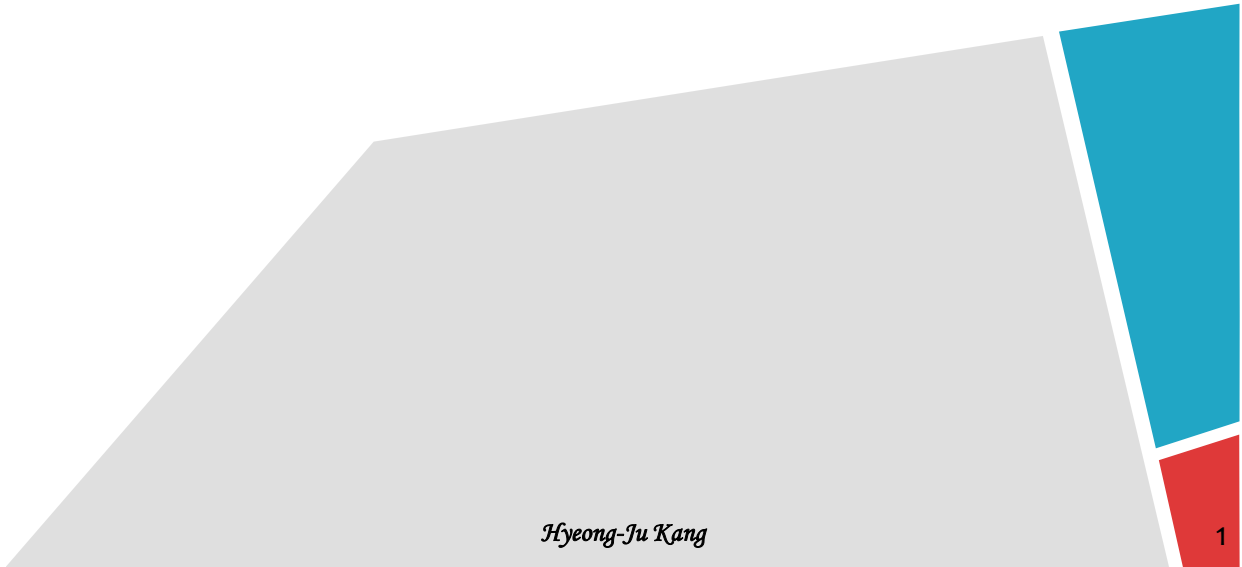




Lecture 2. 2D Filter



Outline



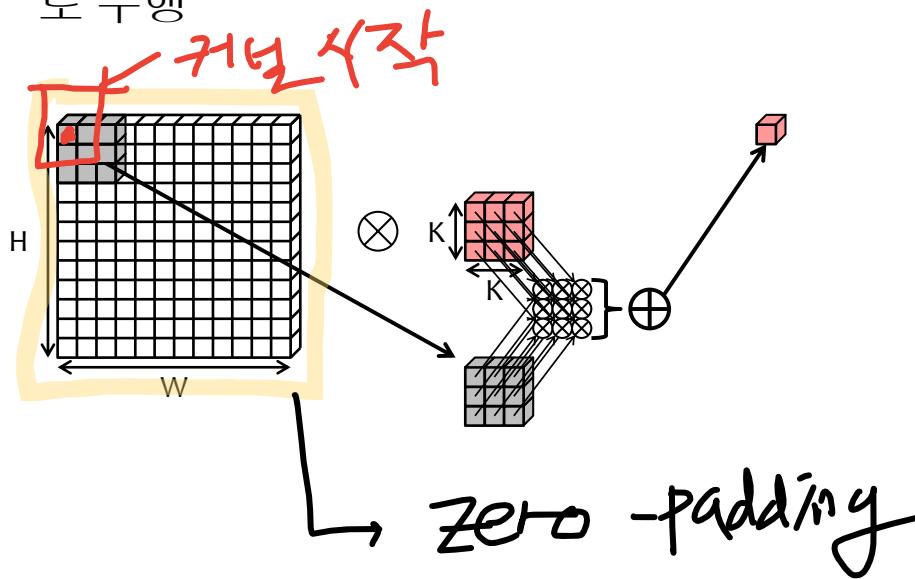
- ▶ Convolution on Image
- ▶ Fixed Point C 구현
- ▶ Verilog 구현
- ▶ Double Buffering
- ▶ Line Buffer 구조
- ▶ Throughput Optimization
- ▶ Direct Programming Interface
- ▶ Reusable Design



Convolution on Image



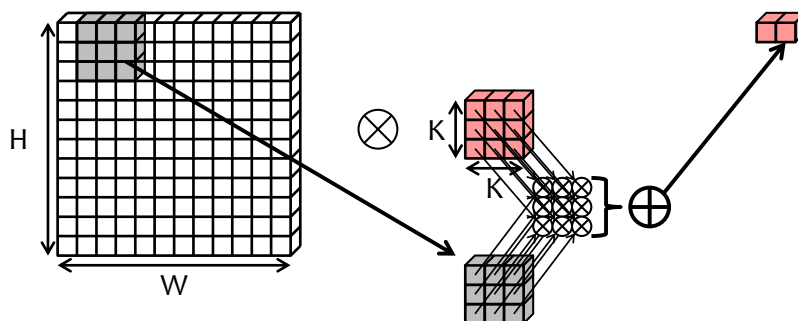
- ▶ 영상은 2차원 데이터, 그러므로 합성곱도 2차원으로 수행



Convolution on Image



- ▶ 영상은 2차원 데이터, 그러므로 합성곱도 2차원으로 수행

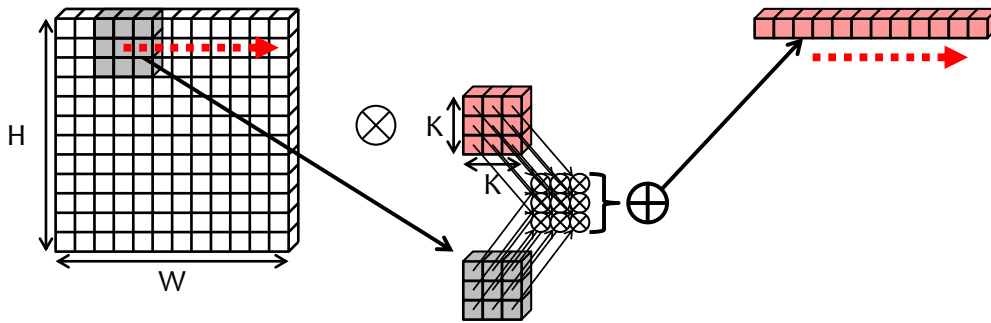




Convolution on Image



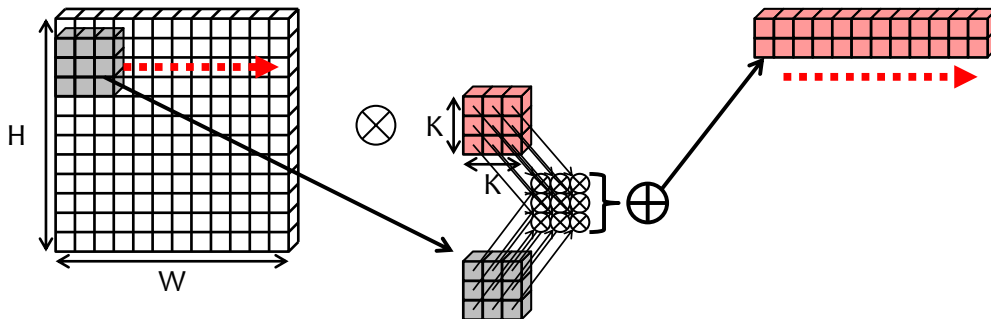
- ▶ 영상은 2차원 데이터, 그러므로 합성곱도 2차원으로 수행



Convolution on Image



- ▶ 영상은 2차원 데이터, 그러므로 합성곱도 2차원으로 수행





Convolution on Image



- ▶ 영상은 2차원 데이터, 그러므로 합성곱도 2차원으로 수행

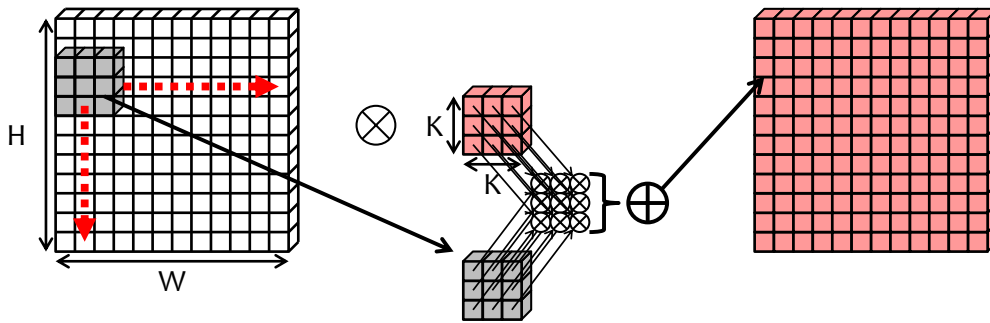


Image Filters

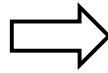


- ▶ Low Pass Filter Example

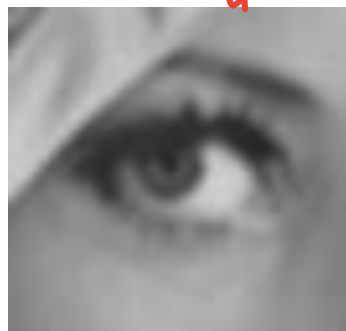
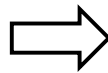
$1/9$	$1/9$	$1/9$
$1/9$	$1/9$	$1/9$
$1/9$	$1/9$	$1/9$

$1/16$	$1/8$	$1/16$
$1/8$	$1/4$	$1/8$
$1/16$	$1/8$	$1/16$

▶ Low Pass Filter Example



▶ Low Pass Filter Example



블러: 경계 제거
↓
EX. 잡티 보정



Image Filters



High Pass Filter Example

Laplacian

-1	-1	-1
-1	8	-1
-1	-1	-1

0	-1	0
-1	4	-1
0	-1	0

Prewitt Kernels

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

Sobel Kernels

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

0	1	2
-1	0	1
-2	-1	0

-2	-1	0
-1	0	1
0	1	2

가르 경계

상하

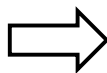
동향



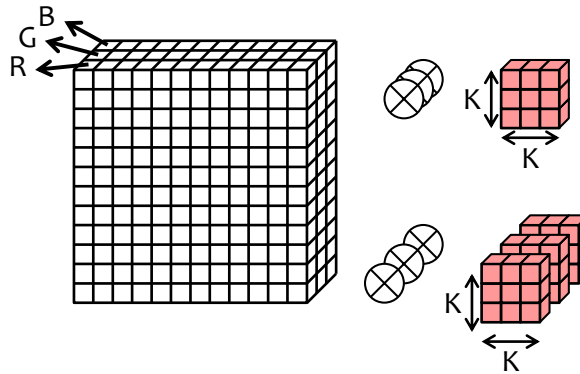
Image Filters



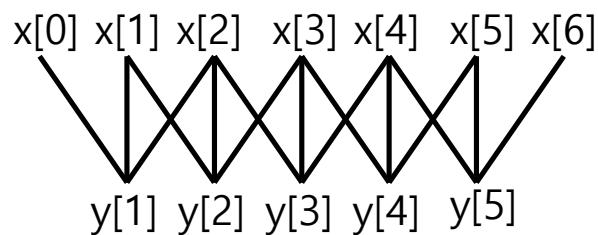
High Pass Filter Example



- ▶ Color image: R, G, B channels
 - ▶ Use the same kernel for all the channels
 - ▶ Or use different kernels



- ▶ 데이터 개수 감소



- ▶ N 개 데이터 $\rightarrow N-K+1$ 개 데이터
- ▶ 피하고 싶다면
 - ▶ Padding **0** x[0] x[1] x[2] x[3] x[4] x[5] x[6] **0**
 - ▶ Copying **x[0]** x[0] x[1] x[2] x[3] x[4] x[5] x[6] **x[6]**
 - ▶ Wrapping **x[6]** x[0] x[1] x[2] x[3] x[4] x[5] x[6] **x[0]**

- ▶ Convolution on Image
- ▶ Fixed Point C 구현
- ▶ Verilog 구현
- ▶ Double Buffering
- ▶ Line Buffer 구조
- ▶ Throughput Optimization
- ▶ Direct Programming Interface
- ▶ Reusable Design



- ▶ Gray image
 - ▶ 하나의 픽셀은 주로 8bit의 데이터(0~255)로 표현됨 → unsigned char 자료형
 - ▶ 이미지는 2차원 → 2차원 배열

```
unsigned char  img[256][256];
```

- ▶ Color image
 - ▶ 하나의 픽셀이 3개(R, G, B)의 8bit 데이터로 구성됨

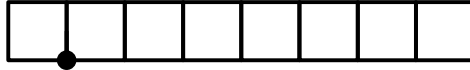
```
unsigned char  img[256][256][3];  
unsigned char  img[3][256][256];
```



▶ 3x3 filter

```
int h[3][3];
```

▶ signed (8,7)



▶ Filter 자체를 구현한 코드

```
void filter2d(unsigned char in_img[], unsigned char out_img[]) {
    int h[3][3] = { ... };
    for(int i=0; i<256; i++) {
        for(int j=0; j<256; j++) {
            int sum = 0;
            sum += in_img[i-1][j-1] * h[0][0];
            sum += in_img[i-1][j] * h[0][1];
            sum += in_img[i-1][j+1] * h[0][2];
            sum += in_img[i][j-1] * h[1][0];
            sum += in_img[i][j] * h[1][1];
            sum += in_img[i][j+1] * h[1][2];
            sum += in_img[i+1][j-1] * h[2][0];
            sum += in_img[i+1][j] * h[2][1];
            sum += in_img[i+1][j+1] * h[2][2];
            out_img[i][j] = sum;
        }
    }
}
```

-----> 입력 이미지: 2차원 배열
결과 이미지: 2차원 배열

▶ 2차원 배열 매개변수?

```
void filter2d(unsigned char in_img[][], unsigned char out_img[][]) {
```

- ▶ C언어 문법상 첫번째 크기만 비울 수 있음

```
...(unsigned char in_img[256][256], unsigned char out_img[256][256]) {
```

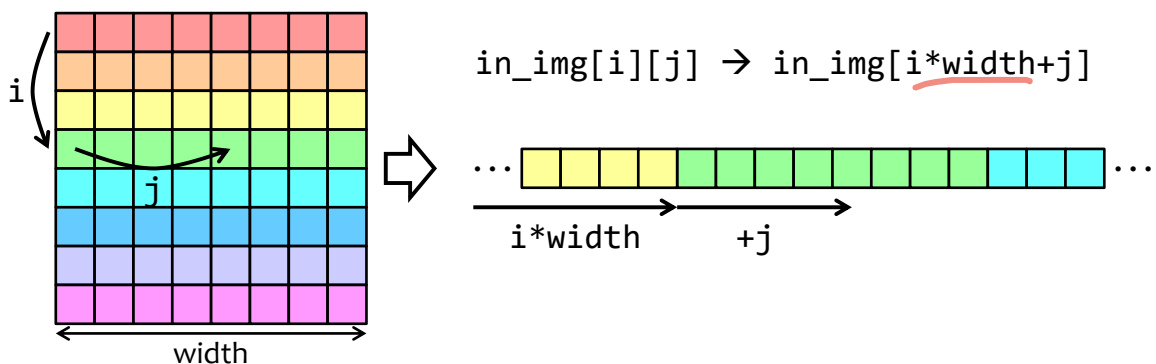
```
...(unsigned char in_img[][256], unsigned char out_img[][256]) {
```

