ĐẠI HỌC QUỐC GIA TP.HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA ĐIỆN – ĐIỆN TỬ
BỘ MÔN ĐIỆN TỬ
----------o0o----------

ĐỒ ÁN 2

BÁO CÁO TUẦN ĐỒ ÁN 2

GVHD: TS. Trần Hoàng Quân

Sinh viên thực hiện	MSSV
Nguyễn Văn Đạt	
Bùi Hải Cường	

TP. HỒ CHÍ MINH, THÁNG 11 NĂM 2025

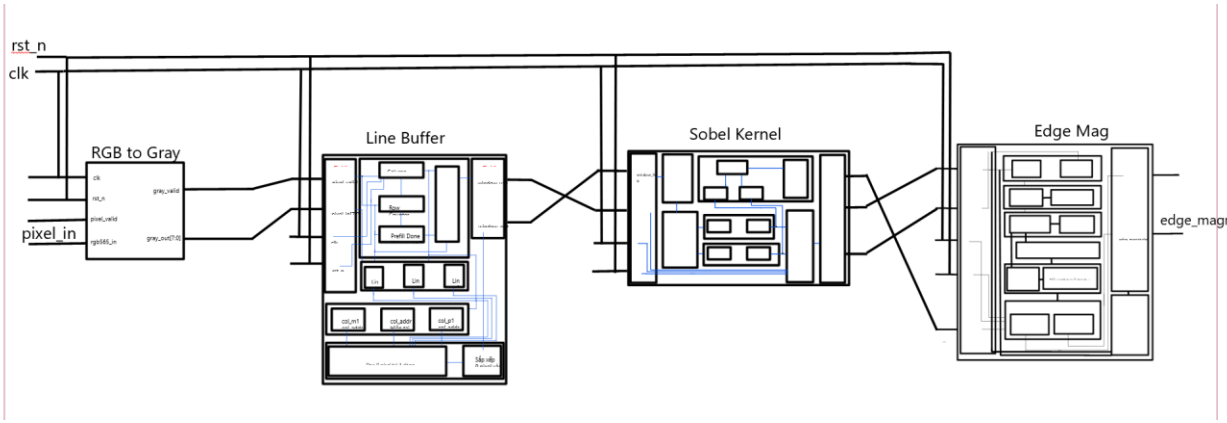
MỤC LỤC

0. NHIỆM VỤ CẦN THỰC HIỆN

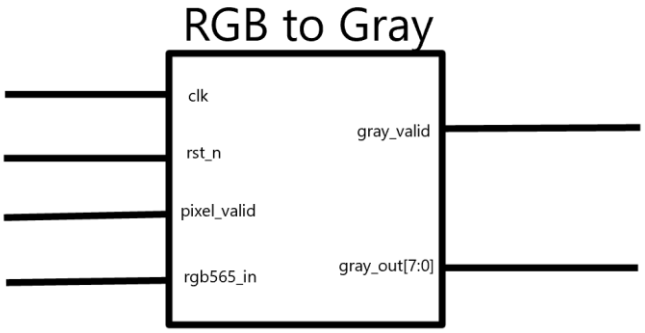
- Đọc lại luận văn tham khảo và code sobel, ma trận 3x3 áp dụng lấy hình ảnh từ camera, chèn sobel vào giữa trong khối HDMI.
- Test waveform modelsim từng khối trong sobel sau đó đưa vào tang nano 4K.
- Vẽ lại sơ đồ khối sobel dùng trong luận văn, tham khảo code verilog trên github và viết lại.
- Vẽ sơ đồ khối HDMI, cách hoạt động để chèn khối sobel vào khoảng phù hợp.

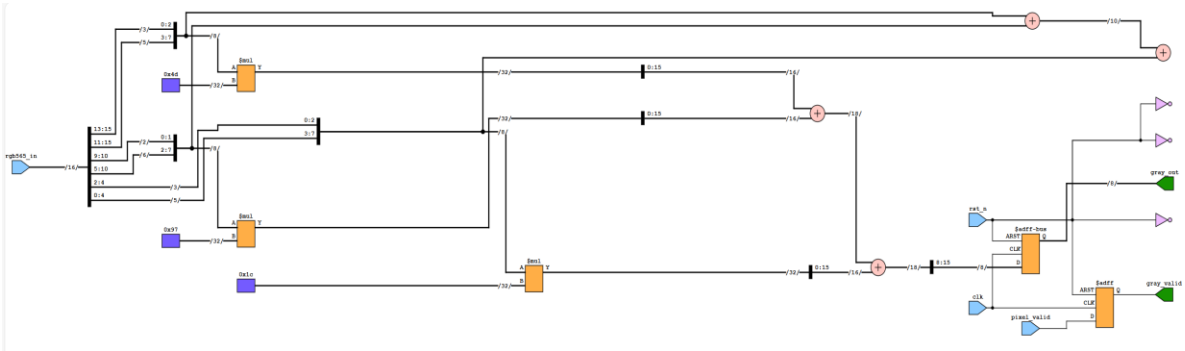
1. KHỐI XỬ LÝ SOBEL

Sơ đồ khối tổng quan (bổ sung ở phụ lục)



1.1 Khối RGB to Gray





Tên tín hiệu	I/O	Độ rộng	Chức năng
clk	Input	1	Clock hệ thống
rst_n	Input	1	Tín hiệu reset (tích cực mức thấp)
pixel_valid	Input	1	Tín hiệu điều khiển báo dữ liệu rgb565_in hợp lệ
rgb565_in	Input	16	Dữ liệu pixel màu đầu vào theo định dạng RGB565
gray_valid	Output	1	Tín hiệu điều khiển báo dữ liệu gray_out hợp lệ

gray_out	Output	8	Kết quả pixel thang độ xám 8-bit

Khối RGB to Gray thực hiện chuyển đổi pixel thời gian thực, nhận đầu vào là dữ liệu RGB565 16-bit và tín hiệu pixel_valid. Quá trình xử lý:

Mở rộng Kênh màu: Các kênh 5-bit Red, 6-bit Green, và 5-bit Blue được tách ra và được mở rộng lên dải 8-bit cho mỗi kênh (R, G, B) bằng phương pháp xấp xỉ phần cứng.

```
wire [7:0] red   = {rgb565_in[15:11], rgb565_in[15:13]};
wire [7:0] green = {rgb565_in[10:5], rgb565_in[10:9]};
wire [7:0] blue  = {rgb565_in[4:0], rgb565_in[4:2]};
```

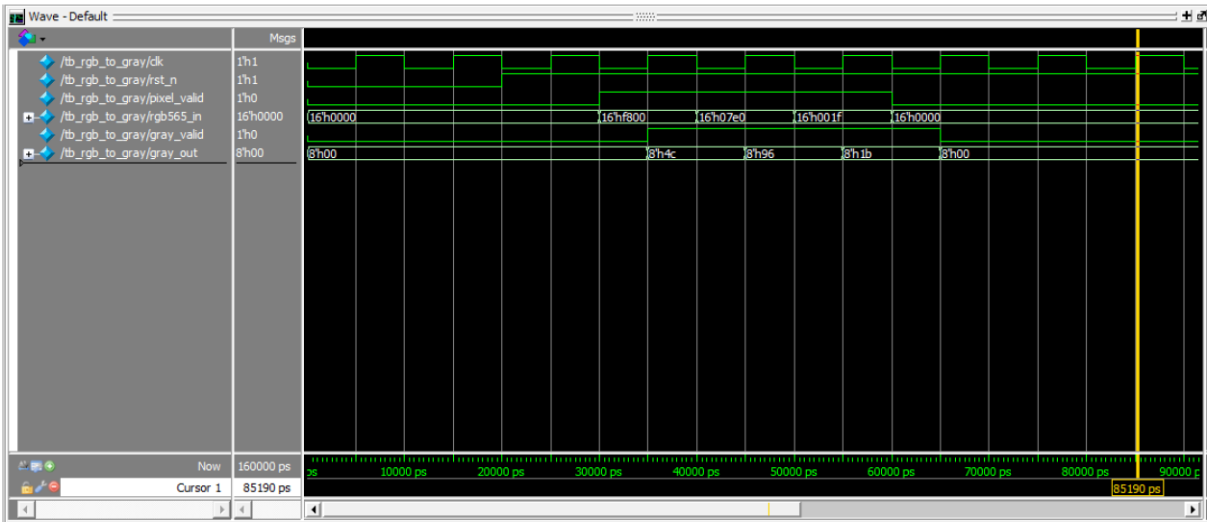
Tính toán Độ sáng: Dữ liệu 8-bit R, G, B được đưa vào logic tổ hợp để tính toán độ sáng, sử dụng công thức trọng số nguyên đã tối ưu hóa: (77*R + 151*G + 28*B). Kết quả tổng được dịch phải 8 bit (>> 8) để chuẩn hóa thành giá trị thang độ xám 8 bit.

