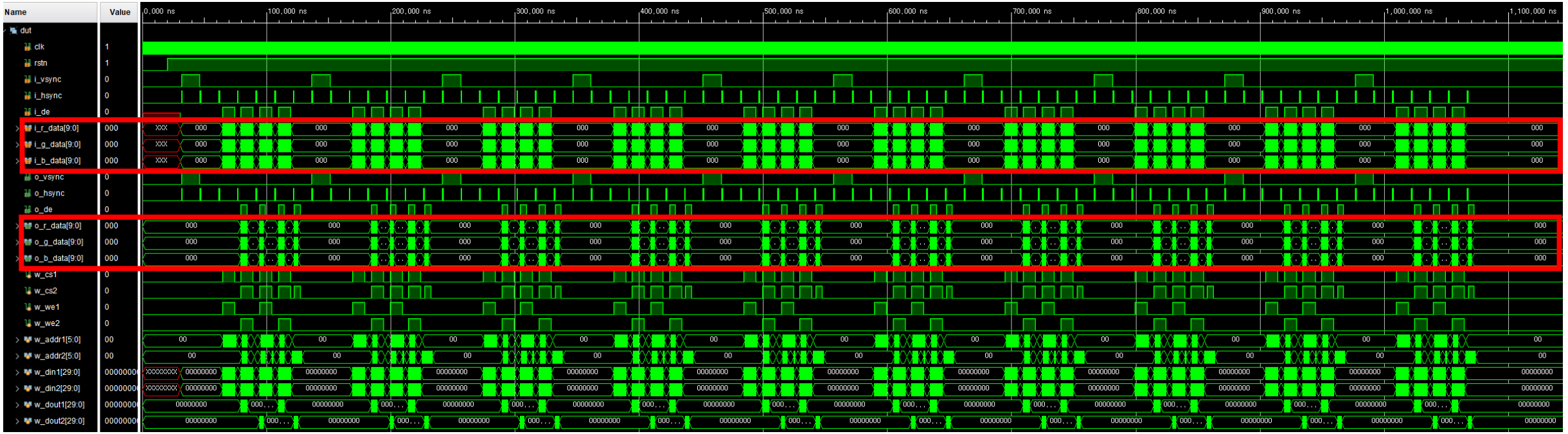
Scaling 파형

Bypass (Ratio = 1)

입력(i_de) 10클락 동안 출력(o_de)도 10클락 동안 유지됨. 입력 데이터가 손실 없이 1:1로 출력됨.

Scaling 파형

1/2 Down Scale (Ratio = 2)



4줄의 데이터가 총 10번 출력되고 있음.

Scaling 파형

1/2 Down Scale (Ratio = 2)

입력(i_de)은 10 clk이지만, 출력(o_de)은 앞쪽 5 clk만 High가 됨.
데이터: 0, 2, 4, 6, 8번 픽셀만 선택되고 1, 3, 5, 7, 9번 픽셀은 버려짐