- ▶ 처리할 수 있는 이미지의 가로 크기가 고정됨

▶ 보통 1차원 배열 매개 변수 사용

```
void filter2d(unsigned char in_img[], unsigned char out_img[]) {
```

- ▶ 2차원 배열 index를 1차원 배열 index로 변환 필요
 - ▶ 실제 메모리에서는 2차원 배열을 1차원적으로 배치



▶ Filter 자체를 구현한 코드

```
void filter2d(unsigned char in_img[], unsigned char out_img[],
             int height, int width) {
    int h[3][3] = { ... };
    for(int i=0; i<height; i++) {
        for(int j=0; j<width; j++) {
            int sum = 0;
            sum += in_img[(i-1)*width+j-1] * h[0][0];
            sum += in_img[(i-1)*width+j]   * h[0][1];
            sum += in_img[(i-1)*width+j+1] * h[0][2];
            sum += in_img[(i )*width+j-1]   * h[1][0];
            sum += in_img[(i )*width+j]     * h[1][1];
            sum += in_img[(i )*width+j+1]   * h[1][2];
            sum += in_img[(i+1)*width+j-1] * h[2][0];
            sum += in_img[(i+1)*width+j]   * h[2][1];
            sum += in_img[(i+1)*width+j+1] * h[2][2];
            out_img[i*width+j] = sum;
        }
    }
}
```

-----> 입력 이미지: 1차원 배열
결과 이미지: 1차원 배열


index 변환

(i, j)

▶ Precision

- ▶ image pixel: unsigned (8,0)
 - ▶ coefficient: signed(8,7)
 - ▶ multiplication result: (16,7)
 - ▶ after sum: (20, 7)
- ▶ The filter coefficients are usually generated so that the result values are also 0~255.

rounding



```
sum = (sum + (1<<6)) >> 7;
if(sum < 0) out_img[i*width+j] = 0;
else if(sum > 255) out_img[i*width+j] = 255;
else out_img[i*width+j] = sum;
```



Padding



```

void filter2d(unsigned char in_img[], unsigned char out_img[],
             int height, int width) {
    int h[3][3] = { ... };
    for(int i=0; i<height; i++) {
        for(int j=0; j<width; j++) {
            int sum = 0;
            sum += in_img[(i-1)*width+j-1] * h[0][0];
            sum += in_img[(i-1)*width+j] * h[0][1];
            sum += in_img[(i-1)*width+j+1] * h[0][2];
            sum += in_img[(i)*width+j-1] * h[1][0];
            sum += in_img[(i)*width+j] * h[1][1];
            sum += in_img[(i)*width+j+1] * h[1][2];
            sum += in_img[(i+1)*width+j-1] * h[2][0];
            sum += in_img[(i+1)*width+j] * h[2][1];
            sum += in_img[(i+1)*width+j+1] * h[2][2];
            sum = (sum + (1<<6)) >> 7;
            if(sum < 0) out_img[i*width+j] = 0;
            else if(sum > 255) out_img[i*width+j] = 255;
            else out_img[i*width+j] = sum;
        }
    }
}

```

Annotations in the original image:

- Red dashed arrows pointing to `(i-1)*width+j-1`, `(i-1)*width+j`, `(i-1)*width+j+1`, `(i)*width+j-1`, `(i)*width+j`, `(i)*width+j+1`, `(i+1)*width+j-1`, `(i+1)*width+j`, and `(i+1)*width+j+1` with the text "i가 0이면? j가 0이면?".
- Red dashed arrows pointing to `(i+1)*width+j-1` and `(i+1)*width+j` with the text "i가 height-1이면? j가 width-1이면?".
- A green circle around the word "padding" in the text "copying padding wrapping" with an arrow pointing to the code.

copying
padding
wrapping



Padding



```

void filter2d(unsigned char in_img[], unsigned char out_img[],
             int height, int width) {
    int h[3][3] = { ... };
    for(int i=0; i<height; i++) {
        for(int j=0; j<width; j++) {
            int sum = 0;
            if(i>0 && j>0) sum += in_img[(i-1)*width+j-1]*h[0][0];
            if(i>0) sum += in_img[(i-1)*width+j]*h[0][1];
            if(i>0 && j<width-1) sum += in_img[(i-1)*width+j+1]*h[0][2];
            if(j>0) sum += in_img[(i)*width+j-1]*h[1][0];
            sum += in_img[(i)*width+j]*h[1][1];
            sum += in_img[(i)*width+j+1]*h[1][2];
            if(j<width-1) sum += in_img[(i+1)*width+j-1]*h[2][0];
            if(i<height-1 && j>0) sum += in_img[(i+1)*width+j]*h[2][1];
            if(i<height-1) sum += in_img[(i+1)*width+j+1]*h[2][2];
            sum = (sum + (1<<6)) >> 7;
            if(sum < 0) out_img[i*width+j] = 0;
            else if(sum > 255) out_img[i*width+j] = 255;
            else out_img[i*width+j] = sum;
        }
    }
}

```

Annotation in the original image:

- Red dashed arrow pointing to the conditional checks with the text "원래는 범위를 벗어나면 0을 곱해야 하지만 누적을 안 하는 것으로 대체".

▶ Filter의 입출력을 담당하는 코드

```
int main(void) {
    FILE      *inf, *outf;
    unsigned char  in_img[256*256];
    unsigned char  out_img[256*256];
    inf = fopen("img_in.txt", "r");
    outf = fopen("img_out.txt", "w");

    for(int i;i<256*256;i++) {
        fscanf(inf, "%d,", in_img+i);
    }
    filter2d(in_img, out_img, 256, 256);
    for(int i;i<256*256;i++) {
        fprintf(outf, "%d ", out_img[i]);
        if(i%256 == 255) fprintf(outf, "\n");
    }

    fclose(inf);
    fclose(outf);
}
```

○○○ 입력 파일 생성과 결과 확인 ○○○

▶ 이미지 파일을 읽어서 크기를 조절한 뒤 텍스트로 저장

```
A = imread('lena_gray.png');    % 512x512 image
B = imresize(A, 0.5);           % to 256x256 image
imshow(B);
dlmwrite('img_in.txt', B);
```

▶ 필터 처리된 출력 텍스트 파일을 읽어서 이미지로 보여 주기

```
img = uint8(importdata('img_out.txt'));
imshow(img);
```

- ▶ Convolution on Image
- ▶ Fixed Point C 구현
- ▶ Verilog 구현
- ▶ Double Buffering
- ▶ Line Buffer 구조
- ▶ Throughput Optimization
- ▶ Direct Programming Interface
- ▶ Reusable Design

- ▶ 각 출력 pixel을 계산
 - ▶ x: 0~255, y: 0~255 → counting

y $\overline{\text{X} \quad 0 \quad \text{X}}$ $\overline{\text{X} \quad 1 \quad \text{X}}$... $\overline{\text{X} \quad 254 \quad \text{X}}$
 x $\overline{\text{X} 0 \text{X}} \overline{\text{X} 1 \text{X}} \overline{\text{X} 2 \text{X}}$... $\overline{\text{X} 255 \text{X}} \overline{\text{X} 0 \text{X}} \overline{\text{X} 1 \text{X}}$... $\overline{\text{X} 255 \text{X}}$... $\overline{\text{X} 255 \text{X}} \overline{\text{X} 0 \text{X}} \overline{\text{X} 1 \text{X}}$... $\overline{\text{X} 255 \text{X}}$

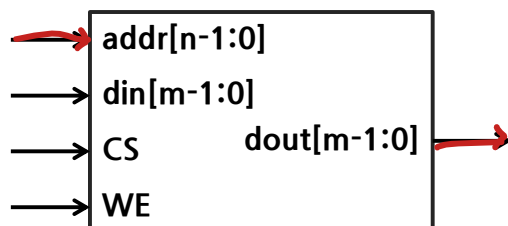
- ▶ 한 pixel 안에서는?
 - ▶ 이미지 데이터는 메모리에 저장되어 있음 → 9번의 메모리 read → 9 cycle
 - ▶ 9번의 곱셈 및 합 구하기 → 곱셈기를 9개 사용하면 한 cycle에 가능
 - ▶ 결과를 메모리에 저장 → 1 번의 메모리 write → 1 cycle



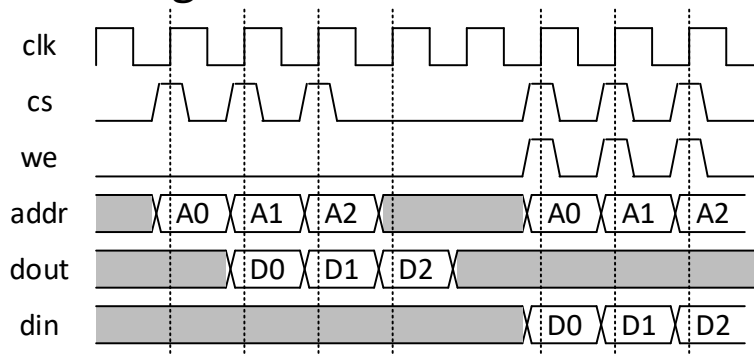
Memory (SRAM) Interface



SRAM ports



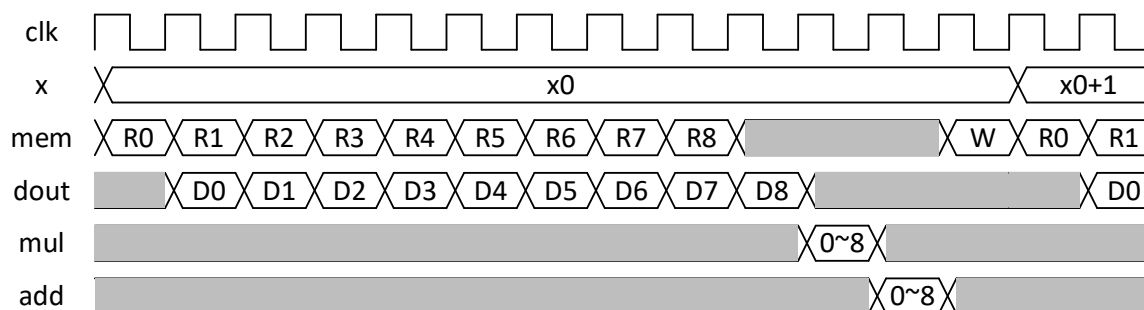
SRAM i/f timing



Timing Diagram



Timing diagram of a pixel



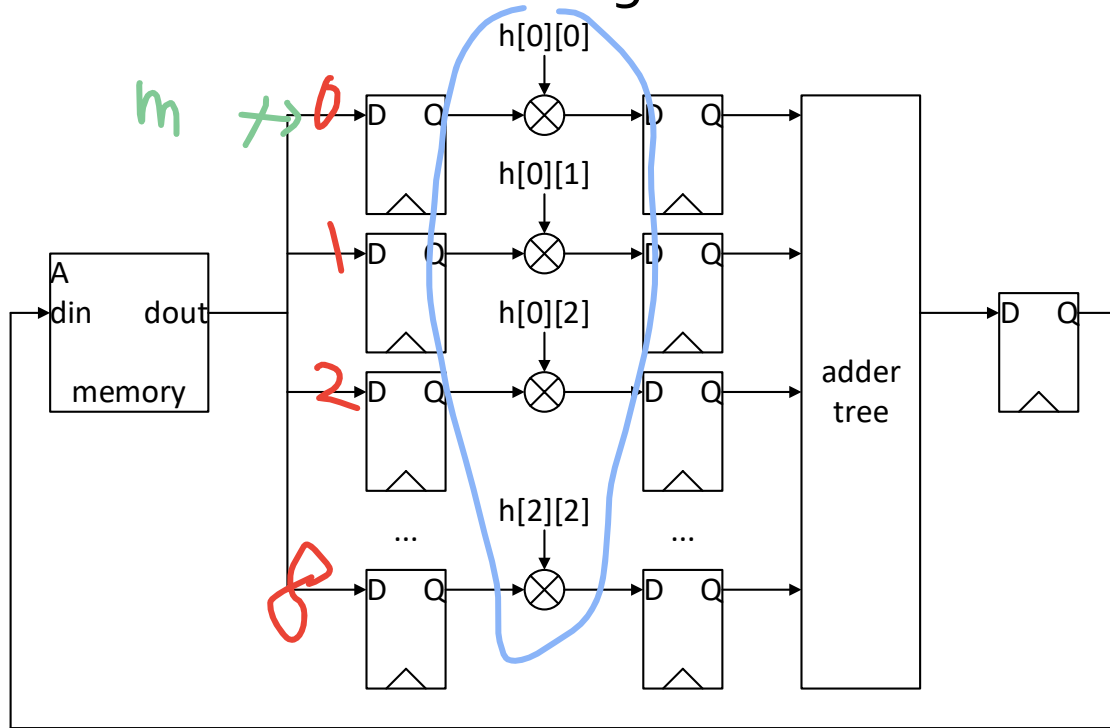
13 cycles per pixel

pipe-fill 전값 8개는 꼭시

○○○

Block Diagram

○○○



Hyeong-Ju Kang

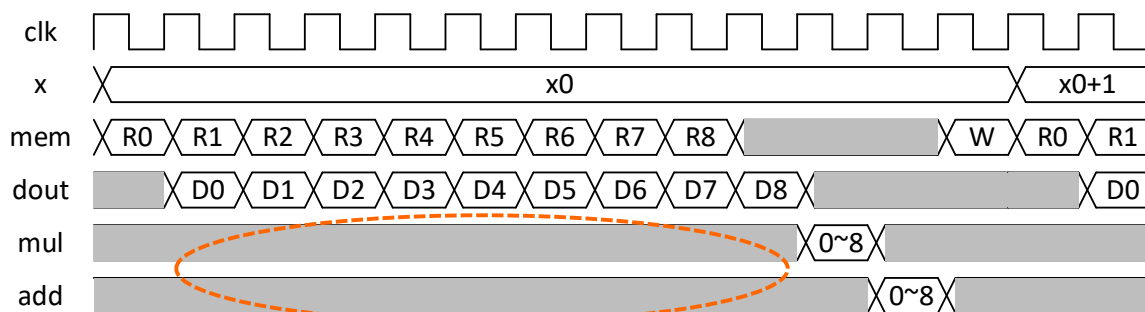
31

○○○

Timing Diagram

○○○

Timing diagram of a pixel



- ▶ 13 cycles per pixel
- ▶ 9 multipliers do nothing for 12 cycles.
- ▶ Let's use just one multiplier.