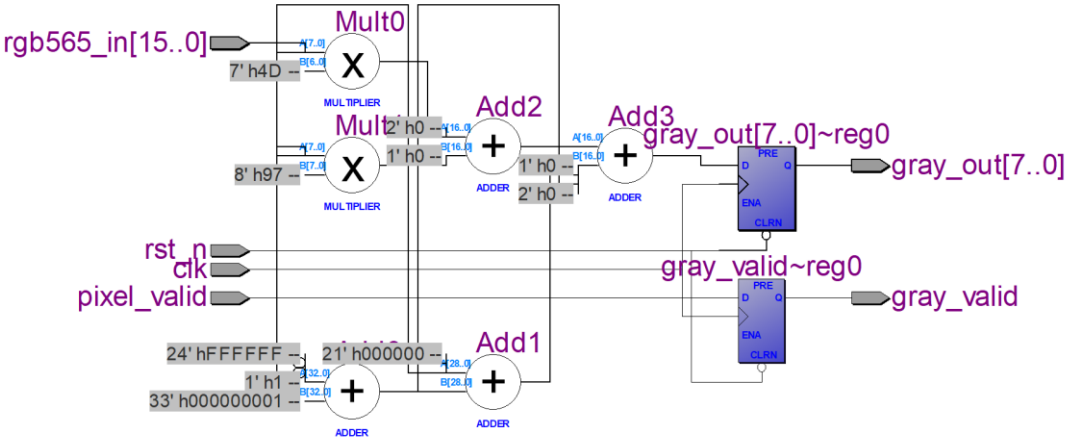
```
wire [15:0] red_weighted   = red * 77;
wire [15:0] green_weighted = green * 151;
wire [15:0] blue_weighted  = blue * 28;
wire [17:0] weighted_sum   = red_weighted + green_weighted + blue_weighted;
wire [7:0]  gray_result    = weighted_sum[15:8];
```

Đầu ra: Một khối always đóng vai trò là một thanh ghi chốt lại kết quả thang độ xám 8 bit và tín hiệu gray_valid.

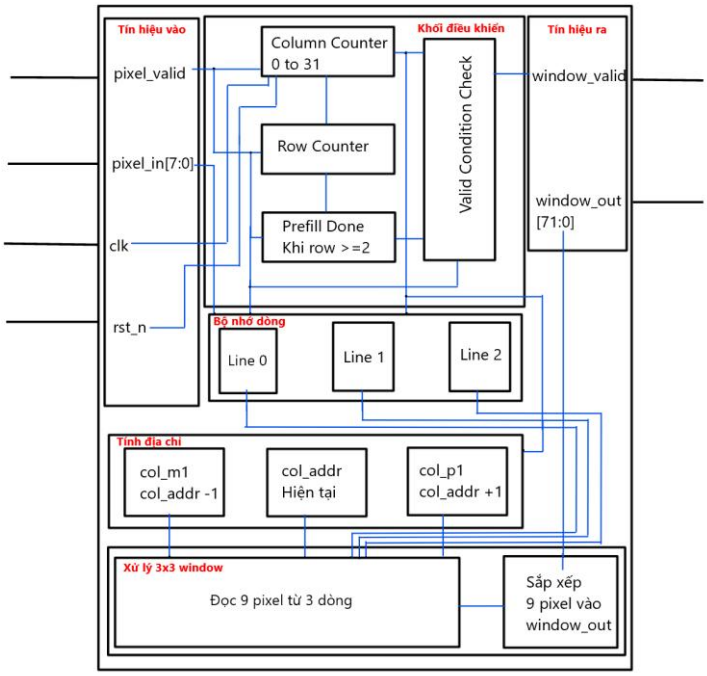
```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        gray_valid <= 1'b0;
        gray_out <= 8'd0;
    end else begin
        gray_valid <= pixel_valid;
        gray_out <= gray_result;
    end
end
```

Kết quả mô phỏng modulesim và RTL:


Wave - Default
Msgs
/tb_rgb_to_gray/clk
/tb_rgb_to_gray/rst_n
/tb_rgb_to_gray/pixel_valid
/tb_rgb_to_gray/rgb565_in
/tb_rgb_to_gray/gray_valid
/tb_rgb_to_gray/gray_out
1'h1
1'h1
1'h0
16'h0000
1'h0
8'h00
16'h0000
16'hf800
16'h07e0
16'h001f
16'h0000
8'h00
8'h4c
8'h96
8'h1b
8'h00
Now
160000 ps
Cursor 1
85190 ps
10000 ps
20000 ps
30000 ps
40000 ps
50000 ps
60000 ps
70000 ps
80000 ps
90000 p
85190 ps


rgb565_in[15..0]
Mult0
7' h4D
MULTIPLIER
Mult1
8' h97
MULTIPLIER
2' h0
1' h0
Add2
ADDER
1' h0
2' h0
Add3
ADDER
gray_out[7..0]~reg0
PRE
D
Q
ENA
CLRN
gray_out[7..0]
rst_n
clk
pixel_valid
gray_valid~reg0
gray_valid
24' hFFFFFF
1' h1
33' h000000001
21' h000000
Add1
ADDER
ADDER

1.2 Khối Line Buffer



Tên tham số	Giá trị mặc định	Chức năng
IMG_WIDTH	32	Độ rộng hình ảnh
PIXEL_WIDTH	8	Độ rộng bit mỗi pixel
ADDR_WIDTH	5	Độ rộng địa chỉ cột

Tên tín hiệu	I/O	Độ rộng	Chức năng
clk	Input	1	Clock hệ thống
rst_n	Input	1	Tín hiệu reset (tích cực mức thấp)
pixel_valid	Input	1	Tín hiệu điều khiển (từ rgb_to_gray) báo pixel_in hợp lệ
pixel_in	Input	PIXEL_WIDTH (8)	Dữ liệu pixel thang độ xám 8 bit đầu vào để ghi vào bộ đệm
window_valid	Output	1	Tín hiệu điều khiển báo window_out chứa một cửa sổ 3x3 hợp lệ
window_out	Output	PIXEL_WIDTH * 9 (72)	Đầu ra chứa toàn bộ 9 pixel (mỗi pixel 8 bit) của cửa sổ 3x3

Mục đích chính của khối line buffer là nhận một luồng pixel (từng pixel một) và biến nó thành một cửa sổ 3x3 pixel. Hoạt động gồm 4 bước chính:

Lưu trữ 3 dòng bằng cách sử dụng ba mảng bộ nhớ line0, line1, line2 mỗi mảng đủ lớn để chứa một dòng ảnh đầy đủ.

```
// Line buffers - distributed RAM
(* ram_style = "distributed" *)
reg [PIXEL_WIDTH-1:0] line0[0:IMG_WIDTH-1];

(* ram_style = "distributed" *)
reg [PIXEL_WIDTH-1:0] line1[0:IMG_WIDTH-1];

(* ram_style = "distributed" *)
reg [PIXEL_WIDTH-1:0] line2[0:IMG_WIDTH-1];
```

Dịch chuyển dòng: khi một pixel mới đến thì nó được ghi vào line0, cùng lúc đó pixel cũ ở line0 được đẩy sang line1, và ở line1 sang line2.

```
// Write to line buffers (shift register behavior)
   always @(posedge clk) begin
      if (pixel_valid) begin
         line0[col_addr] <= pixel_in;
         line1[col_addr] <= line0[col_addr];
         line2[col_addr] <= line1[col_addr];
      end
   end
```

Tạo cửa sổ 3x3 tại mỗi vị trí cột, module đọc đồng thời 3 pixel lân cận từ cả 3 bộ đệm là line0, line1, line2.