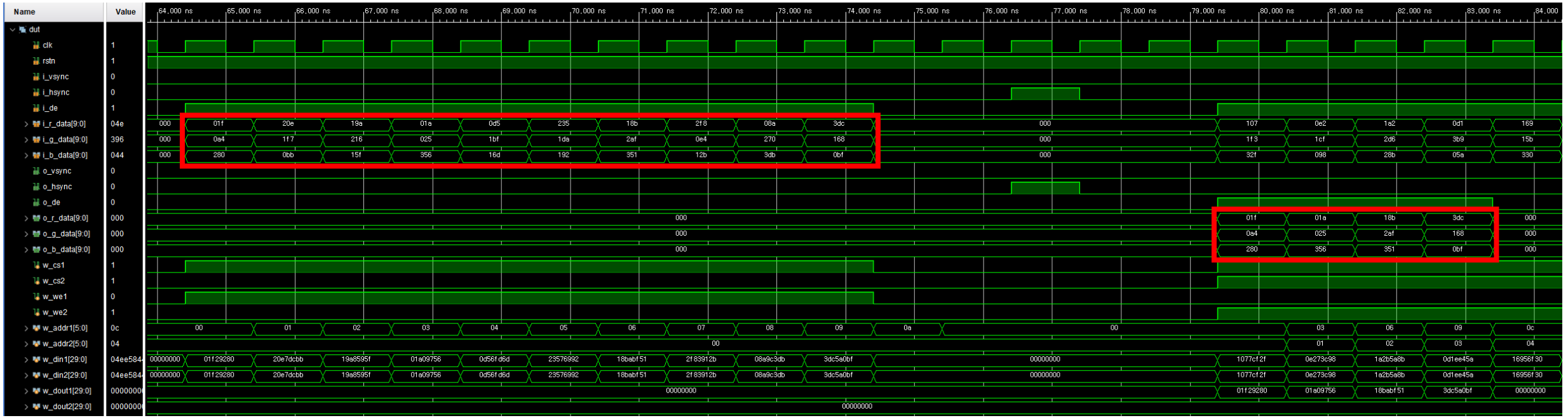
Scaling 파형

1/3 Down Scale (Ratio = 3)

4줄의 데이터가 총 10번 출력되고 있음.

Scaling 파형

1/3 Down Scale (Ratio = 3)

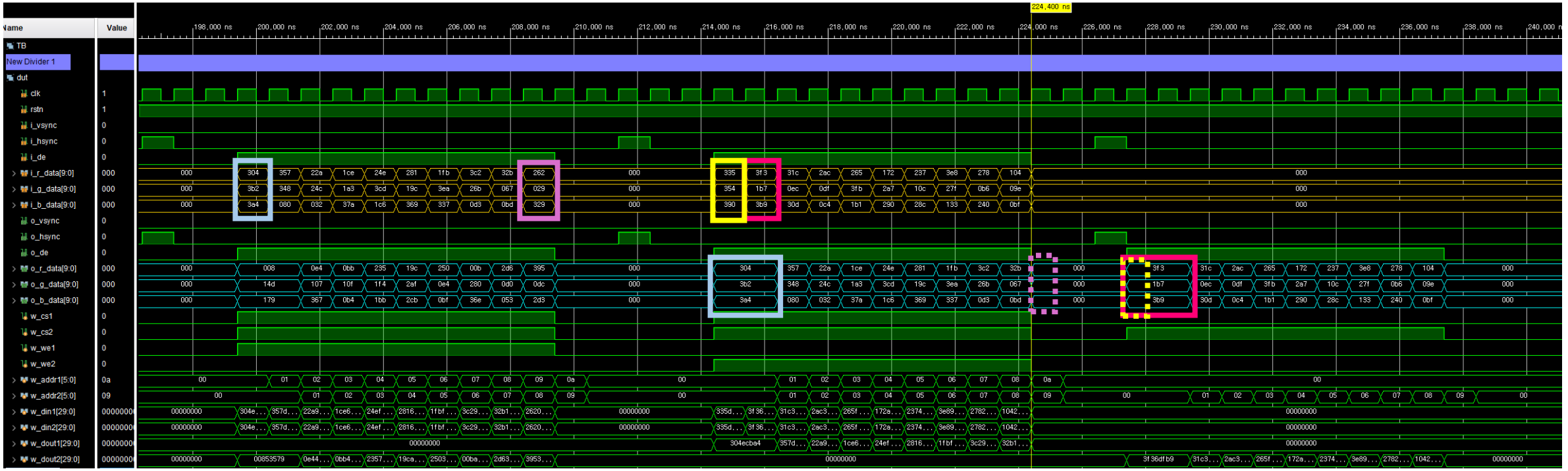


출력(o_de)이 4클락 동안만 High가 됨. (10/3 = 3.33 → 올림 4)
데이터: 0, 3, 6, 9번 픽셀이 출력됨.

Trouble Shooting – 1

문제상황

i_de가 시작되는 첫 clk에 카운터가 동작하지 않아, 첫 번째 픽셀 주소(0번지)가 두 번 반복되는 현상 발생.