▶ operator vs cycle time trade-off

* Sync

Hyeong-Ju Kang

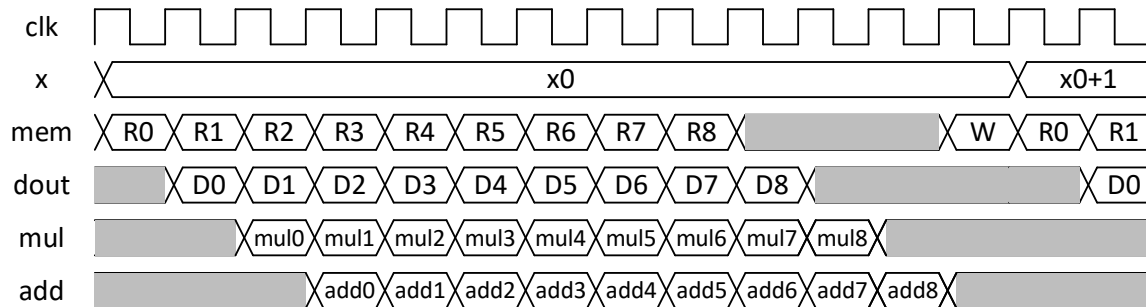
32



Timing Diagram



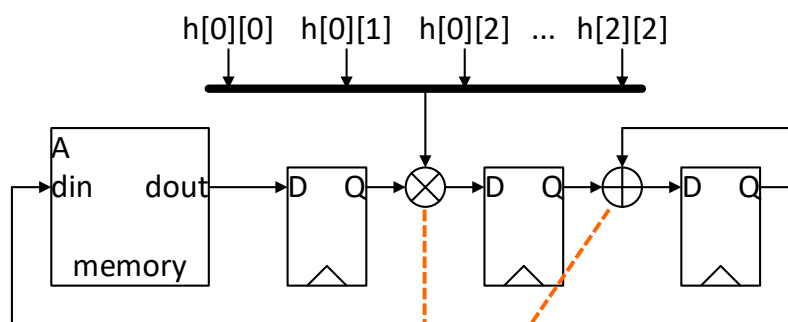
▶ Timing diagram of a pixel



- ▶ 13 cycles per pixel
- ▶ Let's use just one multiplier.
 - ▶ operator vs cycle time trade-off
 - ▶ The multiplier does something for 9 cycles.



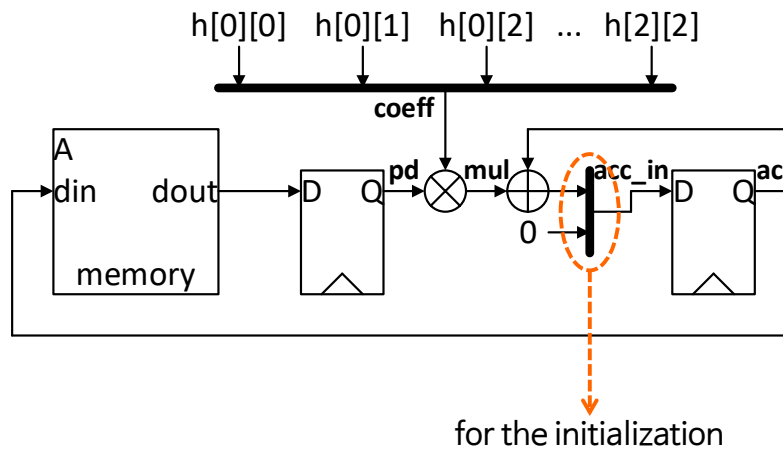
Block Diagram



multiplication and addition →
MAC (multiplication and accumulation) block



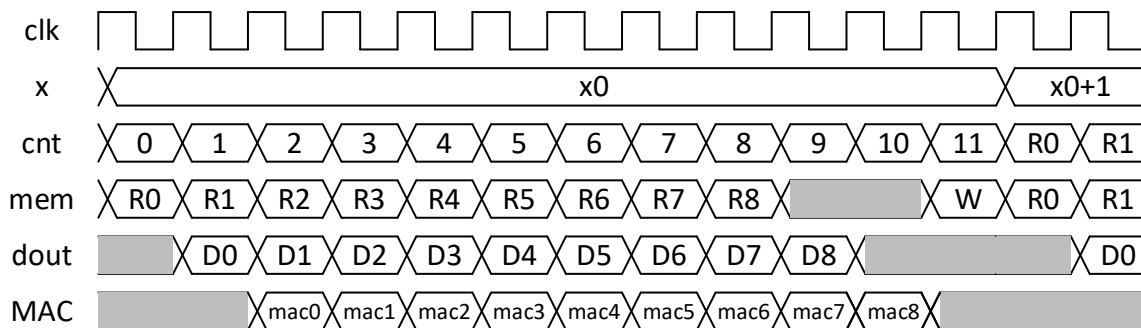
Block Diagram



Timing Diagram

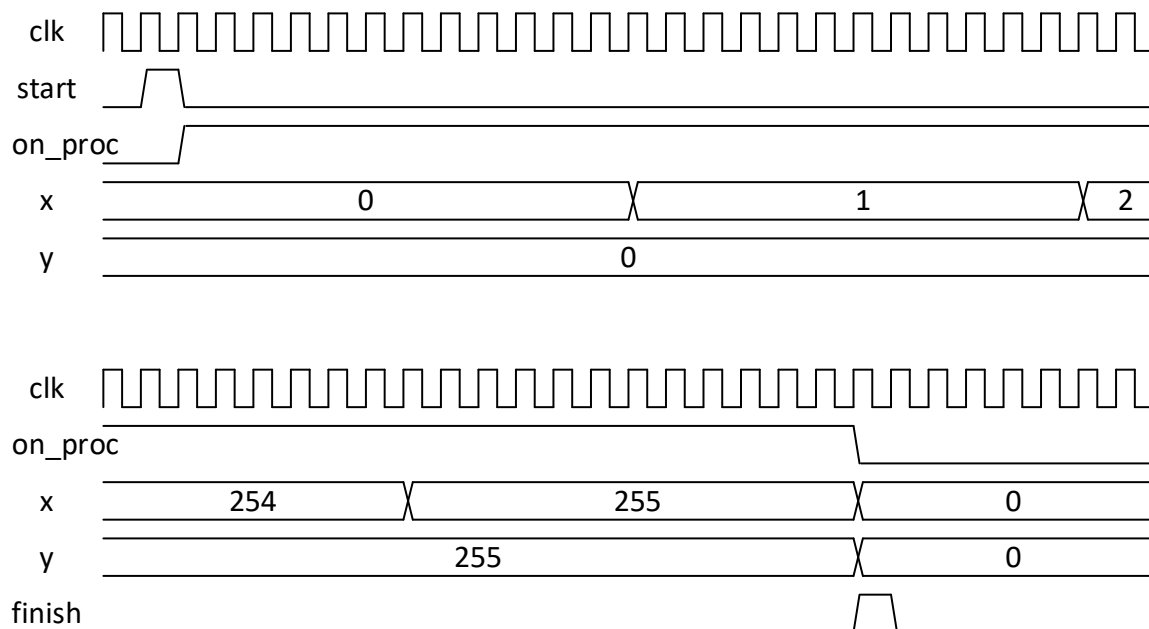
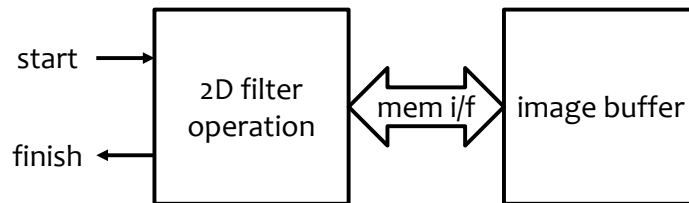


▶ Timing diagram of a pixel



▶ 12 cycles per pixel

- ▶ Input image is assumed to be stored in an image buffer.
 - ▶ Who store an input image in the buffer?
 - ▶ After an image is stored, the start signal is asserted.
 - ▶ When the filter operation finishes, the finish signal is asserted.





Verilog Code



```
module filter2d (  
    input      clk,  
    input      n_reset,  
    input      start,  
    output reg  finish,    } start & finish  
    output      cs,  
    output      we,  
    output [15:0] addr,  
    output [7:0] din,  
    input  [7:0] dout,  
    input      h_write,  
    input  [3:0] h_idx,  
    input  [7:0] h_data    } memory interface  
);  
  
    } coefficient write  
  
reg      on_proc; --> 1 during the process  
reg [3:0] cnt;  
reg [7:0] cnt_x;  
reg [7:0] cnt_y;    } counting signals
```



Verilog Code



```
always@(posedge clk or negedge n_reset) begin  
    if(n_reset == 1'b0) begin  
        on_proc <= 1'b0;  
        cnt <= 0;  
        cnt_x <= 0;  
        cnt_y <= 0;  
        finish <= 1'b0;  
    end else begin  
        if(start == 1'b1) on_proc <= 1'b1;  
        else if((cnt == 11)&&(cnt_x == 255)&&(cnt_y == 255)) on_proc <= 1'b0;  
        if(  
            clk  
            start  
            on_proc  
            x  
            y  
            0  
            1  
            2  
            0  
            254  
            255  
            0  
            0  
            255));  
    end  
end  
end
```

if start, turn on on_proc
after the last cycle of the last pixel, turn off on_proc

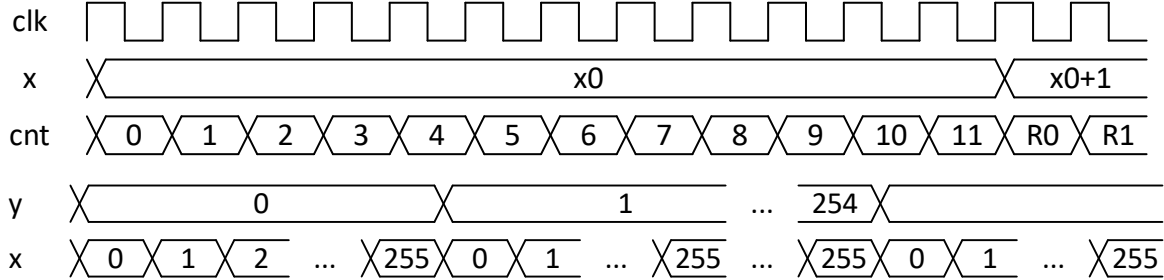
Timing diagram showing signals: clk, start, on_proc, x, y, and finish. The start signal is a pulse. The on_proc signal is high during the start pulse and then turns off after the last pixel (x=255, y=255). The x and y signals are shown as counters from 0 to 255. The finish signal is high at the end of the process.



Verilog Code



```
always@(posedge clk or negedge n_reset) begin
```



```

    if(on_proc == 1'b1) begin----->during the process
        cnt <= (cnt == 11) ? 0 : cnt+1;----->12 cycles in a pixel
        if(cnt == 11) begin
            cnt_x <= (cnt_x == 255) ? 0 : cnt_x+1;
            if(cnt_x == 255) begin
                cnt_y <= (cnt_y == 255) ? 0 : cnt_y+1;
            end
        end
    end
    finish <= ((cnt == 11) && (cnt_x == 255) && (cnt_y == 255));
end

```

} pixel counting

end

Hyeong-Ju Kang

41



Verilog Code



```
always@(posedge clk or negedge n_reset) begin
```

```
    if(n_reset == 1'b0) begin
```

```
        on_proc <= 1'b0;
```

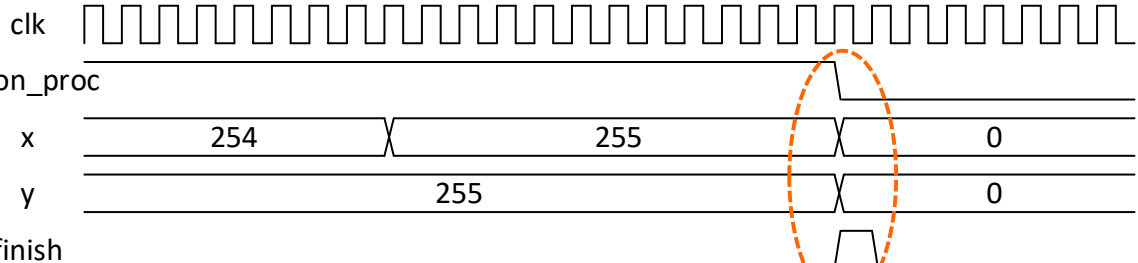
```
        cnt <= 0;
```

```
        cnt_x <= 0;
```

```
        cnt_y <= 0;
```

```
        finish <= 1'b0;
```

```
    end else begin
```



```

    end
    end
    end
    finish <= ((cnt == 11) && (cnt_x == 255) && (cnt_y == 255));
end

```

after the last cycle of the last pixel, assert finish

end

Hyeong-Ju Kang

42

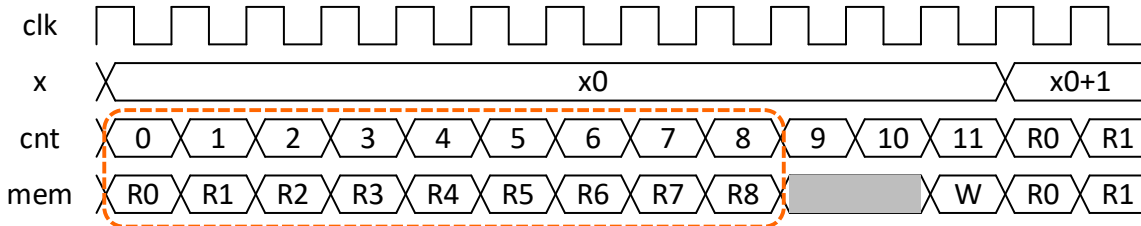
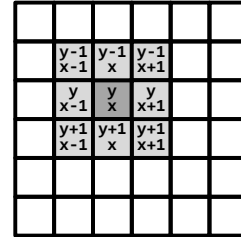


Verilog Code



```
wire      mem_rd = (cnt >= 0) && (cnt <= 8) && (on_proc == 1'b1);
reg [15:0] rd_addr;
always@(*) begin
  case(cnt)
    4'd0: rd_addr = (cnt_y-1)*256 + cnt_x-1;
    4'd1: rd_addr = (cnt_y-1)*256 + cnt_x;
    4'd2: rd_addr = (cnt_y-1)*256 + cnt_x+1;
    4'd3: rd_addr = (cnt_y )*256 + cnt_x-1;
    4'd4: rd_addr = (cnt_y )*256 + cnt_x;
    4'd5: rd_addr = (cnt_y )*256 + cnt_x+1;
    4'd6: rd_addr = (cnt_y+1)*256 + cnt_x-1;
    4'd7: rd_addr = (cnt_y+1)*256 + cnt_x;
    4'd8: rd_addr = (cnt_y+1)*256 + cnt_x+1;
    default: rd_addr = 'bx;
  endcase
end
```

-----> memory read when cnt=0~8



Hyeon-Ju Kang

43

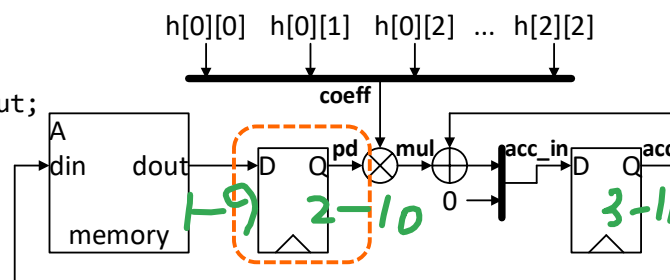


Verilog Code



```
reg [7:0] pd;
wire      pd_en = (cnt >= 1) && (cnt <= 9);
always@(posedge clk or negedge n_reset) begin
  if(n_reset == 1'b0) begin
    pd <= 0;
  end else begin
    if(pd_en == 1'b1) pd <= dout;
  end
end
```

dout is available when cnt=1~9
change pd at that time



Hyeon-Ju Kang

44



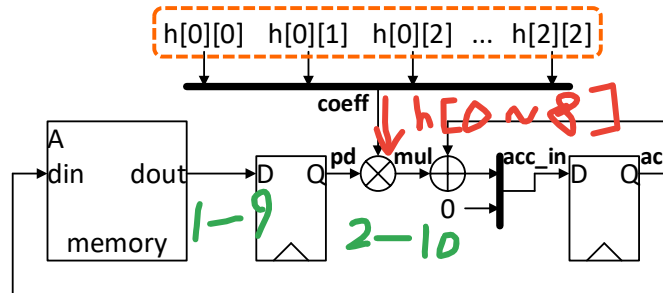
Verilog Code



```

reg signed [7:0] h[0:8];
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        h[0] <= 8'h08;
        ...
    end else begin
        if(h_write == 1'b1) begin
            h[h_idx] <= h_data;
        end
    end
end
end