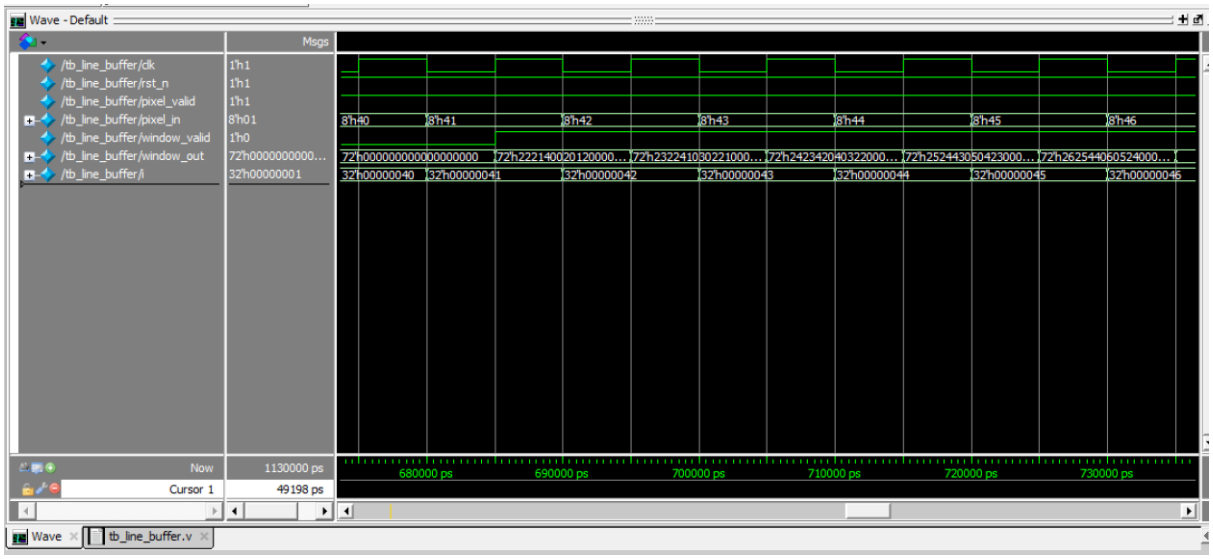
```
// Calculate read addresses for 3x3 window
   wire [ADDR_WIDTH-1:0] col_m1 = (col_addr == 0) ? IMG_WIDTH - 1 :
col_addr - 1;
   wire [ADDR_WIDTH-1:0] col_p1 = (col_addr == IMG_WIDTH - 1) ? 0 : col_addr
+ 1;
```

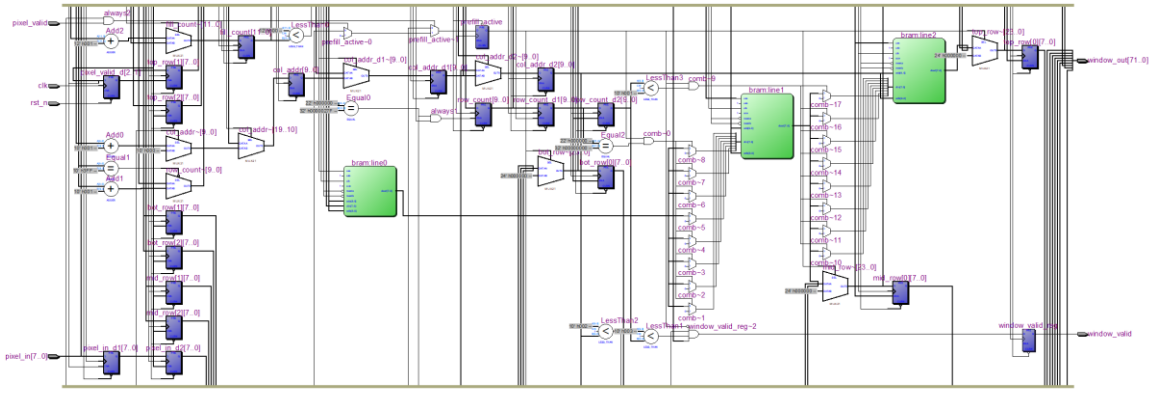
Đóng gói và gửi đi: 9 pixel (8 bit mỗi pixel) được kết hợp lại thành một bus 72 bit (window_out) và gửi đến khối xử lý kế tiếp.

```
always @(posedge clk or negedge rst_n) begin
      if (!rst_n) begin
         window_valid <= 0;
         window_out <= 0;
      end else begin
         window_valid <= valid_condition;
         if (valid_condition) begin
            // Pack in correct order: [p0, p1, p2, p3, p4, p5, p6, p7, p8]
```

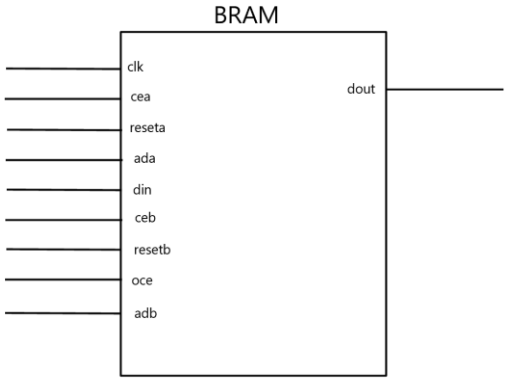
```
            window_out[PIXEL_WIDTH*1-1:PIXEL_WIDTH*0] <= line2[col_m1];
// top-left
            window_out[PIXEL_WIDTH*2-1:PIXEL_WIDTH*1] <= line2[col_addr];
// top-center
            window_out[PIXEL_WIDTH*3-1:PIXEL_WIDTH*2] <= line2[col_p1];   //
top-right
            window_out[PIXEL_WIDTH*4-1:PIXEL_WIDTH*3] <= line1[col_m1];
// mid-left
            window_out[PIXEL_WIDTH*5-1:PIXEL_WIDTH*4] <= line1[col_addr];
// mid-center
            window_out[PIXEL_WIDTH*6-1:PIXEL_WIDTH*5] <= line1[col_p1];   //
mid-right
            window_out[PIXEL_WIDTH*7-1:PIXEL_WIDTH*6] <= line0[col_m1];
// bot-left
            window_out[PIXEL_WIDTH*8-1:PIXEL_WIDTH*7] <= line0[col_addr];
// bot-center
            window_out[PIXEL_WIDTH*9-1:PIXEL_WIDTH*8] <= line0[col_p1];   //
bot-right
          end
       end
   end
```

Kết quả mô phỏng modulesim và RTL:





*Khối BRAM: là Block RAM dùng để tiết kiệm tài nguyên cho các thuật toán bằng cách dùng BRAM thay cho RAM từ LUT. Để thực hiện thuật toán sobel (cửa sổ 3x3) cần đồng thời 3 dòng pixel ảnh. Module line buffer tạo 3 khối bram (line0, line1, line2) mỗi khối như một FIFO lưu trữ một hàng ngang của ảnh từ đó kết hợp tạo thành ma trận 3x3 cho tính toán biên cạnh.



Tên tham số	Giá trị mặc định	Chức năng
ADDR_WIDTH	10	Độ rộng bit của địa chỉ bộ nhớ.
DATA_WIDTH	8	Độ rộng bit của dữ liệu lưu trữ tại mỗi địa chỉ (Mặc định 8 bit cho 1 pixel thang độ xám)

Tên tín hiệu	I/O	Độ rộng	Chức năng
clk	Input	1	Tín hiệu xung nhịp đồng bộ hệ thống
cea	Input	1	Tín hiệu cho phép ghi (Write Enable) cho Cổng A
reseta	Input	1	Tín hiệu Reset cho Cổng A (xóa dữ liệu đầu vào)
ada	Input	ADDR_WIDTH	Địa chỉ ghi dữ liệu cho Cổng A
din	Input	DATA_WIDTH	Dữ liệu đầu vào cần ghi vào bộ nhớ
ceb	Input	1	Tín hiệu cho phép đọc (Read Enable) cho Cổng B
resetb	Input	1	Tín hiệu Reset cho Cổng B (xóa thanh ghi đầu ra)
oce	Input	1	Tín hiệu cho phép xuất dữ liệu (Output Enable) sau khi đọc
adb	Input	ADDR_WIDTH	Địa chỉ đọc dữ liệu từ Cổng B
dout	Output	DATA_WIDTH	Dữ liệu đầu ra được đọc từ bộ nhớ

Dùng lệnh tổng hợp để sử dụng BRAM thay cho RAM tạo từ LUT

```
(* ram_style = "block", syn_ramstyle = "block_ram" *)
```

Tạo cổng đôi cho giao tiếp ghi đọc đồng thời, Port A ghi dữ liệu và Port B đọc dữ liệu ra.

Code verilog khối BRAM:

```
module bram #(
   parameter ADDR_WIDTH = 10,
   parameter DATA_WIDTH = 8
)(
   output reg  [DATA_WIDTH-1:0] dout,
   input  wire                  clk,
   input  wire                  cea,
   input  wire                  reseta,
   input  wire [ADDR_WIDTH-1:0]  ada,
   input  wire [DATA_WIDTH-1:0]  din,
   input  wire                  ceb,
   input  wire                  resetb,
```

```
    input  wire                 oce,
    input  wire [ADDR_WIDTH-1:0]  adb
);
    localparam integer DEPTH = 1 << ADDR_WIDTH;

    // Instruct Gowin tools to infer simple dual-port block RAM.
    (* ram_style = "block", syn_ramstyle = "block_ram" *)
    reg [DATA_WIDTH-1:0] mem [0:DEPTH-1];
    reg [DATA_WIDTH-1:0] dout_reg;

    integer idx;

    initial begin
        for (idx = 0; idx < DEPTH; idx = idx + 1) begin
            mem[idx] = {DATA_WIDTH{1'b0}};
        end
    dout_reg = {DATA_WIDTH{1'b0}};
    dout     = {DATA_WIDTH{1'b0}};
    end

    // Port A write path. reseta acts as a synchronous write enable gate.
    always @(posedge clk) begin
        if (!reseta && cea) begin
            mem[ada] <= din;
        end
    end

    // Port B read path follows Gowin SDPB timing (read-first behaviour).
    always @(posedge clk) begin
        if (resetb) begin
            dout_reg <= {DATA_WIDTH{1'b0}};
            dout     <= {DATA_WIDTH{1'b0}};
        end else begin
            if (ceb) begin
                dout_reg <= mem[adb];
            end
            if (oce) begin
                dout <= dout_reg;
            end
        end
    end

endmodule
```

Tuy nhiên nếu sử dụng bram bình thường này vào hệ thống thì sẽ gây ra vấn đề bị mất data, để giải quyết vấn đề này thì ta cần phải sử dụng pipeline để đồng bộ hóa với độ trễ đọc dữ liệu từ bram và tạo ra cửa sổ trượt 3x3 pixels một cách hiệu quả.