Trouble Shooting - 1

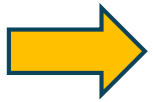
문제원인

카운터가 FSM의 c_state 기준으로 동작함. c_state는 clk edge 후에 업데이트되므로 입력이 들어온
첫 clk 사이클 동안에는 WAIT 상태로 인식하여 카운터가 멈춰 있음.

해결방법

카운트 제어 조건을 c_state 대신 next_state로 변경.
상태가 변하기 전, 입력 신호를 감지하자마자 카운터를 동작시켜 첫 clk 부터 정확한 주소 생성.

```
always_ff @(posedge clk, negedge rstn) begin
    if (!rstn) begin
        cnt_r <= 0;
    end else begin
        // if ((n_state != c_state) && (n_state != ACT_MIDDLE) && (r
        if ((c_state != ACT_MIDDLE) && (c_state != ACT_LAST)) egin
            cnt_r <= 0;
        end else begin  // READ 중.
            if (cnt_r < H_OUT) begin
                if (c_state == ACT_LAST || i_de) begin  // MIDDLE: i
                        cnt_r <= cnt_r + 1;
                    //end
                end
            end
        end
    end
end
```

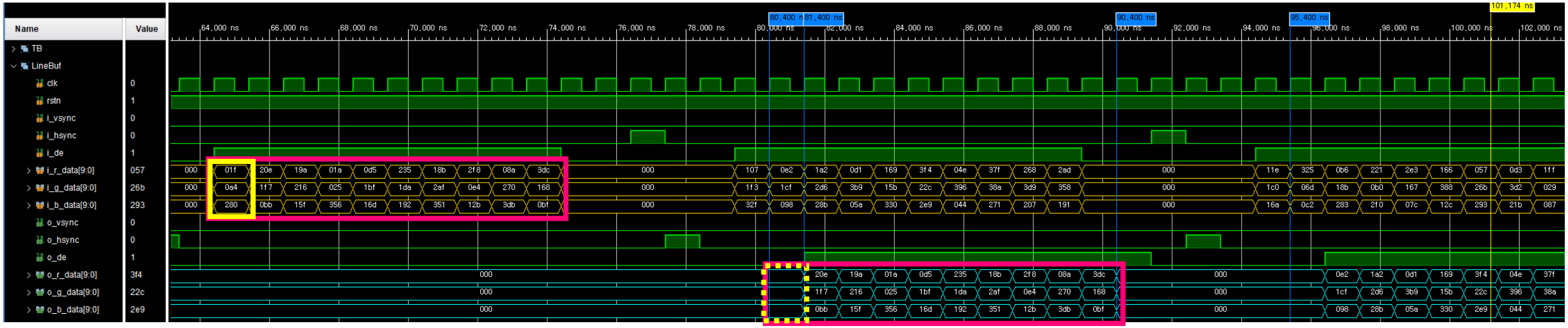


```
always_ff @(posedge clk, negedge rstn) begin
    if (!rstn) begin
        cnt_r <= 0;
    end else begin
        // if ((n_state != c_state) && (n_state != ACT_MIDDLE) && (r
        if ((n_state != ACT_MIDDLE) && (n_state != ACT_LAST)) begin
            cnt_r <= 0;
        end else begin  // READ 중.
            if (cnt_r < H_OUT) begin
                if (n_state == ACT_LAST || i_de) begin  // MIDDLE: i
                        cnt_r <= cnt_r + 1;
                    //end
                end
            end
        end
    end
end
```

Trouble Shooting - 2

문제상황

Line data의 첫번째 값이 출력되지 않고 두번째부터 나오고 있음.



Trouble Shooting - 2

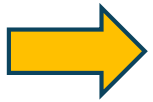
문제원인

초기 설계에서 glitch 방지를 위해 컨트롤러의 출력을 always_ff (flip-flop)를 통해 내보내도록 함.
SRAM이 비동기식으로 데이터를 바로 줘도, 컨트롤러 출력단에서 한 박자 쉬고 내보내기 때문에 지연 발생

해결방법

출력단을 assign (combinational logic)으로 변경.
메모리에서 읽은 데이터를 레지스터 저장 없이 즉시 출력 포트로 연결하여 지연 시간 제거

```
always_ff @(posedge clk) begin
    o_r_data <= w_mux_data;
end
```