```



```

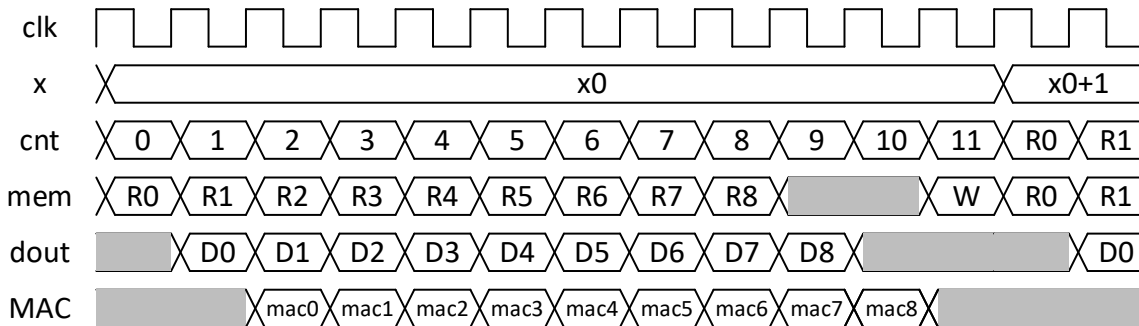
wire signed [7:0] coeff = h[cnt-2];
wire signed [15:0] mul = pd * h;
reg signed [19:0] acc;
wire signed [19:0] acc_in = (cnt == 1) ? 0 : mul + acc;

```

acc rst ↑



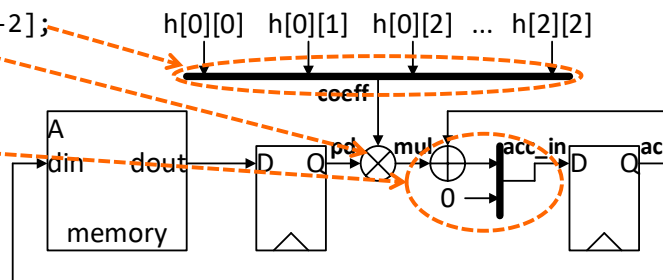
Verilog Code



```

wire signed [7:0] coeff = h[cnt-2];
wire signed [15:0] mul = pd * h;
reg signed [19:0] acc;
wire signed [19:0] acc_in =
    (cnt == 1) ? 0 : mul + acc;

```





Verilog Code



```

reg          acc_en;
always@(*) begin
    acc_en = 1'b0;
    case(cnt)
        4'd 1: acc_en = 1'b1;
        4'd 2: if((cnt_y > 0) && (cnt_x > 0)) acc_en = 1'b1;
        4'd 3: if((cnt_y > 0)                ) acc_en = 1'b1;
        4'd 4: if((cnt_y > 0) && (cnt_x < 255)) acc_en = 1'b1;
        4'd 5: if( cnt_x > 0)   acc_en = 1'b1;
        4'd 6:                acc_en = 1'b1;
        4'd 7: if( cnt_x < 255) acc_en = 1'b1;
        4'd 8: if((cnt_y < 255) && (cnt_x > 0)) acc_en = 1'b1;
        4'd 9: if((cnt_y < 255)                ) acc_en = 1'b1;
        4'd10: if((cnt_y < 255) && (cnt_x < 255)) acc_en = 1'b1;
        default: acc_en = 1'b0;
    endcase
end
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        acc <= 'b0;
    end else begin
        if(acc_en == 1'b1) acc <= acc_in;
    end
end

```

end

Hyeon-Ju Kang

47



Verilog Code



```

wire [19:0] pd_rnd_1 = acc + (1<<6);
wire [12:0] pd_rnd = pd_rnd_1[19:7];
wire [7:0] pd_out = (pd_rnd < 0) ? 0 :
                    (pd_rnd > 255) ? 255 :
                    pd_rnd[7:0];
assign      din = pd_out;

```

rounding & saturation



```

sum = (sum + (1<<6)) >> 7;
if(sum < 0) out_img[i*width+j] = 0;
else if(sum > 255) out_img[i*width+j] = 255;
else out_img[i*width+j] = sum;

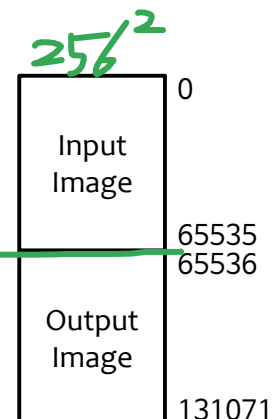
```

```

wire      mem_wr = (cnt == 11);
wire [16:0] wr_addr = cnt_y * 256 + cnt_x + 256*256;
assign cs = mem_rd | mem_wr;
assign we = mem_wr;
assign addr = (mem_rd == 1'b1) ? rd_addr : wr_addr;
endmodule

```

memory write



Hyeon-Ju Kang

48



Testbench



```
module top_filter_2d;

reg    clk, n_reset;
reg    start;
wire   finish;
initial clk = 1'b0;
always #5 clk = ~clk;

initial begin
    n_reset = 1'b1;
    $readmemh("../c/img_in.dat", i_buf.data); --> To store image data on
    #3;                                         memory
    n_reset = 1'b0;
    #20;
    n_reset = 1'b1;
    @(posedge clk);
    @(posedge clk);
    @(posedge clk);
    start = 1'b1; --> start 신호
    @(posedge clk);
    start = 1'b0;
end
```



Testbench



```
wire  cs, we;
wire  [16:0] addr;
wire  [7:0] din;
wire  [7:0] dout;

filter2d i_filter
    (.clk(clk), .n_reset(n_reset), .start(start), .finish(finish),
     .cs(cs), .we(we), .addr(addr), .din(din), .dout(dout),
     .h_write(1'b0), .h_idx('b0), .h_data('b0));

mem_single #(
    .WD(8),
    .DEPTH(256*256*2)
) i_buf (
    .clk(clk),
    .cs(cs),
    .we(we),
    .addr(addr),
    .din(din),
    .dout(dout)
);
```



Testbench



```
always@(posedge clk) begin
    if(finish == 1'b1) begin -----> if finish is asserted
        for(int i=0;i<256;i++) begin
            for(int j=0;j<256;j++) begin
                $write("%3d ", i_buf.data[i*256+j+256*256]);
            end
            $write("\n");
        end
        $finish; -----> to finish the simulation
    end
end
endmodule
```

} to print output image on the screen



Outline



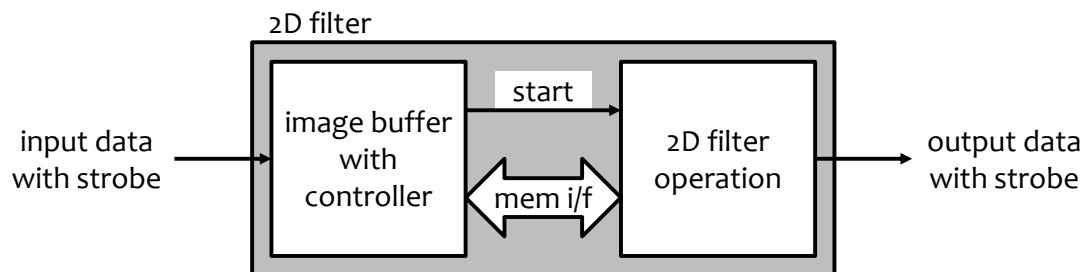
- ▶ Convolution on Image
- ▶ Fixed Point C 구현
- ▶ Verilog 구현
- ▶ Double Buffering
- ▶ Line Buffer 구조
- ▶ Throughput Optimization
- ▶ Direct Programming Interface
- ▶ Reusable Design

▶ 이전 설계

- ▶ image buffer가 testbench에 위치
- ▶ image buffer에 임의로 접근 가능
- ▶ 누가 image buffer에 이미지 저장?

▶ 이번에는

- ▶ 입력 데이터가 하나 씩 차례로 들어옴
- ▶ image buffer도 설계 안에
- ▶ 하나 씩 들어오는 데이터를 차례로 저장하고 모두 저장 되면 처리한 뒤 하나 씩 출력

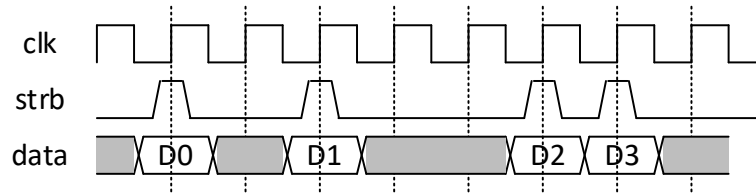


▶ Buffer controller의 역할

- ▶ 입력 데이터를 image buffer의 적절한 위치에 저장
- ▶ 한 frame의 입력 데이터 저장이 끝나면 start 신호를 보냄

▶ Strobed input

- ▶ strb 신호가 1일 때 유효한 데이터가 입력됨

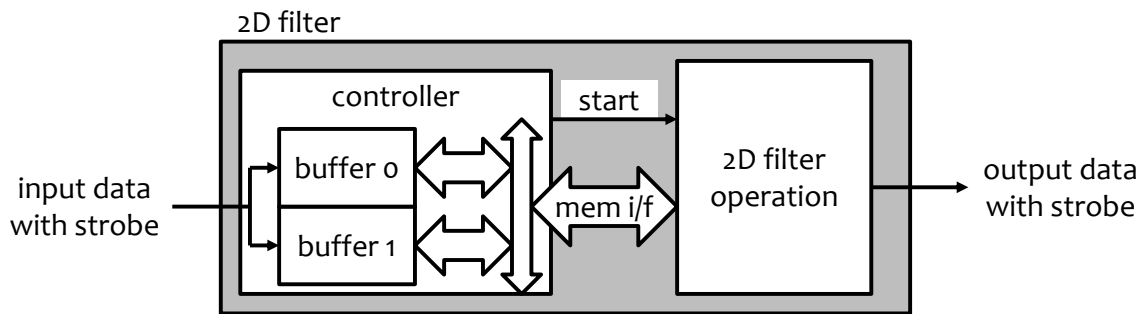


▶ 데이터 충돌

- ▶ 한 frame의 데이터를 Image buffer에 저장한 뒤 처리 시작
- ▶ 처리하는 동안 다음 frame의 데이터가 수신됨
- ▶ 아직 처리하지 않은 데이터가 새 데이터로 덮어 쓰여질 수 있음

▶ Double Buffering

- ▶ 두 개의 image buffer 사용
- ▶ 한 쪽의 image buffer에서 데이터를 처리하는 동안, 다른 image buffer에 수신되는 데이터를 저장
- ▶ 데이터 처리와 한 frame 데이터의 수신이 끝나면 두 image buffer의 역할을 바꿈



▶ Buffer controller의 역할

- ▶ 입력 데이터를 image buffer의 적절한 위치에 저장
- ▶ 한 frame의 입력 데이터 저장이 끝나면 start 신호를 보냄
- ▶ 입력 데이터 버퍼와 처리 데이터 버퍼의 선택

```

module filter2d_buf (
    input        clk,
    input        n_reset,

    input        i_strb,
    input [7:0]  i_data, } input data with strobe

    output reg   start, -----> start signal
    input        mem_rd,
    input [15:0] rd_addr, } buffer read from
    output [7:0] rd_data } operation block
);
reg [7:0] cnt_x;
reg [7:0] cnt_y;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        cnt_x <= 255;
        cnt_y <= 255;
    end else begin
        if(i_strb == 1'b1) begin -----> when i_strb is asserted,
            cnt_x <= (cnt_x == 255) ? 0 : cnt_x+1;
            if(cnt_x == 255) begin
                cnt_y <= (cnt_y == 255) ? 0 : cnt_y+1; } pixel counting
            end
        end
    end
end
end

```



Buffer Controller



```

reg      mode;
wire     mode_change;
reg      mem_wr;
reg[7:0] wr_data;

assign mode_change = (mem_wr==1'b1) && (cnt_x == 255) && (cnt_y == 255);

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        mode <= 1'b0;
        start <= 1'b0;
    end else begin
        if(mode_change == 1'b1) begin
            mode <= ~mode;
        end
        start <= mode_change;
    end
end

```

0: buf0에 입력 데이터 저장, buf1의 데이터를 처리
 1: buf1에 입력 데이터 저장, buf0의 데이터를 처리

if the last pixel of a frame is store, change mode and start



Buffer Controller



```

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        mem_wr <= 1'b0;
        wr_data <= 8'b0;
    end else begin
        mem_wr <= i_strb;
        wr_data <= i_data;
    end
end
wire [15:0] wr_addr = cnt_y*256 + cnt_x;

wire      cs0 = (mode == 1'b0) ? mem_wr : mem_rd;
wire      we0 = (mode == 1'b0) ? mem_wr : 1'b0;
wire [15:0] addr0 = (mode == 1'b0) ? wr_addr : rd_addr;
wire [7:0] din0 = (mode == 1'b0) ? wr_data : 'b0;
wire [7:0] dout0;
wire      cs1 = (mode == 1'b1) ? mem_wr : mem_rd;
wire      we1 = (mode == 1'b1) ? mem_wr : 1'b0;
wire [15:0] addr1 = (mode == 1'b1) ? wr_addr : rd_addr;
wire [7:0] din1 = (mode == 1'b1) ? wr_data : 'b0;
wire [7:0] dout1;

assign rd_data = (mode == 1'b0) ? dout1 : dout0;

```

memory write at the next cycle of the input strobe

buf0 signals
 if mode is 0,
 data write
 if mode is 1,
 data processing

buf1 signals
 if mode is 1,
 data write
 if mode is 0,
 data processing



Buffer Controller



```
mem_single #(
    .WD(8),
    .DEPTH(256*256)
) i_buf0 (
    .clk(clk),
    .cs(cs0),
    .we(we0),
    .addr(addr0),
    .din(din0),
    .dout(dout0)
);
mem_single #(
    .WD(8),
    .DEPTH(256*256)
) i_buf1 (
    .clk(clk),
    .cs(cs1),
    .we(we1),
    .addr(addr1),
    .din(din1),
    .dout(dout1)
);
endmodule
```

Now, SRAM is in our design.