Pipeline trong khối line_buffer này sẽ gồm có 3 giai đoạn.

Stage 0 - Input Capture (Giai đoạn Thu Nhận Đầu Vào):

- Nhận tín hiệu pixel_in[7:0] và pixel_valid từ nguồn đầu vào
- Sinh địa chỉ ghi (col_addr) tự động tăng từ 0 đến 639
- Đếm số dòng đã xử lý (row_count)
- Ghi dữ liệu trực tiếp vào BRAM Line0

Stage 1 - BRAM Address Register (Giai đoạn Đăng Ký Địa Chỉ):

- Trì hoãn 1 chu kỳ clock cho các tín hiệu điều khiển
- Lưu trữ: pixel_valid_d1, col_addr_d1, row_count_d1, pixel_in_d1
- Mục đích: Đáp ứng yêu cầu timing của BRAM (registered address)
- Chuẩn bị địa chỉ cho các BRAM Line1 và Line2

Stage 2 - BRAM Output Register (Giai đoạn Xuất Dữ Liệu):

- Trì hoãn thêm 1 chu kỳ clock (tổng cộng 2 cycles từ input)
- Lưu trữ: pixel_valid_d2, col_addr_d2, row_count_d2, pixel_in_d2
- Dữ liệu từ BRAM sẵn sàng: line0_q, line1_q, line2_q
- Tạo cửa sổ 3×3 thông qua shift registers
- Sinh tín hiệu window_valid để báo hiệu cửa sổ hợp lệ

Phân tích stage:

- Stage 0: Input Capture

Giai đoạn Input Capture là điểm khởi đầu của pipeline, chịu trách nhiệm thu nhận pixel đầu vào và tạo các tín hiệu điều khiển cần thiết. Giai đoạn này hoạt động mà không có bất kỳ độ trễ nào, xử lý dữ liệu ngay lập tức trong chu kỳ clock hiện tại.

Nhiệm vụ chính của Stage 0:

- **Thu nhận dữ liệu:** Nhận pixel_in[7:0] và tín hiệu pixel_valid từ module nguồn (ví dụ: camera interface hoặc image buffer)
- **Tạo địa chỉ cột:** Sử dụng counter col_addr tự động tăng từ 0 đến 639, sau đó wrap về 0
- **Đếm dòng:** Tăng row_count mỗi khi hoàn thành một dòng (khi col_addr = 639)
- **Ghi BRAM:** Ghi dữ liệu pixel_in trực tiếp vào BRAM Line0 tại địa chỉ col_addr

Logic tạo địa chỉ được thiết kế để tự động quản lý vị trí ghi dữ liệu trong BRAM một cách hiệu quả. Thay vì sử dụng 3 bộ đếm riêng biệt cho mỗi BRAM, thiết kế này chỉ sử dụng một bộ đếm duy nhất (col_addr), từ đó tiết kiệm đáng kể tài nguyên phần cứng.

```
col_addr = (col_addr == IMG_WIDTH - 1) ? 0 : col_addr + 1
```

Mô tả hoạt động:

- Khi pixel_valid = 1, col_addr tăng thêm 1
- Khi col_addr đạt 639 (IMG_WIDTH - 1), chu kỳ tiếp theo sẽ reset về 0
- Điều này tạo ra một counter vòng lặp (circular counter) từ 0 → 639 → 0

```
if (col_addr == IMG_WIDTH - 1) {
    row_count = (row_count != MAX) ? row_count + 1 : row_count;
}
```

Đặc điểm của row_count:

- Tăng mỗi khi hoàn thành một dòng (wraparound của col_addr)
- Sử dụng cơ chế saturating counter (không overflow khi đạt MAX)
- Được sử dụng để kiểm tra điều kiện window_valid (cần ≥ 3 dòng)

- Stage 1: BRAM address register

Giai đoạn BRAM Address Register đóng vai trò như một buffer trung gian, đảm bảo rằng tất cả các tín hiệu điều khiển được đồng bộ hóa chính xác với timing requirements của BRAM. Đây là yêu cầu bắt buộc để đạt được timing closure và hoạt động ổn định ở tần số cao.

Tín hiệu được delay tại Stage 1:

Tín hiệu gốc	Tín hiệu delay	Bit width	Mục đích
pixel_valid	pixel_valid_d1	1 bit	Đồng bộ enable signal
col_addr	col_addr_d1	10 bits	Địa chỉ cho Line1/Line2
row_count	row_count_d1	11 bits	Kiểm tra prefill Line1
pixel_in	pixel_in_d1	8 bits	Dữ liệu cho Line1 input

BRAM trong FPGA yêu cầu địa chỉ đọc/ghi phải được register (đăng ký) để đảm bảo timing. Điều này có nghĩa là địa chỉ phải ổn định trong ít nhất một chu kỳ clock trước khi BRAM có thể bắt đầu thao tác đọc/ghi. Nếu không có giai đoạn này, thiết kế sẽ gặp phải timing violations và hoạt động không ổn định.

- STAGE 2: BRAM output Register

Giai đoạn BRAM Output Register là stage cuối cùng của pipeline, nơi mà dữ liệu từ BRAM đã sẵn sàng và được sử dụng để tạo ra cửa sổ trượt 3×3. Đây là giai đoạn quan trọng nhất vì tại đây, module thực hiện việc kết hợp dữ liệu từ 3 dòng khác nhau để tạo thành 9 pixels đồng thời.

Tổng độ trễ = 2 cycles (Stage 1 + Stage 2)

Tín hiệu được delay tại Stage 2:

- **pixel_valid_d2:** Tín hiệu valid sau 2 cycles delay
- **col_addr_d2:** Địa chỉ cột tương ứng với dữ liệu BRAM output
- **row_count_d2:** Số dòng đã xử lý (để kiểm tra window_valid)
- **pixel_in_d2:** Pixel input ban đầu sau 2 cycles (cho dòng hiện tại)

Tại thời điểm Stage 2, dữ liệu từ cả 3 BRAM đều đã được đọc và sẵn sàng sử dụng. Mỗi BRAM cung cấp một pixel từ một dòng khác nhau trong ảnh, tạo thành cơ sở cho việc xây dựng cửa sổ 3×3.

Tín hiệu BRAM	Nguồn dữ liệu	Vị trí trong ảnh	Sử dụng cho
line2_q[7:0]	BRAM Line2 output	Dòng n-2, cột hiện tại	Top row của window 3×3
line1_q[7:0]	BRAM Line1 output	Dòng n-1, cột hiện tại	Middle row của window 3×3
pixel_in_d2[7:0]	Input delay 2 cycles	Dòng n, cột hiện tại	Bottom row của window 3×3

Lưu ý quan trọng: pixel_in_d2 không đến từ BRAM mà là pixel input ban đầu được delay 2 cycles. Điều này là cần thiết vì:

- Pixel hiện tại đang được ghi vào Line0 tại Stage 0
- Phải đợi 2 cycles để Line0 cascade sang Line1, rồi Line1 cascade sang Line2

- Vì vậy, pixel của dòng hiện tại được lấy trực tiếp từ input delay chain

Thông số kĩ thuật pipeline:

Thông Số	Giá Trị	Mô Tả
IMG_WIDTH	640	Độ rộng ảnh (pixels/dòng)
PIXEL_WIDTH	8	Độ rộng dữ liệu mỗi pixel (bits)
Số giai đoạn Pipeline	3 stages	Input → Address → Output
Độ trễ tổng	2 cycles	Do BRAM latency
Thông lượng	1 window/cycle	Khi window_valid = 1
Startup delay	1281 cycles	Prefill mechanism (2 dòng + 1)

Sơ đồ hoạt động timing pipeline:

Cycle	Input Pixel	Stage 0 (Input Capture)	Stage 1 (Address Register)	Stage 2 (Output Register)	Window Valid
T+0	P(0,0)	Write Line0 col=0, row=0	-	-	✗
T+1	P(0,1)	Write Line0 col=1, row=0	Delay P(0,0) col_d1=0	-	✗
T+2	P(0,2)	Write Line0 col=2, row=0	Delay P(0,1) col_d1=1	Delay P(0,0) col_d2=0 Line0_q ready	✗
T+639	P(0,639)	Write Line0 col=639, row=0 *End of Row 0*	Delay P(0,638) col_d1=638	Delay P(0,637) col_d2=637	✗
T+640	P(1,0)	Write Line0→Line1 col=0, row=1 *Start Row 1*	Delay P(0,639) col_d1=639, row_d1=0	Delay P(0,638) col_d2=638, row_d2=0	✗
T+641	P(1,1)	Write Line0→Line1 col=1, row=1	Write Line1 col_d1=0, row_d1=1	Delay P(0,639) Line0_q, Line1_q ready	✗
T+1279	P(1,639)	Write Line0→Line1 col=639, row=1 *End of Row 1*	Write Line1 col_d1=638, row_d1=1	Write Line1 col_d2=637, row_d2=1	✗
T+1280	P(2,0)	Write Line0→ Line1→Line2 col=0, row=2 *Start Row 2*	Write Line1→Line2 col_d1=639, row_d1=1	Write Line1 col_d2=638, row_d2=1	✗
T+1281	P(2,1)	Cascade write col=1, row=2	Write Line1→Line2 col_d1=0, row_d1=2	Write Line1 col_d2=639, row_d2=1	✗
T+1282	P(2,2)	Cascade write col=2, row=2	Write Line2 col_d1=1, row_d1=2	All 3 BRAMs active col_d2=0, row_d2=2 Need col ≥ 2	✗
T+1283	P(2,3)	Cascade write col=3, row=2	Write Line2 col_d1=2, row_d1=2	All 3 lines ready col_d2=1, row_d2=2 Need col ≥ 2	✗
T+1284	**P(2,4)**	**Cascade write col=4, row=2**	**Write Line2 col_d1=3, row_d1=2**	**✓ col_d2=2, row_d2=2 ✓ 3×3 window formed**	✓

				✓ Prefill done	
T+1285	P(2,5)	Cascade write col=5, row=2	Write Line2 col_d1=4, row_d1=2	✓ Window shifts col_d2=3, row_d2=2	✓

Chú thích bảng:

- **P(row, col):** Pixel tại dòng *row*, cột *col*
- **Cascade write:** Dữ liệu được ghi đồng thời vào Line0→Line1→Line2
- **Màu vàng nhạt:** Kết thúc một dòng (row boundary)
- **Màu xanh nhạt:** Bắt đầu dòng mới
- **Màu xanh lá:** Window valid = 1 (output hợp lệ)
- **T+1284:** Cycle đầu tiên có window_valid = 1 (sau khi đáp ứng đủ điều kiện)

Giải thích quy trình xử lý

- *Cycle T+0 đến T+639: Xử Lý Dòng Đầu Tiên*

Trong giai đoạn này, pipeline đang xử lý 640 pixels của dòng đầu tiên (dòng 0). Tất cả các pixels được ghi vào BRAM Line0. Tại thời điểm này, chưa có dữ liệu nào trong Line1 và Line2, do đó window_valid = 0.

- *Cycle T+640 đến T+1279: Xử Lý Dòng Thứ Hai*

Khi bắt đầu dòng 1, cơ chế cascade writing được kích hoạt. Pixels mới được ghi vào Line0, trong khi dữ liệu từ Line0 (dòng 0) được chuyển sang Line1. Tuy nhiên, vẫn chưa đủ 3 dòng để tạo window 3×3, nên window_valid vẫn = 0.

- *Cycle T+1280: Bắt Đầu Dòng Thứ Ba (Critical Point)*

Đây là thời điểm quan trọng khi pipeline bắt đầu có đủ 3 dòng dữ liệu. Cascade writing giờ đây hoạt động đầy đủ với cả 3 BRAM: Line0 nhận pixel mới, Line1 nhận từ Line0, và Line2 nhận từ Line1.

- *Cycle T+1284: First Valid Window Output*

Đây là chu kỳ đầu tiên mà tất cả các điều kiện để tạo window hợp lệ được đáp ứng:

✓ row_count_d2 = 2 (≥ 3 dòng đã được xử lý: dòng 0, 1, 2)

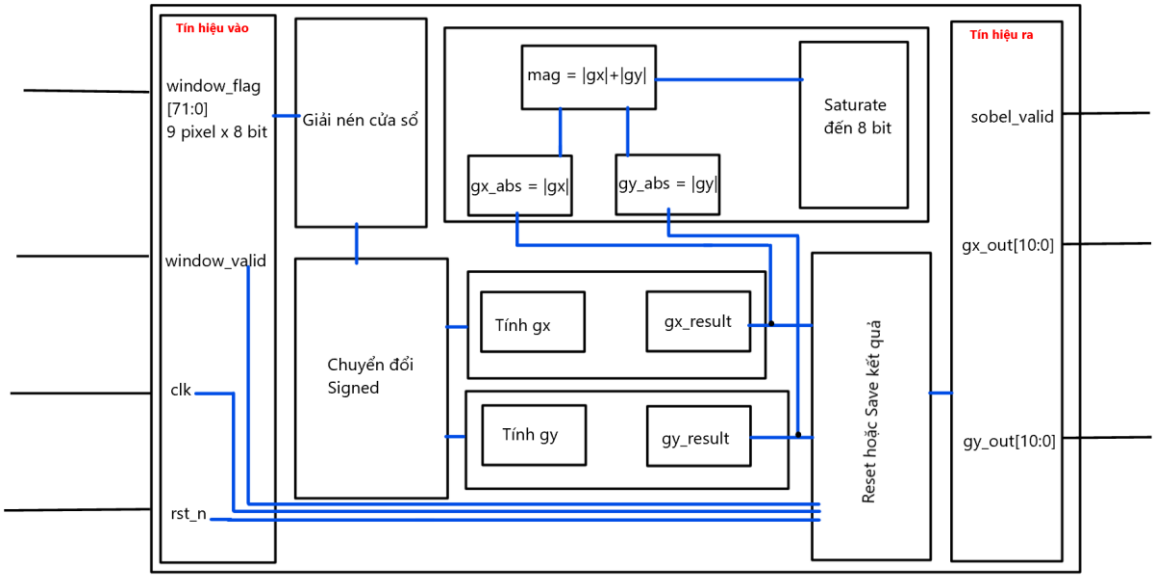
✓ col_addr_d2 = 2 (≥ 3 cột đã được xử lý: cột 0, 1, 2)

✓ prefill_active = 0 (đã hoàn thành 1281 cycles prefill)

✓ pixel_valid_d2 = 1 (dữ liệu hợp lệ)

Từ cycle này trở đi, mỗi chu kỳ clock sẽ tạo ra một cửa sổ 3×3 mới (trừ khi ở biên ảnh hoặc pixel_valid = 0)

1.3 Khối Sobel Kernel



Tên tham số	Giá trị mặc định	Chức năng
SOBEL_WIDTH	11	Độ rộng bit của các giá trị gx và gy đầu ra
PIXEL_WIDTH	8	Độ rộng bit mỗi pixel

Tên tín hiệu	I/O	Độ rộng	Chức năng
clk	Input	1	Clock hệ thống
rst_n	Input	1	Tín hiệu reset (tích cực mức thấp)
window_valid	Input	1	Tín hiệu điều khiển (từ line_buffer) báo window_flat chứa cửa sổ 3x3 hợp lệ
window_flag	Input	PIXEL_WIDTH * 9 (72)	Đầu vào chính: Bus 72 bit chứa 9 pixel 8-bit của cửa sổ 3x3
sobel_valid	Output	1	Tín hiệu điều khiển (gửi đến edge_mag) báo gx_out và gy_out hợp lệ.
gy_out	Output	SOBEL_WIDTH (11)	Kết quả có dấu của phép tích chập với kernel Gy
gx_out	Output	SOBEL	Kết quả có dấu của phép tích chập với kernel Gx

		_WIDTH (11)	

Khối sobel kernel thực hiện tính toán các thuật toán để chuẩn bị cho bước xử lý kế tiếp, các bước chính trong khối:

Giải nén: nhận 72 bit từ khối Line Buffer và giải nén trở lại thành 9 pixel 8 bit riêng biệt từ p0 đến p8.

```
// Unpack flat window
    wire [PIXEL_WIDTH-1:0] p0 = window_flat[7:0];
    wire [PIXEL_WIDTH-1:0] p1 = window_flat[15:8];
    wire [PIXEL_WIDTH-1:0] p2 = window_flat[23:16];
    wire [PIXEL_WIDTH-1:0] p3 = window_flat[31:24];
    wire [PIXEL_WIDTH-1:0] p4 = window_flat[39:32];
    wire [PIXEL_WIDTH-1:0] p5 = window_flat[47:40];
    wire [PIXEL_WIDTH-1:0] p6 = window_flat[55:48];
    wire [PIXEL_WIDTH-1:0] p7 = window_flat[63:56];
    wire [PIXEL_WIDTH-1:0] p8 = window_flat[71:64];
```

P0	**P1**	**P2**
P3	**P4**	**P5**
P6	**P7**	**P8**

Tính toán song song trong cùng một chu kỳ clock bằng logic tổ hợp cả 2 phép tích chập 3x3 cho Gx và Gy.