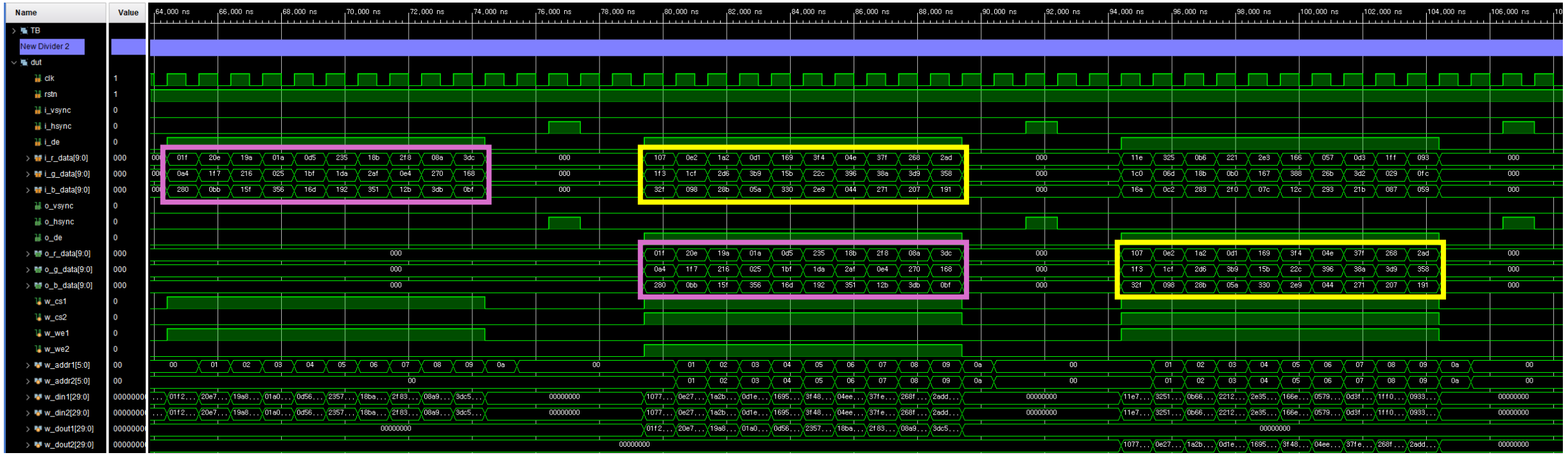


```
assign o_r_data = o_de ? o_mux[29:20] : 10'd0;
```

Trouble Shooting

문제해결

알맞은 타이밍에 데이터 손실 없이 Line Buffer 기능과 Bypass가 작동함.



고찰

기술적 성과

- Line Buffer에서 SRAM의 Read/Write 타이밍을 정밀하게 제어하여 데이터 손실 없는 Ping-Pong 구조를 확립함.
- 기존 동기식 설계의 clk 지연 문제를 발견하고 Async Read SRAM 모델과 조합 출력 로직을 도입하여 해결함.
- SCALE_RATIO 파라미터만 변경하면 회로 수정 없이 다양한 배율(1/2, 1/3, ...)에 대응 가능한 유연한 구조 확보.

설계 시 고려사항

- 고속 영상 처리 시 SRAM의 Access Time과 조합 Logic Delay(계산 시간)가 Critical Path가 될 수 있음을 인지함.
- 라인의 시작과 끝, 그리고 프레임의 마지막 라인(Flush) 등 경계 조건에서의 카운터오동작을 방지하기 위한 예외 처리 로직(Hold, Reset)의 중요성을 확인함.

향후 발전 방향

- 화질 개선 : 인접 픽셀의 평균을 구하는 Average Scaling이나 가중치를 적용하는 Bilinear Scaling 알고리즘 적용 예정.
- 인터페이스 표준화: AXI-Stream 등 표준 인터페이스를 적용하여 타 IP와의 호환성 확보.

감사합니다.

Code

Line Buffer(수정) - https://www.edaplayground.com/x/Arer

Scaler - https://www.edaplayground.com/x/MeEZ