SRAM Implementation



- ▶ Note
 - ▶ SRAM is not the target of synthesis.
 - ▶ Memory compiler is provided by the fab company.
- ▶ In RTL simulation
 - ▶ Use a simulation model
- ▶ In Synthesis and P&R
 - ▶ Use the db generated by the memory compiler

```
module mem_single #(
    WD = 128
    , DEPTH = 64
    , WA = $clog2(DEPTH)
) (
    input          clk
    , input        cs
    , input        we
    , input        [WA-1:0] addr
    , input        [WD-1:0] din
    , output       [WD-1:0] dout
);
reg [WD-1:0] data[DEPTH-1:0];
reg [WA-1:0] addr_d;

always@(posedge clk) begin
    if(cs == 1'b1) begin
        if(we == 1'b1) data[addr] <= din;
        addr_d <= addr;
    end
end
assign dout = data[addr_d];
endmodule
```



Operation Block



```

module filter2d_op (
    input      clk,
    input      n_reset,
    input      start, -----> start signal

    output      mem_rd,
    output reg [15:0] rd_addr, } buffer read
    input      [7:0] rd_data,

    output reg [7:0] o_strb, } output data with strobe
    output reg [7:0] o_data  } -----> coefficient write
    ...

);
...
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        o_strb <= 1'b0;
        o_data <= 'b0;
    end else begin
        o_strb <= (cnt == 11);
        if(cnt == 11) o_data <= pd_out; } write cycle is substituted
                                         with data output.
    end
end
endmodule

```

Hyeon-Ju Kang

63



Integration



```

module filter2d (
    input      clk,
    input      n_reset,
    input      i_strb,
    input [7:0] i_data,
    output [7:0] o_strb,
    output [7:0] o_data

);
    wire      start;
    wire      mem_rd;
    wire [15:0] rd_addr;
    wire [7:0] rd_data;

    filter2d_buf i_buf(
        .clk(clk),
        .n_reset(n_reset),
        .i_strb(i_strb),
        .i_data(i_data),

        .start(start),

        .mem_rd(mem_rd),
        .rd_addr(rd_addr),
        .rd_data(rd_data)
    );

    filter2d_op i_op(
        .clk(clk),
        .n_reset(n_reset),
        .start(start),

        .mem_rd(mem_rd),
        .rd_addr(rd_addr),
        .rd_data(rd_data),

        .o_strb(o_strb),
        .o_data(o_data)
    );
endmodule

```

Hyeon-Ju Kang

64



Testbench



```

module top_filter_2d;

reg    clk, n_reset;
reg    start;

initial clk = 1'b0;
always #5 clk = ~clk;

reg[7:0] img_data[0:65535]; -----> Just for simulation
reg      i_strb;
reg[7:0] i_data;
integer  idx, cnt;
initial begin
    cnt = 0;
    n_reset = 1'b1;
    $readmemh("../c/img_in.dat", img_data); -----> Image data load
    i_strb = 1'b0;
    i_data = 'bx; } no data
    #3;
    n_reset = 1'b0;
    #20;
    n_reset = 1'b1;
end

```



Testbench



```

n_reset = 1'b1;
@(posedge clk);
@(posedge clk);
@(posedge clk);
repeat(3) begin
    for(idx=0;idx<65536;idx=idx+1) begin
        i_strb = 1'b1;
        i_data = img_data[idx];
        @(posedge clk);
        repeat(16) begin
            i_strb = 1'b0;
            i_data = 'bx;
            @(posedge clk);
        end
    end
end
@(posedge clk);
@(posedge clk);
@(posedge clk);
$finish;
end

```

with 16 cycle interval

input data of a frame

3 frames

제대로 된 검증을 위해서는 각 프레임에 대해 서로 다른 이미지 데이터를 사용해야 함



Testbench



```
wire      o_strb;
wire [7:0] o_data;
filter2d i_filter (
    .clk(clk),
    .n_reset(n_reset),
    .i_strb(i_strb),
    .i_data(i_data),
    .o_strb(o_strb),
    .o_data(o_data),
    .h_write(1'b0),
    .h_idx('b0),
    .h_data('b0)
);

always@(posedge clk) begin
    if(o_strb == 1'b1) begin
        $write("%3d ", o_data);
        cnt = cnt + 1;
        if(cnt[7:0] == 0) begin
            $write("\n");
        end
    end
end
```

} Print output data

} Just for printing format

Hyeon-Ju Kang

67



Outline

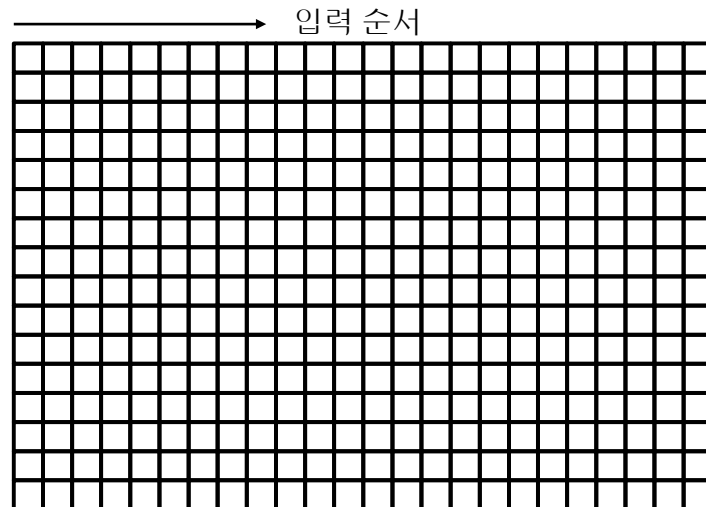


- ▶ Convolution on Image
- ▶ Fixed Point C 구현
- ▶ Verilog 구현
- ▶ Double Buffering
- ▶ Line Buffer 구조
- ▶ Throughput Optimization
- ▶ Direct Programming Interface
- ▶ Reusable Design

Hyeon-Ju Kang

68

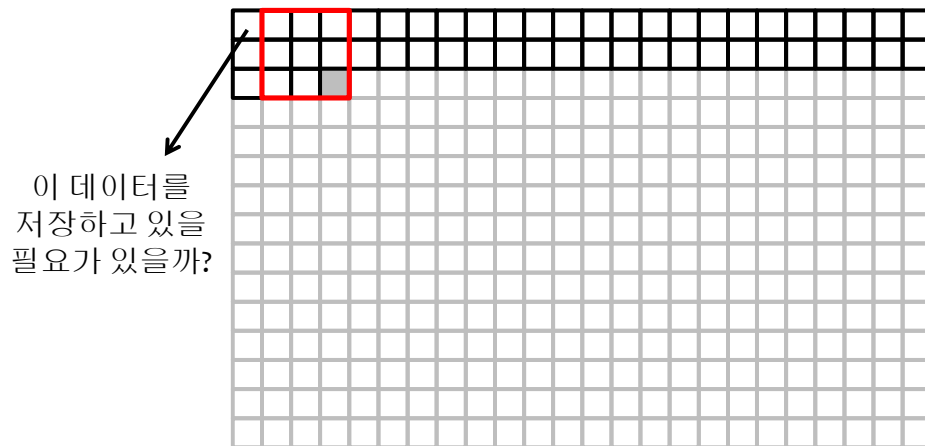
- ▶ 입력 데이터가 하나 씩 차례로 들어옴
 - ▶ 들어오는 중에도 처리할 수 있으면 처리하자.



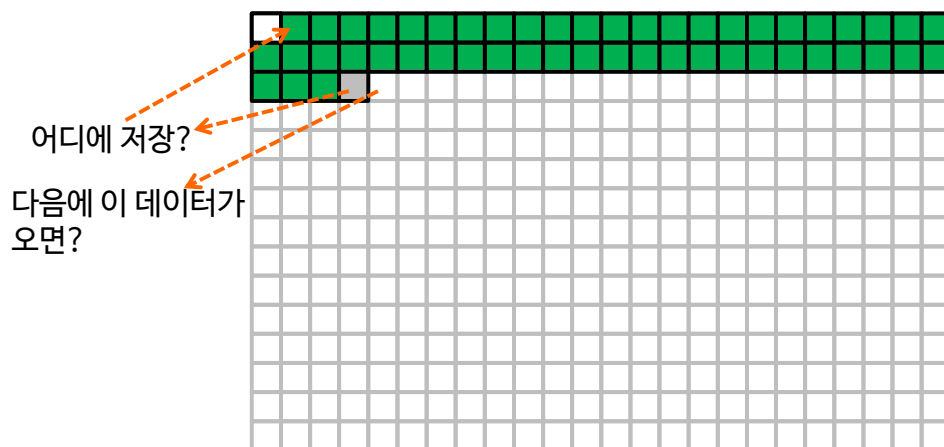
- ▶ 입력 데이터가 하나 씩 차례로 들어옴
 - ▶ 들어오는 중에도 처리할 수 있으면 처리하자.



- ▶ 입력 데이터가 하나 씩 차례로 들어옴
 - ▶ 들어오는 중에도 처리할 수 있으면 처리하자.



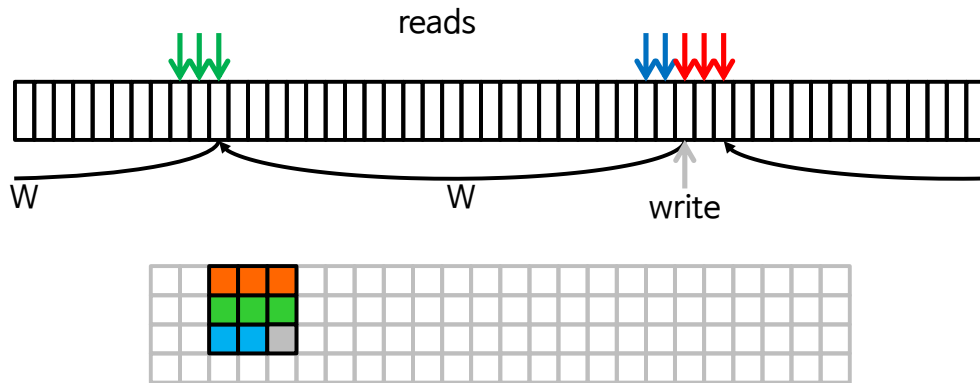
- ▶ 입력 데이터가 하나 씩 차례로 들어옴
 - ▶ 들어오는 중에도 처리할 수 있으면 처리하자.



■ 현재 가지고 있어야 하는 데이터 = ~2 line

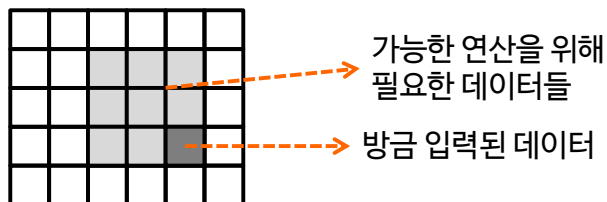


- ▶ 일종의 circular buffer임
- ▶ 차이점
 - ▶ read 위치와 write 위치가 일정한 관계가 있음

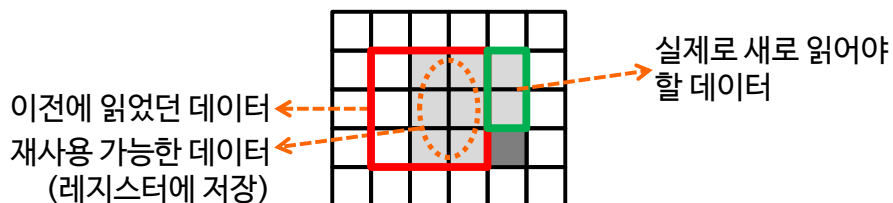


$$\text{최소 버퍼 길이} = 2 * W + 2$$

- ▶ 하나의 데이터가 들어왔을 때 메모리를 몇 번 읽어야 할까?

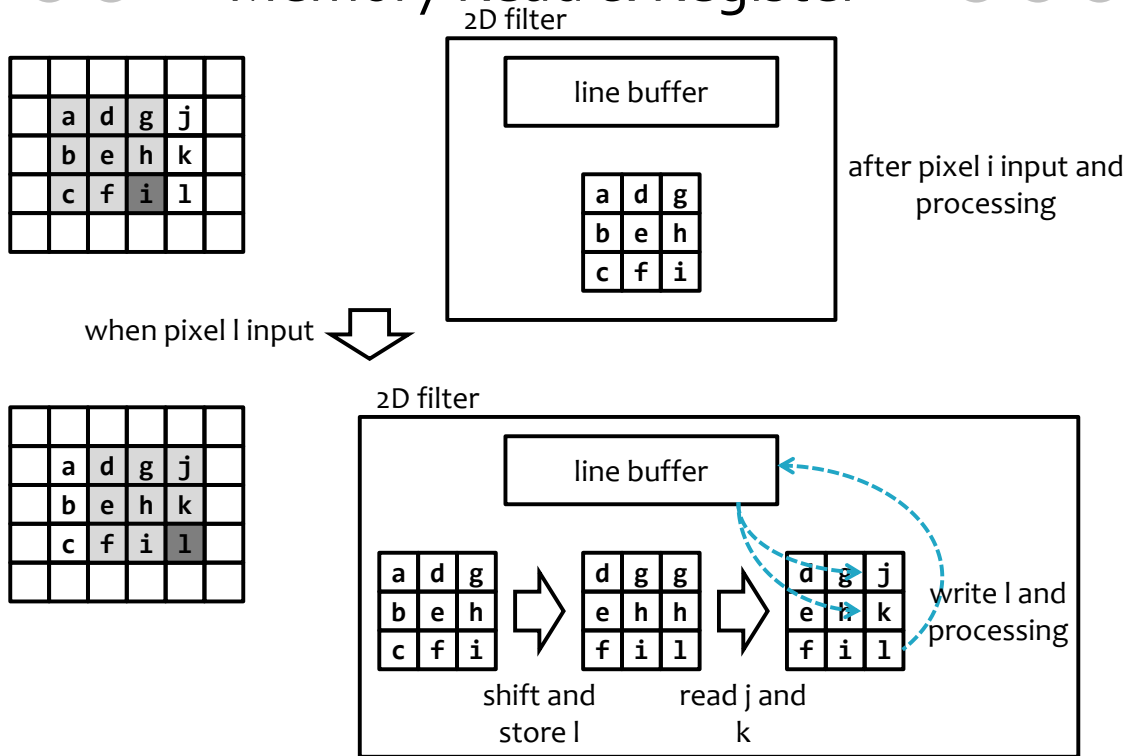


- ▶ 그런데

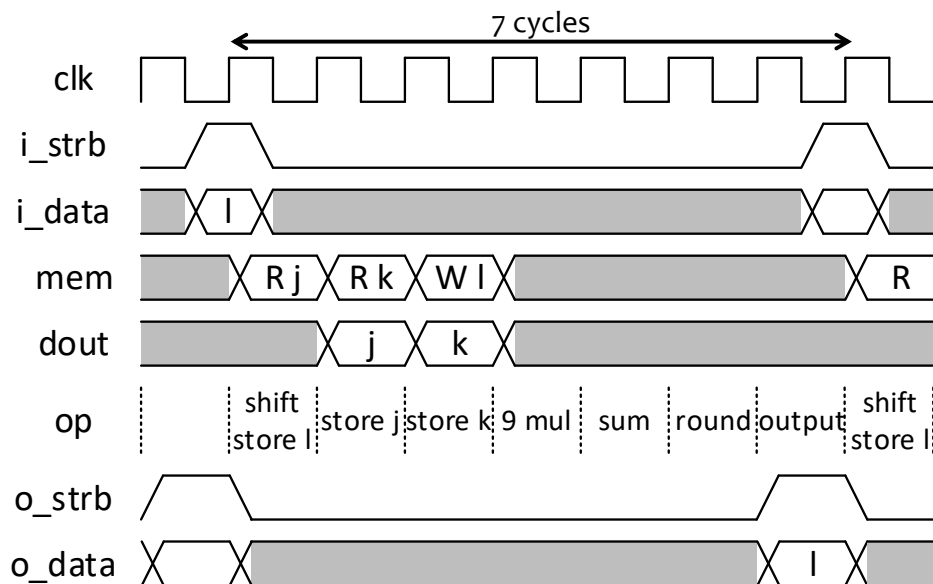




Memory Read & Register



Timing Diagram





Verilog Code



```

module filter2d (
    input      clk,
    input      n_reset,
    input      i_strb,
    input [7:0] i_data,
    output reg  o_strb,
    output reg  [7:0] o_data,
    input      h_write,
    input [3:0] h_idx,
    input [7:0] h_data
);

reg      garbage; -----> 나중에 설명
reg[3:0] cnt;             -----> counting b.w. input strobes
reg[7:0] cnt_x;          } counting the coordinates of the current output data
reg[7:0] cnt_y;          }
reg[7:0] i_data_d;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        garbage <= 1'b1;
        cnt <= 7;
    end
end