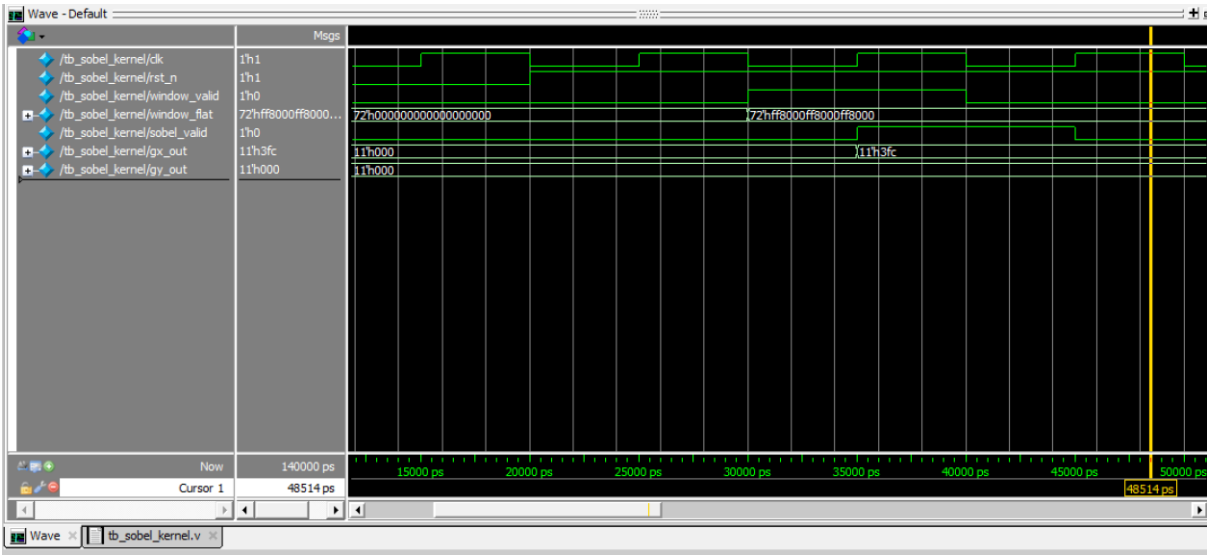
```
// Convert to signed
    wire signed [PIXEL_WIDTH:0] ps0 = {1'b0, p0};
    wire signed [PIXEL_WIDTH:0] ps1 = {1'b0, p1};
    wire signed [PIXEL_WIDTH:0] ps2 = {1'b0, p2};
    wire signed [PIXEL_WIDTH:0] ps3 = {1'b0, p3};
    wire signed [PIXEL_WIDTH:0] ps4 = {1'b0, p4};
    wire signed [PIXEL_WIDTH:0] ps5 = {1'b0, p5};
    wire signed [PIXEL_WIDTH:0] ps6 = {1'b0, p6};
    wire signed [PIXEL_WIDTH:0] ps7 = {1'b0, p7};
    wire signed [PIXEL_WIDTH:0] ps8 = {1'b0, p8};

    // Gx: [-1 0 1; -2 0 2; -1 0 1]
    wire signed [SOBEL_WIDTH-1:0] gx_result =
```

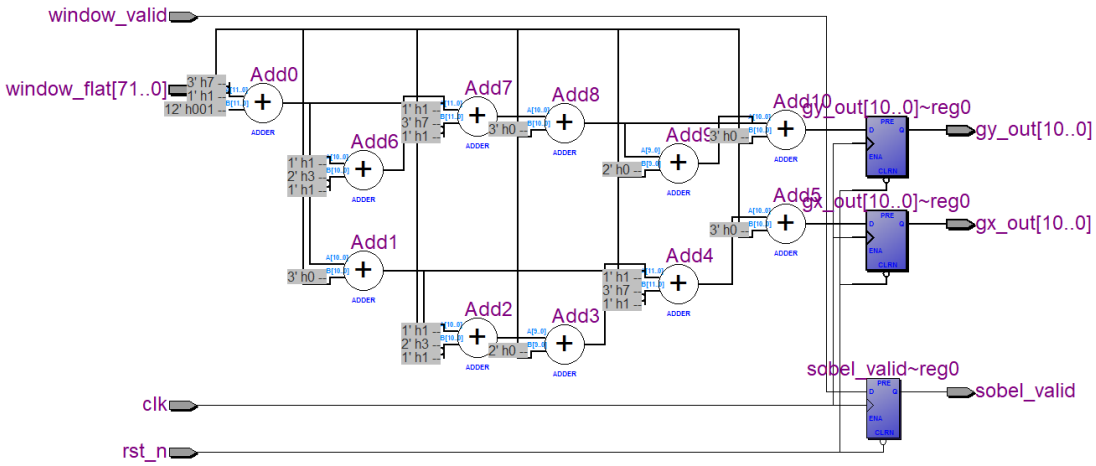
```
        -ps0 + ps2 - (ps3 <<< 1) + (ps5 <<< 1) - ps6 + ps8;

    // Gy: [-1 -2 -1; 0 0 0; 1 2 1]
    wire signed [SOBEL_WIDTH-1:0] gy_result =
        -ps0 - (ps1 <<< 1) - ps2 + ps6 + (ps7 <<< 1) + ps8;
```

Tầng pipeline: sử dụng một khối alway để chốt kết quả gx và gy vào các đầu ra tương ứng.

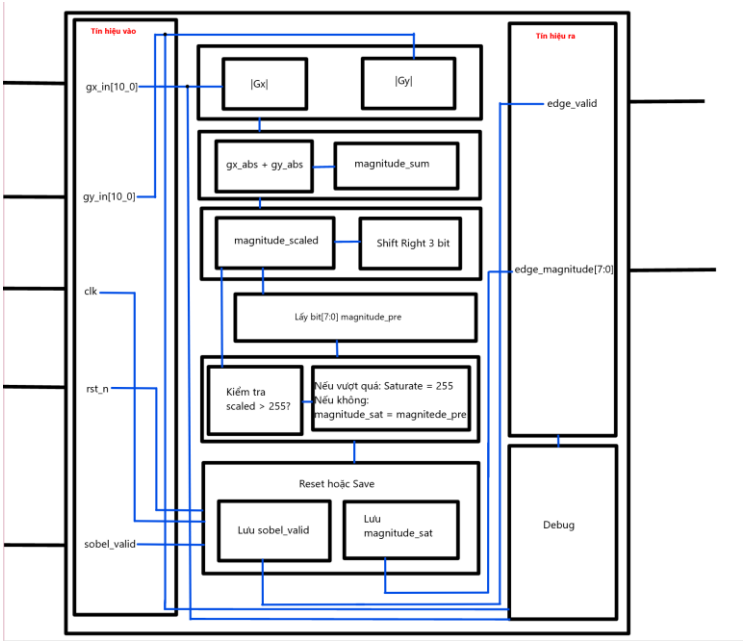
```
always @(posedge clk or negedge rst_n) begin
        if (!rst_n) begin
            sobel_valid <= 1'b0;
            gx_out <= 0;
            gy_out <= 0;
        end else begin
            sobel_valid <= window_valid;
            gx_out <= gx_result;
            gy_out <= gy_result;
        end
    end
```

Kết quả mô phỏng modulesim và RTL:





1.4 Khối Edge Mag



Tên tham số	Giá trị mặc định	Chức năng
SOBEL_WIDTH	11	Độ rộng bit của các tín hiệu gradient gx_in và gy_in (có dấu)
OUTPUT_WIDTH	8	Độ rộng bit của kết quả độ lớn edge_magnitude (không dấu)

Tên tín hiệu	I/O	Độ rộng	Chức năng
clk	Input	1	Clock hệ thống
rst_n	Input	1	Tín hiệu reset (tích cực mức thấp)
sobel_valid	Input	1	Tín hiệu điều khiển (từ sobel_kernel) báo gx_in và gy_in hợp lệ
gx_in	Input	SOBEL_WIDTH (11)	Giá trị gradient Gx có dấu từ khối kernel
gy_in	Input	SOBEL_WIDTH (11)	Giá trị gradient Gy có dấu từ khối kernel.
edge_valid	Output	1	Tín hiệu báo edge_magnitude hợp lệ
edge_magnitude	Output	OUTPUT_WIDTH (8)	Kết quả: Độ lớn của biên cạnh 8 bit

Khối edge mag là khối cuối cùng trong xử lý sobel, khối này tính toán độ lớn của biên cạnh dựa trên công thức xấp xỉ |Gx| + |Gy|, các bước triển khai:

Tính giá trị tuyệt đối bằng logic tổ hợp, nhận gx và gy (11 bit có dấu) và tính giá trị tuyệt đối bằng phép toán bù 2 nếu là số âm.

```
   wire [SOBEL_WIDTH-1:0] gx_abs = gx_in[SOBEL_WIDTH-1] ? (~gx_in + 1'b1) : gx_in;
   wire [SOBEL_WIDTH-1:0] gy_abs = gy_in[SOBEL_WIDTH-1] ? (~gy_in + 1'b1) : gy_in;
```

Tính tổng bằng logic tổ hợp (cộng hai giá trị tuyệt đối đã tính trước đó)

```
wire [SOBEL_WIDTH:0] magnitude_sum = {1'b0, gx_abs} + {1'b0, gy_abs};
```

Chuẩn hóa và bão hòa: Chuẩn hóa giá trị tổng được dịch phải 3 bit, tương đương phép chia cho 8 (Có thể đưa giá trị lớn về dải 8 bit tối đa 255). Bão hòa là kiểm tra kết quả sau khi chuẩn hóa còn vượt qua 255 không và tiếp tục xử lý.