```



Verilog Code



```

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        garbage <= 1'b1;
        cnt <= 7;
        cnt_x <= 254; } Not 0?
        cnt_y <= 254; }
        i_data_d <= 'b0;
    end else begin
        if(i_strb == 1'b1) begin
            cnt_x <= (cnt_x == 255) ? 0 : cnt_x+1;
            if(cnt_x == 255) begin
                cnt_y <= (cnt_y == 255) ? 0 : cnt_y+1;
                if(cnt_y == 255) garbage <= 1'b0;
            end
        end
        if(i_strb == 1'b1) cnt <= 0; } counting x and y
        else if(cnt < 7) cnt <= cnt+1; } counting b.w. input strobes
                                         but 0~7?

        if(i_strb == 1'b1) i_data_d <= i_data; -----> Store the input data
    end
end

```



Verilog Code



```

reg [7:0] ibuf[2:0][2:0]; -----> 9 registers for pixel data
wire [7:0] dout; -----> memory output data

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        for(int i=0;i<3;i++) begin
            for(int j=0;j<3;j++) begin
                ibuf[i][j] <= 'b0;
            end
        end
    end else begin
        if(cnt == 0) begin -----> at the cycle 0
            for(int i=0;i<3;i++) begin
                for(int j=0;j<2;j++) begin
                    ibuf[i][j] <= ibuf[i][j+1];
                end
            end
            ibuf[2][2] <= i_data_d; -----> Store the input data
        end
        if(cnt == 1) ibuf[0][2] <= dout;
        if(cnt == 2) ibuf[1][2] <= dout; -----> store the data from memory
    end
end
end

```

a	d	g
b	e	h
c	f	i



Verilog Code



```

reg [7:0] ibuf[2:0][2:0]; -----> 9 registers for pixel data
wire [7:0] dout; -----> memory output data

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        for(int i=0;i<3;i++) begin
            for(int j=0;j<3;j++) begin
                ibuf[i][j] <= 'b0;
            end
        end
    end else begin
        if(cnt == 0) begin -----> at the cycle 0
            for(int i=0;i<3;i++) begin
                for(int j=0;j<2;j++) begin
                    ibuf[i][j] <= ibuf[i][j+1];
                end
            end
            ibuf[2][2] <= i_data_d; -----> Store the input data
        end
        if(cnt == 1) ibuf[0][2] <= dout;
        if(cnt == 2) ibuf[1][2] <= dout; -----> store the data from memory
    end
end
end

```

a	d	g
b	e	h
c	f	i

a	d	g
b	e	h
c	f	i

shift and store l

d	g	g
e	h	h
f	i	l

read j and k

d	g	j
e	h	k
f	i	l

write l and processing


```

wire      mem_rd = (cnt == 0) || (cnt == 1); } memory read/write
wire      mem_wr = (cnt == 2);

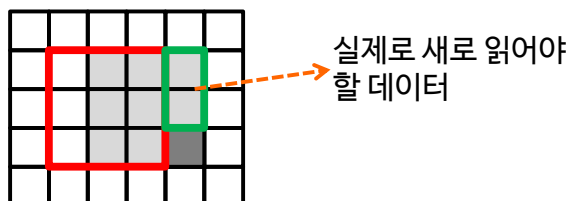
reg       [8:0] wr_addr;
wire      [8:0] rd_addr0 = wr_addr;
wire      [8:0] rd_addr1 = (wr_addr < 256) ? wr_addr+256 : wr_addr-256; } read
wire      [8:0] rd_addr = (cnt == 0) ? rd_addr0 : rd_addr1; } address?

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        wr_addr <= 1'b0;
    end else begin
        if(mem_wr == 1'b1) wr_addr <= wr_addr + 1;
    end
end

wire      cs = mem_rd | mem_wr;
wire      we = mem_wr;
wire      [8:0] addr = (mem_wr == 1'b1) ? wr_addr : rd_addr;
wire      [7:0] din = i_data_d;

```

- ▶ 두 번의 읽기
 - ▶ 새로 들어온 데이터의 바로 위 두 개 데이터



$\text{read address 0} = \text{write address} - 2W$
 $\text{read address 1} = \text{write address} - W$
 + line buffer length if negative

```

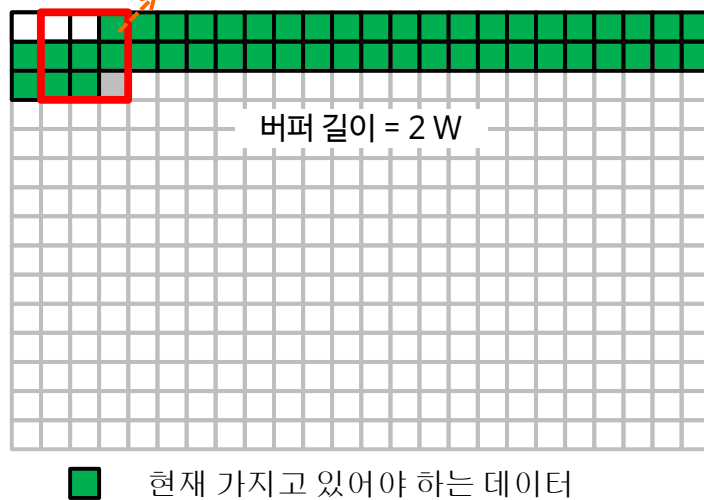
rd_addr0 = (wr_addr < 2*W) ? (wr_addr-2*W+BUF_LEN) : wr_addr-2*W;
rd_addr1 = (wr_addr < W) ? (wr_addr-W+BUF_LEN) : wr_addr-W;

```

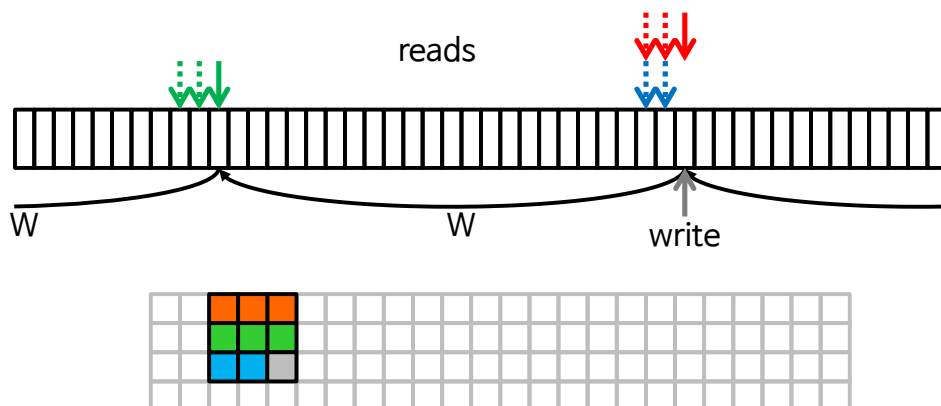
▶ 두 번의 읽기

- ▶ 새로 들어온 데이터의 바로 위 두 개 데이터

새로 들어온 데이터는 이 데이터가 있던 자리에 저장하면 됨



▶ read 위치와 write 위치가 일정한 관계가 있음





Read Address



▶ 두 번의 읽기

- ▶ 새로 들어온 데이터의 바로 위 두 개 데이터

```
rd_addr0 = (wr_addr < 2*W) ? (wr_addr-2*W+BUF_LEN) : wr_addr-2*W;
```

언제나 true
BUF_LEN=2*W

wr_addr < BUF_LEN=2*W



```
rd_addr0 = wr_addr;
```

```
rd_addr1 = (wr_addr < W) ? (wr_addr- W+BUF_LEN) : wr_addr- W;
```

언제나 true
BUF_LEN=2*W

wr_addr < W



```
rd_addr1 = (wr_addr < W) ? wr_addr+W : wr_addr-W;
```



85



Verilog Code



```
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        wr_addr <= 0;
    end else begin
        if(mem_wr == 1'b1) wr_addr <= (wr_addr == 256*2-1) ? 0 : wr_addr + 1;
        end
    end
end

wire      cs = mem_rd | mem_wr;
wire      we = mem_wr;
wire [8:0] addr = (mem_wr == 1'b1) ? wr_addr : rd_addr;
wire [7:0] din = i_data_d;
mem_single #(
    .WD(8),
    .DEPTH(2*256)
) i_buf0 (
    .clk(clk),
    .cs(cs),
    .we(we),
    .addr(addr),
    .din(din),
    .dout(dout)
);
```

→ Increase write address at each write



86



Verilog Code



```

reg signed [15:0] mul[2:0][2:0];
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        for(int i=0;i<3;i++) begin
            for(int j=0;j<3;j++) begin
                mul[i][j] <= 'b0;
            end
        end
    end else begin
        if((cnt == 3) && (garbage == 1'b0)) begin
            mul[0][0] <= ((cnt_y > 0) && (cnt_x > 0)) ? ibuf[0][0]*h[0] : 'b0;
            mul[0][1] <= ((cnt_y > 0) ) ? ibuf[0][1]*h[1] : 'b0;
            mul[0][2] <= ((cnt_y > 0) && (cnt_x < 255)) ? ibuf[0][2]*h[2] : 'b0;
            mul[1][0] <= (cnt_x > 0) ? ibuf[1][0] * h[3] : 'b0;
            mul[1][1] <= ibuf[1][1] * h[4];
            mul[1][2] <= (cnt_x < 255) ? ibuf[1][2] * h[5] : 'b0;
            mul[2][0] <= ((cnt_y < 255) && (cnt_x > 0)) ? ibuf[2][0]*h[6] : 'b0;
            mul[2][1] <= ((cnt_y < 255) ) ? ibuf[2][1]*h[7] : 'b0;
            mul[2][2] <= ((cnt_y < 255) && (cnt_x < 255)) ? ibuf[2][2]*h[8] : 'b0;
        end
    end
end

```



Verilog Code



```

reg signed [19:0] sum_in;
reg signed [19:0] sum;
always@(*) begin
    sum_in = 0;
    for(int i=0;i<3;i++) begin
        for(int j=0;j<3;j++) begin
            sum_in = sum_in + mul[i][j];
        end
    end
end
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        sum <= 'b0;
    end else begin
        if((cnt == 4) && (garbage == 1'b0)) begin
            sum <= sum_in;
        end
    end
end

```



Verilog Code



```

wire [19:0] pd_rnd_1 = sum + (1<<6);
wire [12:0] pd_rnd = pd_rnd_1[19:7];
wire [7:0] pd_out = (pd_rnd < 0) ? 0 :
                    (pd_rnd > 255) ? 255 :
                    pd_rnd[7:0];

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        o_strb <= 1'b0;
        o_data <= 'b0;
    end else begin
        o_strb <= ((cnt == 5) && (garbage == 1'b0));
        if((cnt == 5) && (garbage == 1'b0)) begin
            o_data <= pd_out;
        end
    end
end
endmodule

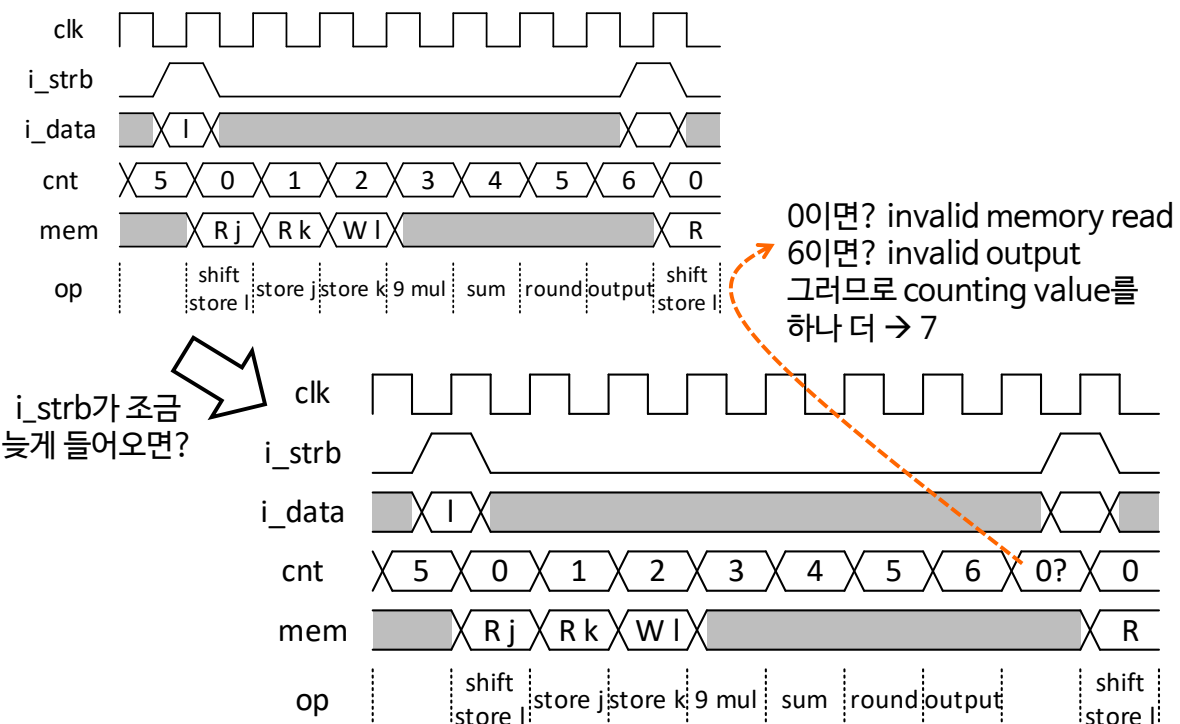
```