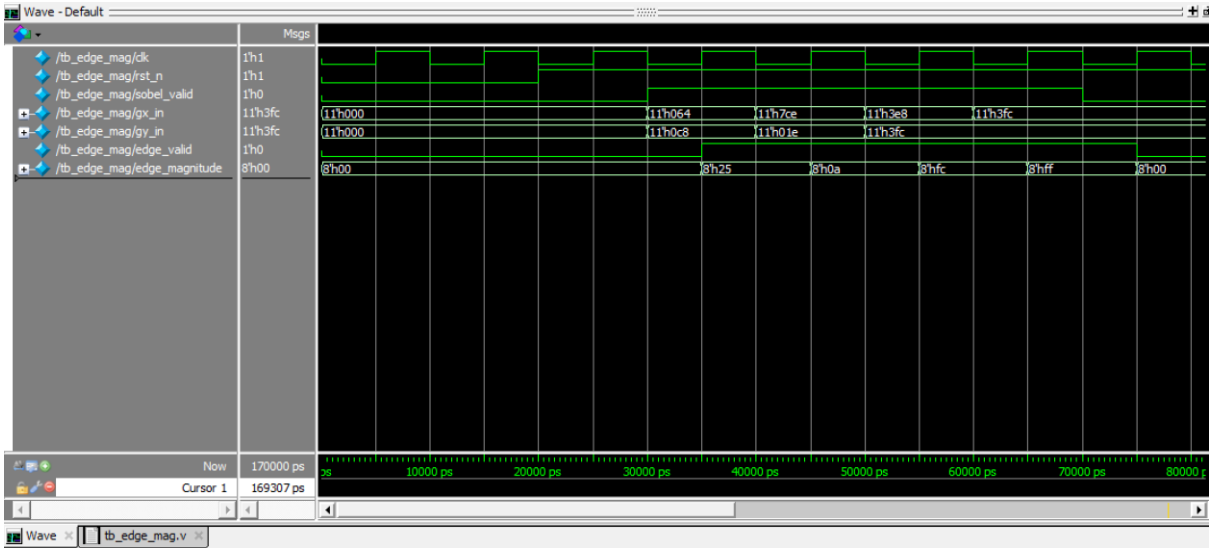
```
wire [SOBEL_WIDTH:0] magnitude_scaled = magnitude_sum >> 3;
   wire [OUTPUT_WIDTH-1:0] magnitude_pre = magnitude_scaled[OUTPUT_WIDTH-1:0];
   wire [OUTPUT_WIDTH-1:0] max_value = {OUTPUT_WIDTH{1'b1}};
   wire [OUTPUT_WIDTH-1:0] magnitude_sat = (magnitude_scaled > max_value) ? max_value : magnitude_pre;
```

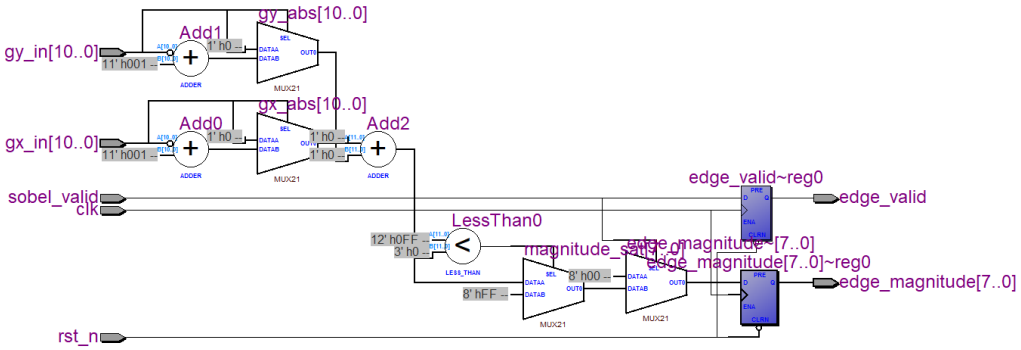
Tầng pipeline cuối sử dụng một khối alway để gửi kết quả magnitude_sat vào đầu ra edge_magnitude.

```
always @(posedge clk or negedge rst_n) begin
      if (!rst_n) begin
         edge_valid <= 1'b0;
         edge_magnitude <= {OUTPUT_WIDTH{1'b0}};
      end else begin
```

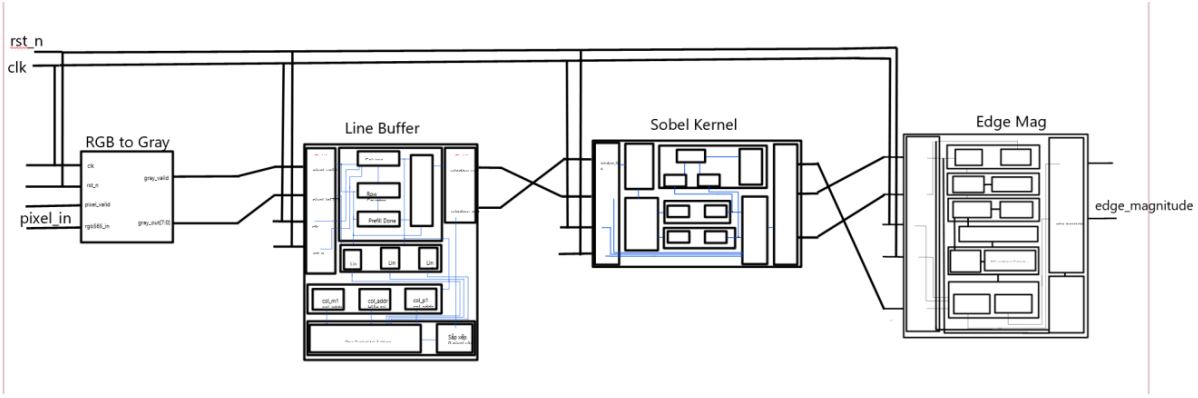
```
        edge_valid <= sobel_valid;
        if (sobel_valid) begin
            edge_magnitude <= magnitude_sat;
        end else begin
            edge_magnitude <= {OUTPUT_WIDTH{1'b0}};
        end
    end
end
```

Kết quả mô phỏng modulesim và RTL:





1.5 Khối Sobel Processor (Top Module)



Tên tham số	Giá trị mặc định	Chức năng
IMG_WIDTH	32	Chiều rộng ảnh
IMG_HEIGHT	24	Chiều cao ảnh
PIXEL_WIDTH	8	Độ rộng bit của pixel thang độ xám

Tên tín hiệu	I/O	Độ rộng	Chức năng
clk	Input	1	Clock hệ thống
rst_n	Input	1	Tín hiệu reset (tích cực mức thấp)
href	Input	1	Tín hiệu "Horizontal Reference" từ camera, dùng làm tín hiệu pixel_valid đầu vào
vsync	Input	1	Tín hiệu "Vertical Sync" từ camera
pixel_in	Input	16	Dữ liệu video RGB565 16 bit đầu vào (ảnh gốc)
sobel_enable	Input	1	Tín hiệu điều khiển: 1 = Bật Sobel, 0 = Tắt Sobel
pixel_valid	Output	1	Tín hiệu pixel_valid đầu ra (gửi tới HDMI)
pixel_out	Output	16	Dữ liệu video RGB565 16 bit đầu ra (gửi tới HDMI)

Module này kết nối các khối với nhau:

Kết nối các module theo thứ tự: RGB to Gray -> Line Buffer -> Sobel Kernel -> Edge Mag. Dữ liệu pixel_in đi vào và edge_magnitude 8 bit đi ra.

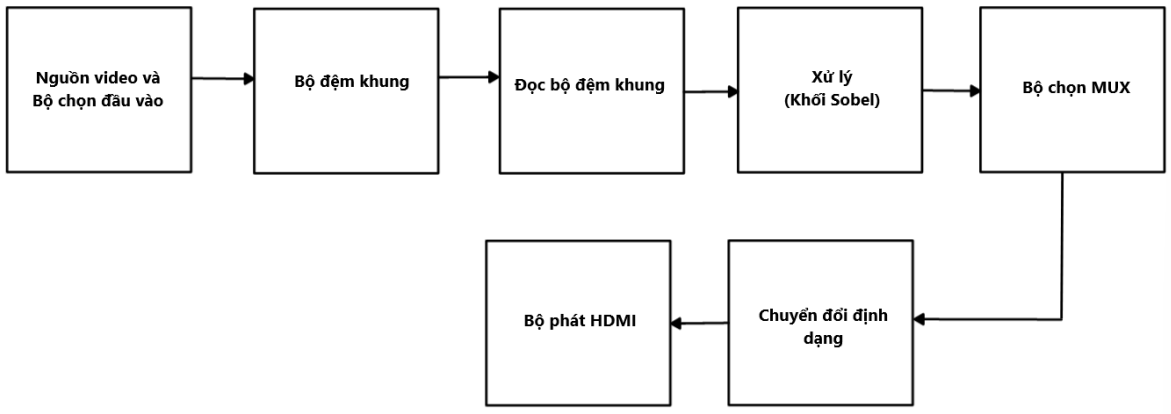
```
rgb_to_gray u_rgb2gray (
  .clk(clk), .rst_n(rst_n), .pixel_valid(href),
```

```
        .rgb565_in(pixel_in), .gray_out(gray_pixel), .gray_valid(gray_valid)
    );

    line_buffer #(.IMG_WIDTH(IMG_WIDTH)) u_linebuf (
        .clk(clk), .rst_n(rst_n), .pixel_valid(gray_valid),
        .pixel_in(gray_pixel), .window_valid(window_valid),
.window_out(window_flat)
    );

    sobel_kernel u_sobel (
        .clk(clk), .rst_n(rst_n), .window_valid(window_valid),
        .window_flat(window_flat), .sobel_valid(sobel_valid), .gx_out(gx), .gy_out(gy)
    );

    edge_mag u_magnitude (
        .clk(clk), .rst_n(rst_n), .sobel_valid(sobel_valid),
        .gx_in(gx), .gy_in(gy), .edge_valid(edge_valid),
.edge_magnitude(edge_magnitude)
    );
```

Chuyển đổi định dạng và sử dụng bộ MUX để chọn ảnh xuất là ảnh gốc hoặc ảnh sobel.

```
    assign pixel_valid = sobel_enable ? edge_valid : href;
    assign pixel_out = sobel_enable ? sobel_rgb565 : pixel_in;
```

2. VỊ TRÍ ĐẶT KHỐI XỬ LÝ SOBEL TRONG CAMERA HDMI



Mô tả xử lý: Thu thập tín hiệu input từ camera OV2640 sau đó qua một bộ đệm khung ghi vào bộ nhớ để xử lý chênh lệch clock giữa camera và HDMI. Sau đó chèn một khối xử lý

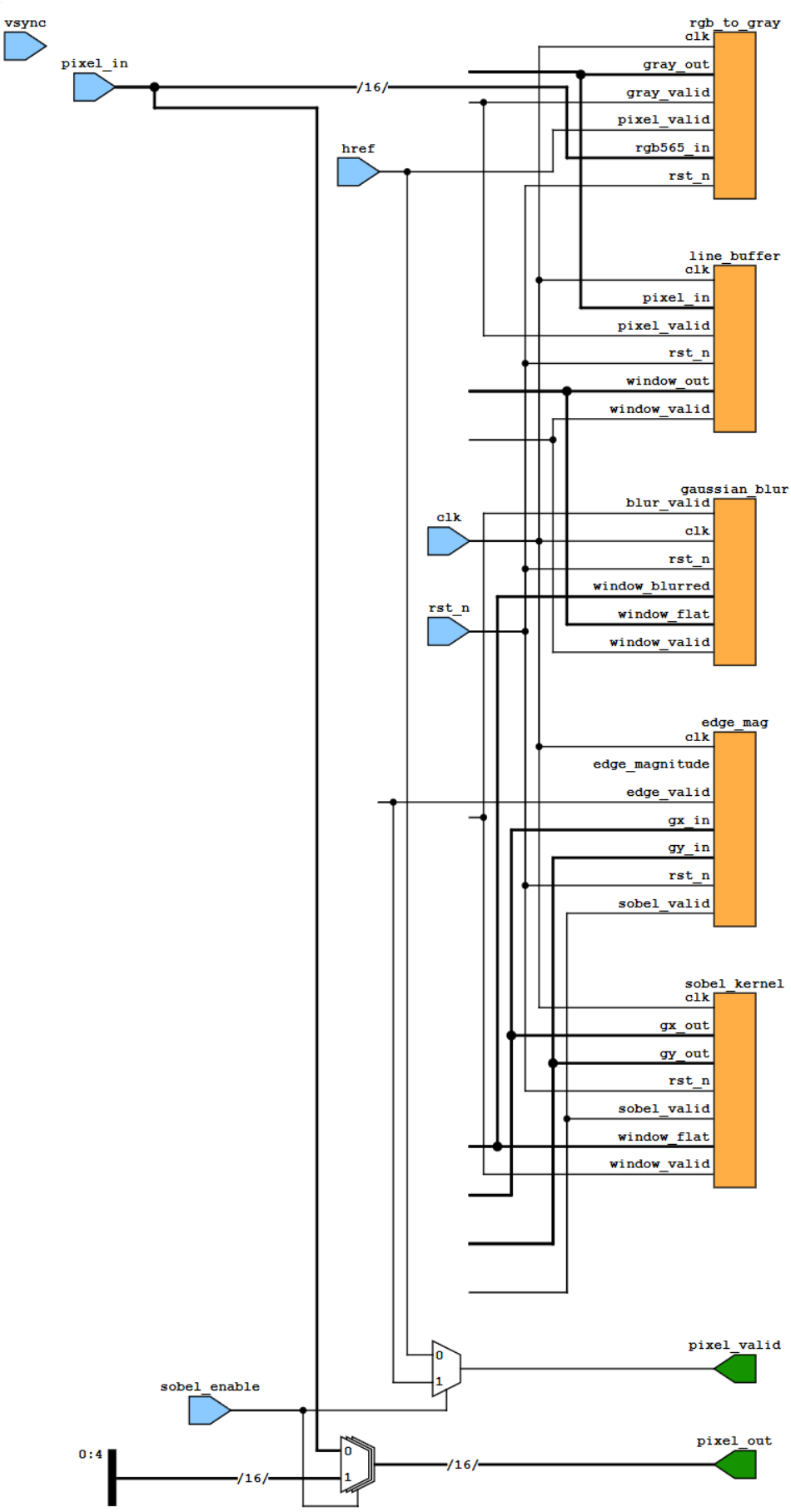
Sobel vào để thực hiện xử lý biên và đưa qua một bộ MUX để chọn ảnh gốc hoặc ảnh đã qua xử lý rồi xuất ra thông qua HDMI.

3. TÀI LIỆU THAM KHẢO

1. OpenCV, “Sobel Derivatives”, truy cập tại: https://docs.opencv.org/4.x/d2/d2c/tutorial_sobel_derivatives.html
2. Github, “Sobel_DE10”, truy cập tại: https://github.com/grant4001/Sobel_DE10
3. Hany Hamed, “Real-time edge detection using FPGA” (2018), truy cập tại: https://habr.com/ru/articles/431326/
4. Github, “Real-time-edge-detection-on-FPGA” (2018), truy cập tại: https://github.com/hany606/Real-time-edge-detection-on-FPGA
5. Handong Mo, Xinhong Zhou, Chenglang Lu – Advances in Engineering Innovation (2024) Volume 9, “Design of a real-time image processing system based on FPGA”, (31/07/2024)
6. Laigong Guo and Sitong Wu, “FPGA Implementation of a Real-Time Edge Detection System Based on an Improved Canny Algorithm” (08/01/2023), truy cập tại: https://www.mdpi.com/2076-3417/13/2/870
7. William Christensen and Brock Harris, “Sign Detection System for Real Time Applications” (03/2020), truy cập tại: https://digitalworks.union.edu/theses/2482

4. PHỤ LỤC BỔ SUNG

Sơ đồ tổng quan của khối xử lý Sobel và hệ thống:


vsync
pixel_in
/16/
href
rgb_to_gray
clk
gray_out
gray_valid
pixel_valid
rgb565_in
rst_n
line_buffer
clk
pixel_in
pixel_valid
rst_n
window_out
window_valid
gaussian_blur
blur_valid
clk
rst_n
window_blurred
window_flat
window_valid
clk
rst_n
edge_mag
clk
edge_magnitude
edge_valid
gx_in
gy_in
rst_n
sobel_valid
sobel_kernel
clk
gx_out
gy_out
rst_n
sobel_valid
window_flat
window_valid
0
1
pixel_valid
sobel_enable
0:4
/16/
0
1
/16/
pixel_out