Hyeong-Ju Kang

89



Counting



Hyeong-Ju Kang

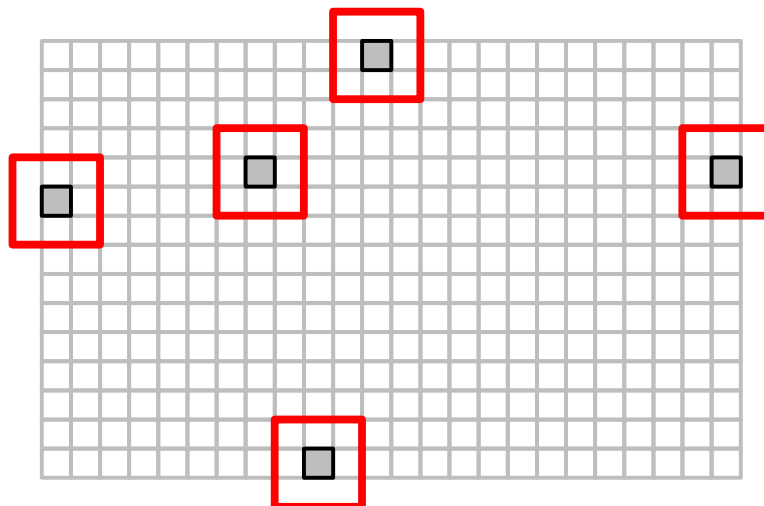
90

```
always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        garbage <= 1'b1;
        cnt <= 7;
        cnt_x <= 254;
        cnt_y <= 254;
        i_data_d <= 'b0;
    end else begin
        if(i_strb == 1'b1) begin
            cnt_x <= (cnt_x == 255) ? 0 : cnt_x+1;
            if(cnt_x == 255) begin
                cnt_y <= (cnt_y == 255) ? 0 : cnt_y+1;
                if(cnt_y == 255) garbage <= 1'b0;
            end
        end

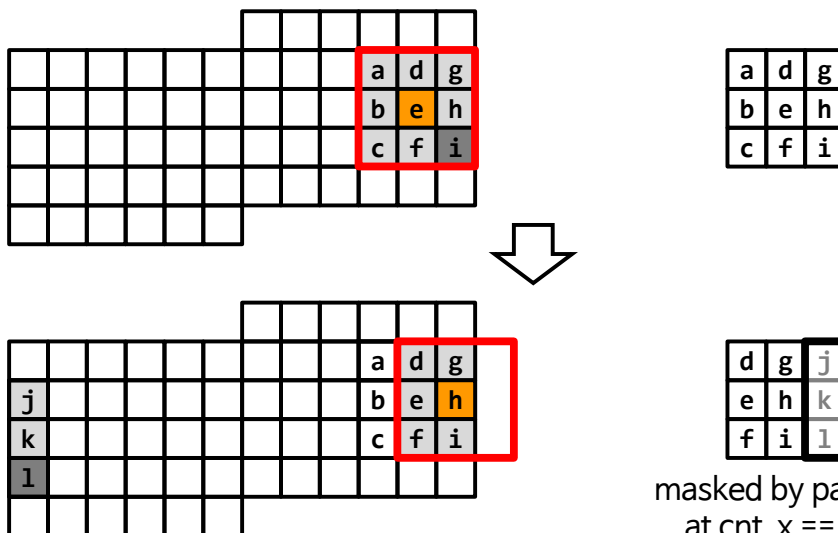
        if(i_strb == 1'b1) cnt <= 0;
        else if(cnt < 7) cnt <= cnt+1;

        if(i_strb == 1'b1) i_data_d <= i_data;
    end
end
```

► With padding



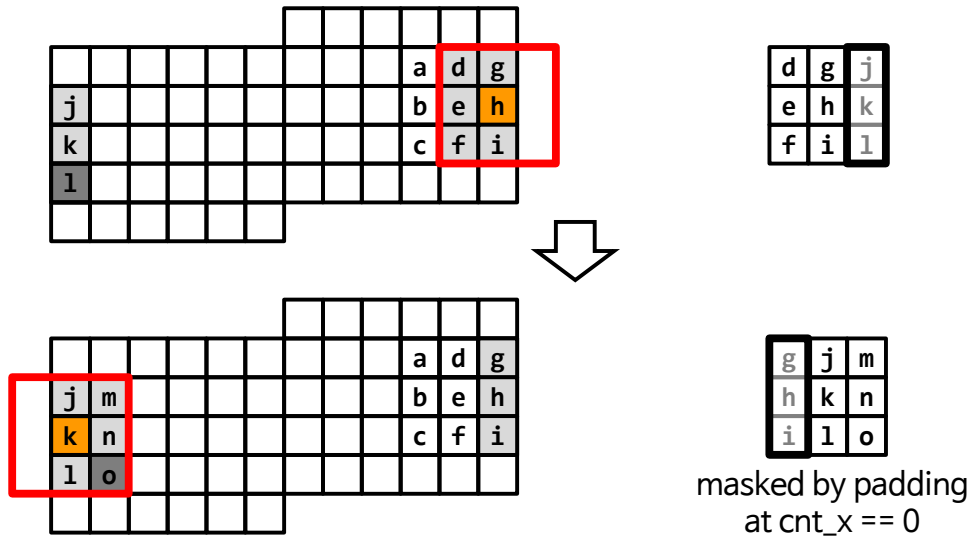
- ▶ 우선 일반적인 경우의 동작을 그대로 적용할 수 있는지 검토!!



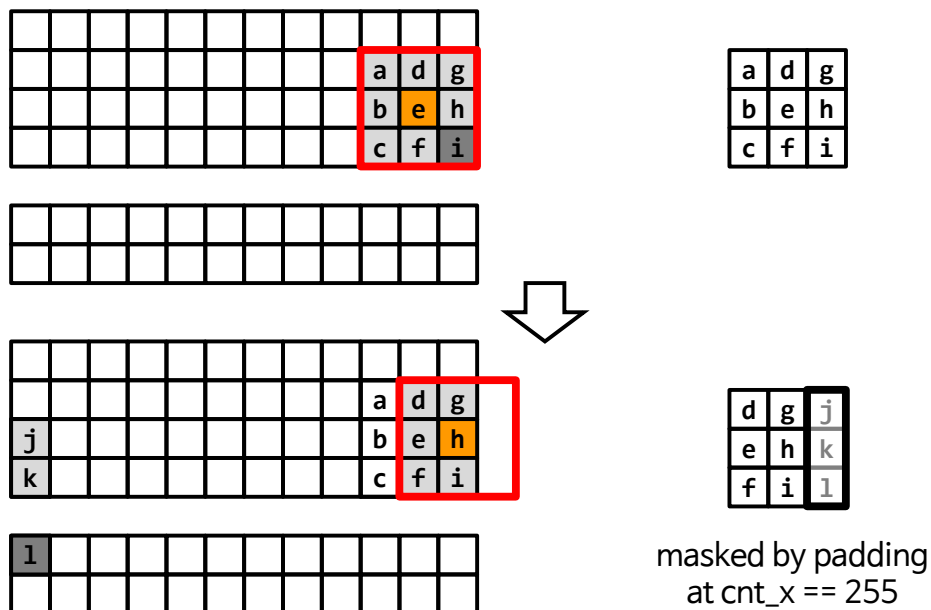
masked by padding
at `cnt_x == 255`

주의: `cnt_x`와 `cnt_y`는 출력 데이터 기준
(여기서는 `h`의 위치)

```
mul[0][2] <= ((cnt_y > 0) && (cnt_x < 255)) ? ibuf[0][2]*h[2] : 'b0';
mul[1][2] <= (cnt_x < 255) ? ibuf[1][2] * h[5] : 'b0';
mul[2][2] <= ((cnt_y < 255) && (cnt_x < 255)) ? ibuf[2][2]*h[8] : 'b0';
```



```
mul[0][0] <= ((cnt_y > 0) && (cnt_x > 0)) ? ibuf[0][0]*h[0] : 'b0;  
mul[1][0] <= (cnt_x > 0) ? ibuf[1][0] * h[3] : 'b0;  
mul[2][0] <= ((cnt_y < 255) && (cnt_x > 0)) ? ibuf[2][0]*h[6] : 'b0;
```



```
mul[0][2] <= ((cnt_y > 0) && (cnt_x < 255)) ? ibuf[0][2]*h[2] : 'b0';
mul[1][2] <= (cnt_x < 255) ? ibuf[1][2] * h[5] : 'b0';
mul[2][2] <= ((cnt_y < 255) && (cnt_x < 255)) ? ibuf[2][2]*h[8] : 'b0';
```



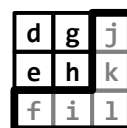
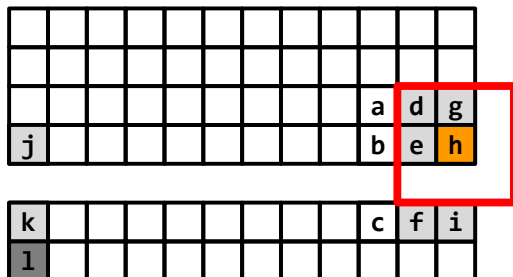
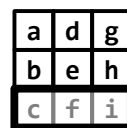

masked by padding
at cnt_x == 0 or cnt_y == 255

97

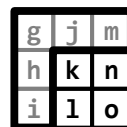
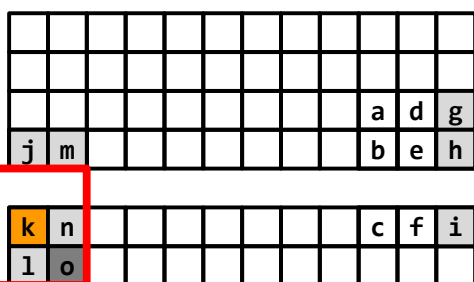


masked by padding
at cnt_y == 255

98



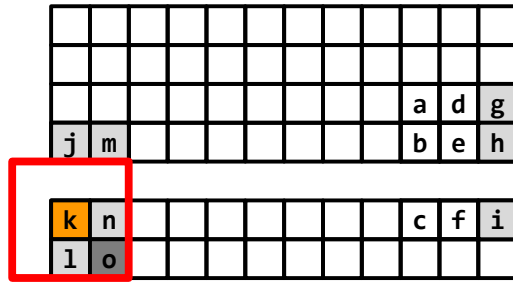
```
mul[0][2] <= ((cnt_y > 0) && (cnt_x < 255)) ? ibuf[0][2]*h[2] : 'b0;
mul[1][2] <= (cnt_x < 255) ? ibuf[1][2] * h[5] : 'b0;
mul[2][0] <= ((cnt_y < 255) && (cnt_x > 0)) ? ibuf[2][0]*h[6] : 'b0;
mul[2][1] <= ((cnt_y < 255) ) ? ibuf[2][1]*h[7] : 'b0;
mul[2][2] <= ((cnt_y < 255) && (cnt_x < 255)) ? ibuf[2][2]*h[8] : 'b0;
```



```
mul[0][0] <= ((cnt_y > 0) && (cnt_x > 0)) ? ibuf[0][0]*h[0] : 'b0';
mul[0][1] <= ((cnt_y > 0) ) ? ibuf[0][1]*h[1] : 'b0';
mul[0][2] <= ((cnt_y > 0) && (cnt_x < 255)) ? ibuf[0][2]*h[2] : 'b0';
mul[1][0] <= (cnt_x > 0) ? ibuf[1][0] * h[3] : 'b0';
mul[2][0] <= ((cnt_y < 255) && (cnt_x > 0)) ? ibuf[2][0]*h[6] : 'b0';
```

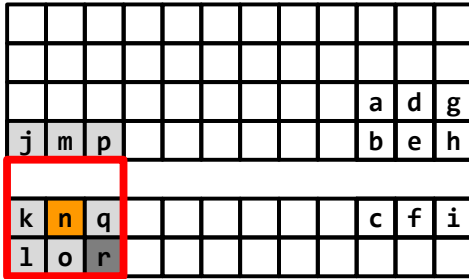


At the first line



g	j	m
h	k	n
i	l	o

masked by padding
at cnt_x == 0 or cnt_y == 0



j	m	p
k	n	q
l	o	r

masked by padding
at cnt_y == 0

```
mul[0][0] <= ((cnt_y > 0) && (cnt_x > 0)) ? ibuf[0][0]*h[0] : 'b0';
mul[0][1] <= ((cnt_y > 0) ) ? ibuf[0][1]*h[1] : 'b0';
mul[0][2] <= ((cnt_y > 0) && (cnt_x < 255)) ? ibuf[0][2]*h[2] : 'b0';
```

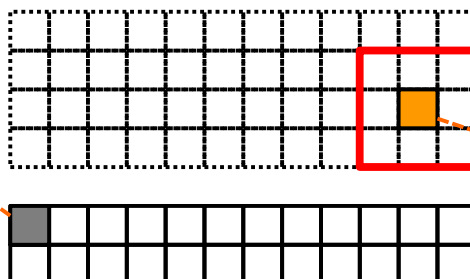


Initialization



```
always@(posedge clk or negedge n_reset) begin
  if(n_reset == 1'b0) begin
    garbage <= 1'b1;
    cnt <= 0;
    cnt_x <= 254;
    cnt_y <= 254;
    i_data_d <= 'b0;
  end else begin
    ...
  end
end
```

이 픽셀이 들어올 때는



이 부분의 연산이 끝난
상태가 자연스러움

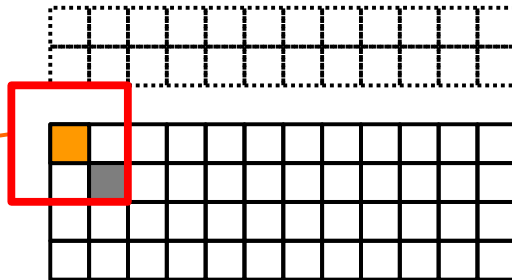
(254,254)

```

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        garbage <= 1'b1;
        ...
    end else begin
        if(i_strb == 1'b1) begin
            cnt_x <= (cnt_x == 255) ? 0 : cnt_x+1;
            if(cnt_x == 255) begin
                cnt_y <= (cnt_y == 255) ? 0 : cnt_y+1;
                if(cnt_y == 255) garbage <= 1'b0;
            end
        end
    end
    ...
end
end

```

처음 정상 출력 값 계산



(255,255)까지는
계속 garbage 출력

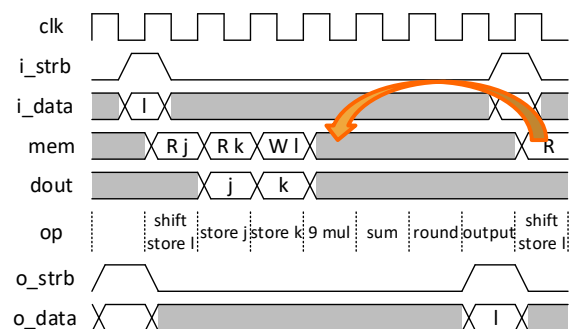
```

    if(i_strb == 1'b1) begin
        cnt_x <= (cnt_x == 255) ? 0 : cnt_x+1;
        if(cnt_x == 255) begin
            cnt_y <= (cnt_y == 255) ? 0 : cnt_y+1;
            if(cnt_y == 255) garbage <= 1'b0;
        end
    end
    ...
    if((cnt == 3) && (garbage == 1'b0)) begin
        mul[0][0] <= ((cnt_y > 0) && (cnt_x > 0)) ? ibuf[0][0]*h[0] : 'b0;
        ...
    end
    ...
    if((cnt == 4) && (garbage == 1'b0)) begin
        sum <= sum_in;
    end
    ...
    o_strb <= ((cnt == 5) && (garbage == 1'b0));
    if((cnt == 5) && (garbage == 1'b0)) begin
        o_data <= pd_out;
    end
    ...

```

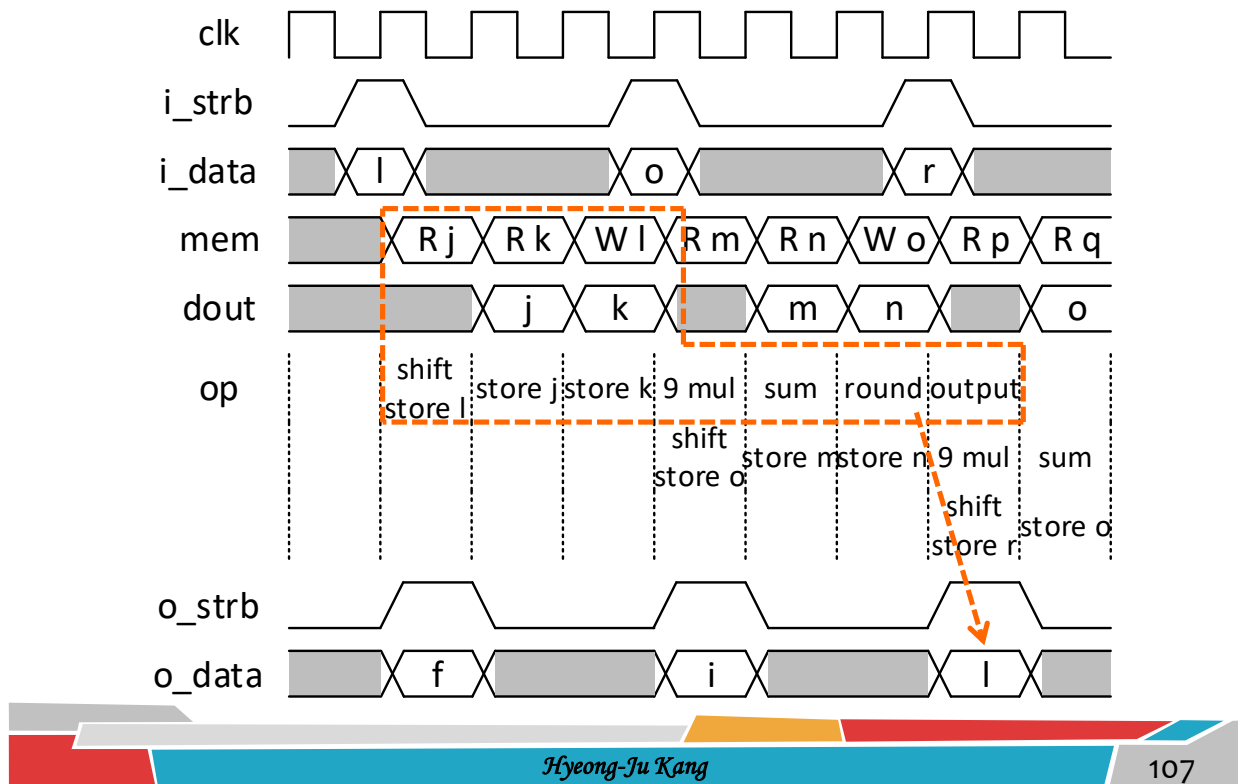
- ▶ Convolution on Image
- ▶ Fixed Point C 구현
- ▶ Verilog 구현
- ▶ Double Buffering
- ▶ Throughput Optimization
- ▶ Direct Programming Interface
- ▶ Reusable Design

- ▶ Throughput of the current design: 7 cycles/pixel
- ▶ But,
 - ▶ On the view of operators
 - ▶ 9 multiplications per pixel and 9 multipliers
 - ▶ 1 cycle/pixel is possible
 - ▶ On the view of memory
 - ▶ 2 reads and 1 write per pixel
 - ▶ 3 cycles/pixel is possible
- ▶ 3 cycles/pixel !!

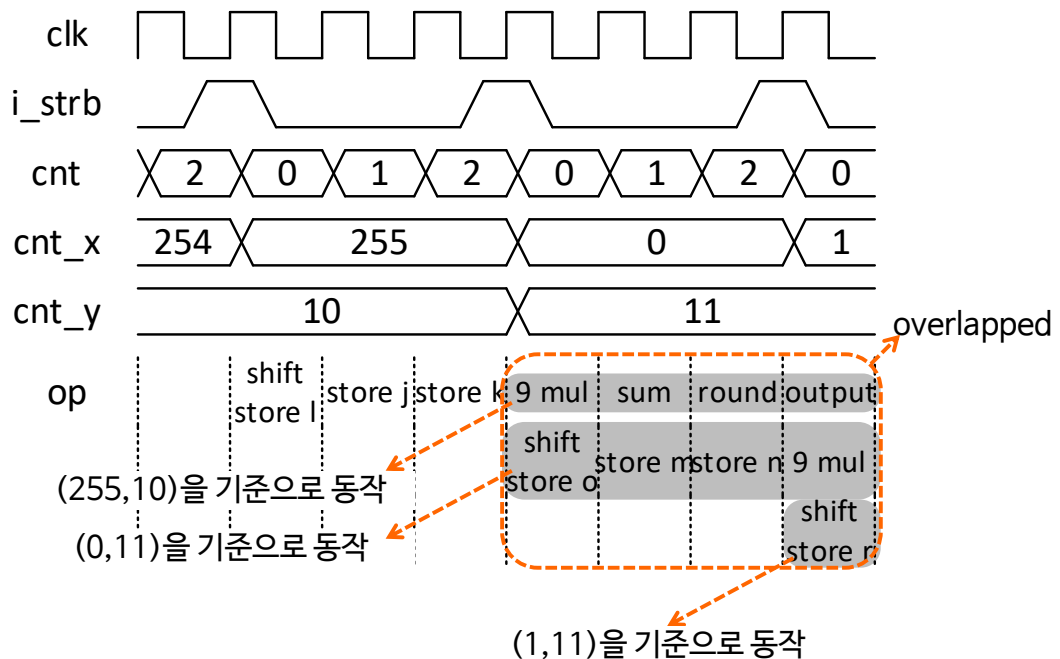




Higher Throughput



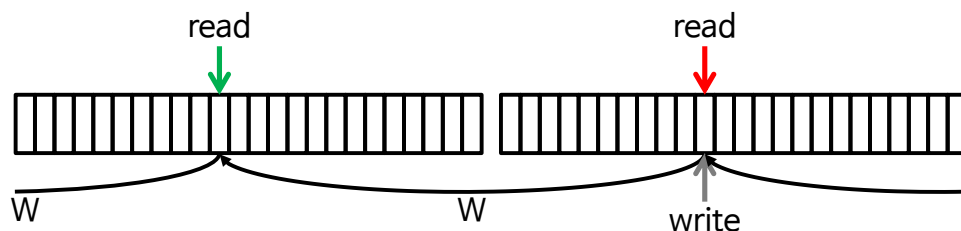
Counting Problem



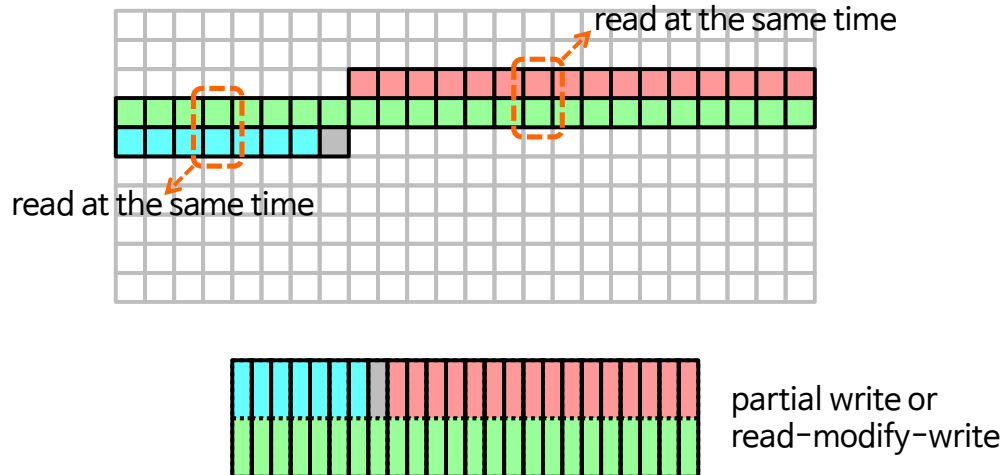
- ▶ Operator reduction
 - ▶ If throughput is 3 cycles/pixel, only 3 multipliers are sufficient.
- ▶ Further higher throughput
 - ▶ 3 cycles/pixel because of memory bandwidth
 - ▶ 2 reads and 1 write
 - ▶ How to get more bandwidth
 - ▶ double-port SRAM
 - ▶ memory partitioning
 - ▶ data pairing



- ▶ Partition the line buffer into two buffers
 - ▶ Two read addresses are always separated by W .
 - ▶ Two reads are always performed on the different buffers.
 - read at the same time



- ▶ Pair the two data that will be read at the same time
- ▶ Store the pair in the same entry



- ▶ Convolution on Image
- ▶ Fixed Point C 구현
- ▶ Verilog 구현
- ▶ Double Buffering
- ▶ Line Buffer 구조
- ▶ Throughput Optimization
- ▶ Direct Programming Interface
- ▶ Reusable Design

○○○ Direct Programming Interface ○○○

▶ Direct Programming Interface (DPI)

- ▶ Verilog 실행 중에 C 언어 등으로 작성된 함수를 호출하는 인터페이스
- ▶ 이전의 Programming Language Interface (PLI)를 더 편리하게
- ▶ 주로 testbench를 만들 때 사용

▶ 여기서는

- ▶ 각 입력 데이터 값을 C 함수를 호출하여 얻음
- ▶ 각 출력 데이터 값을 C 함수의 출력값과 비교

○○○ Direct Programming Interface ○○○

▶ 여기서는

- ▶ 각 입력 데이터 값을 C 함수를 호출하여 얻음
- ▶ 각 출력 데이터 값을 C 함수의 출력값과 비교

```
$readmemh("../c/img_in.dat", img_data);  
...  
repeat(3) begin  
    for(idx=0;idx<65536;idx=idx+1) begin  
        i_strb = 1'b1;  
        i_data = img_data[idx];  
        @(posedge clk);  
    ...  
    if(o_strb == 1'b1) begin  
        $write("%3d ", o_data);  
        cnt = cnt + 1;  
        if(cnt[7:0] == 0) begin  
            $write("\n");  
        end  
    end  
end  
...
```

```
init_filter2d(256, 256);  
...  
repeat(3) begin  
    for(idx=0;idx<65536;idx=idx+1) begin  
        i_strb = 1'b1;  
        i_data = get_input();  
        @(posedge clk);  
    ...  
    if(o_strb == 1'b1) begin  
        out_ref = get_output();  
        if(o_data != out_ref) begin  
            $display("Error!!");  
            #10;  
            $finish;  
        end  
    end  
end  
...
```



DPI C Code



```
#include <stdio.h>
#include <stdlib.h>
unsigned char *in_img, *out_img;
int height, width;

void filter2d(void) {
    ...
}
void init_filter2d(int h, int w) {
    int i, a;
    FILE *inf;
    height = h; width = w;

    inf = fopen("../c/img_in.txt", "r");
    in_img = malloc(height*width*sizeof(unsigned char));
    out_img = malloc(height*width*sizeof(unsigned char));
    for(i=0;i<height*width;i++) {
        fscanf(inf, "%d,", &a);
        in_img[i] = a;
    }
    filter2d();
    fclose(inf);
```

출력 데이터를 미리 계산함
(입력 데이터를 생성하면서 계산할 수도 있음)



DPI C Code



```
unsigned char get_input(void) {
    static int i;
    unsigned char res = in_img[i];
    i++;
    if(i==height*width) i = 0;
    return res;
}

unsigned char get_output(void) {
    static int i;
    unsigned char res = out_img[i];
    i++;
    if(i==height*width) i = 0;
    return res;
}
```



DPI Declaration in Verilog



- ▶ DPI 함수를 호출하기 전에 선언해 주어야 함

```
...  
initial clk = 1'b0;  
always #5 clk = ~clk;  
  
import "DPI" function void init_filter2d(input int h, input int w);  
import "DPI" function byte get_input();  
import "DPI" function byte get_output();  
  
reg      i_strb;  
reg[7:0] i_data;
```



Compile & Run



- ▶ C 파일은 shared library로 컴파일
 - ▶ 시뮬레이션 실행할 때 library로 연결

```
user@hostname$ gcc -std=c99 -fPIC -shared -o filter_2d.so ../c/filter_2d_dpi.c  
user@hostname$ vcs -full64 -sverilog top3.v filter2d_line.v mem_sim.v  
user@hostname$ ./simv -sv_lib filter_2d
```



Automatic Check



- ▶ Testbench에서 출력값을 자동으로 검사
 - ▶ 에러가 발생하면 왜 에러가 발생했는지 간단히 출력하는 것이 좋음
 - ▶ Error 표시가 안 나온다고 정말로 문제가 없는 것일까?

```
reg    [7:0] out_ref;
always@(posedge clk) begin
    if(o_strb == 1'b1) begin -----> o_strb가 1이면?
        out_ref = get_output();
        if(o_data != out_ref) begin -----> o_data가 x이면?
            $display("Error!! o_data = %3d, out_ref = %3d", o_data, out_ref);
            #10;
            $finish;
        end
    end
end
end
```



Outline



- ▶ Convolution on Image
- ▶ Fixed Point C 구현
- ▶ Verilog 구현
- ▶ Double Buffering
- ▶ Line Buffer 구조
- ▶ Direct Programming Interface
- ▶ Throughput Optimization
- ▶ Reusable Design



Parameterization for Reusable Design



- ▶ 도입하기 쉬운 parameter: height & width
 - ▶ convolution 연산은 대상 이미지 크기에 영향을 받지 않음

```
module filter2d #(
    H = 256,
    W = 256
) (
    input      clk,
    input      n_reset,
```



Verilog Code



```
reg[$clog2(W)-1:0] cnt_x;          clog2: ceiling of log2
reg[$clog2(H)-1:0] cnt_y;          $clog2(W): bit width to represent 0~W-1

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        garbage <= 1'b1;
        cnt <= 7;
        cnt_x <= W-2;
        cnt_y <= H-2;
        i_data_d <= 'b0;
    end else begin
        if(i_strb == 1'b1) begin
            cnt_x <= (cnt_x == W-1) ? 0 : cnt_x+1;
            if(cnt_x == W-1) begin
                cnt_y <= (cnt_y == H-1) ? 0 : cnt_y+1;
                if(cnt_y == H-1) garbage <= 1'b0;
            end
        end
        if(i_strb == 1'b1) cnt <= 0;
        else if(cnt < 7) cnt <= cnt+1;
        if(i_strb == 1'b1) i_data_d <= i_data;
    end
end
```



Verilog Code



```
localparam BUF_LEN = 2*W;
reg    [$clog2(BUF_LEN)-1:0] wr_addr;
wire   [$clog2(BUF_LEN)-1:0] rd_addr0 = wr_addr;
wire   [$clog2(BUF_LEN)-1:0] rd_addr1 = (wr_addr<W) ? wr_addr+W: wr_addr-W;
wire   [$clog2(BUF_LEN)-1:0] rd_addr = (cnt == 0) ? rd_addr0 : rd_addr1;

always@(posedge clk or negedge n_reset) begin
    if(n_reset == 1'b0) begin
        wr_addr <= 0;
    end else begin
        if(mem_wr == 1'b1) begin
            wr_addr <= (wr_addr == BUF_LEN-1) ? 0 : wr_addr + 1;
        end
    end
end
...
mem_single #(
    .WD(8),
    .DEPTH(BUF_LEN)
) i_buf0 (
```



Parameterization for Reusable Design



- ▶ Other possible parameters
 - ▶ Bit width: input data, coefficients, output data
 - ▶ Convolution: K, PAD