차세대 반도체 디지털 혁신공유대학 사업단

POLARIS

System Integration I

Sliding Windows, CNN Controller

2022.02.23 (Thu)

Road map

Review

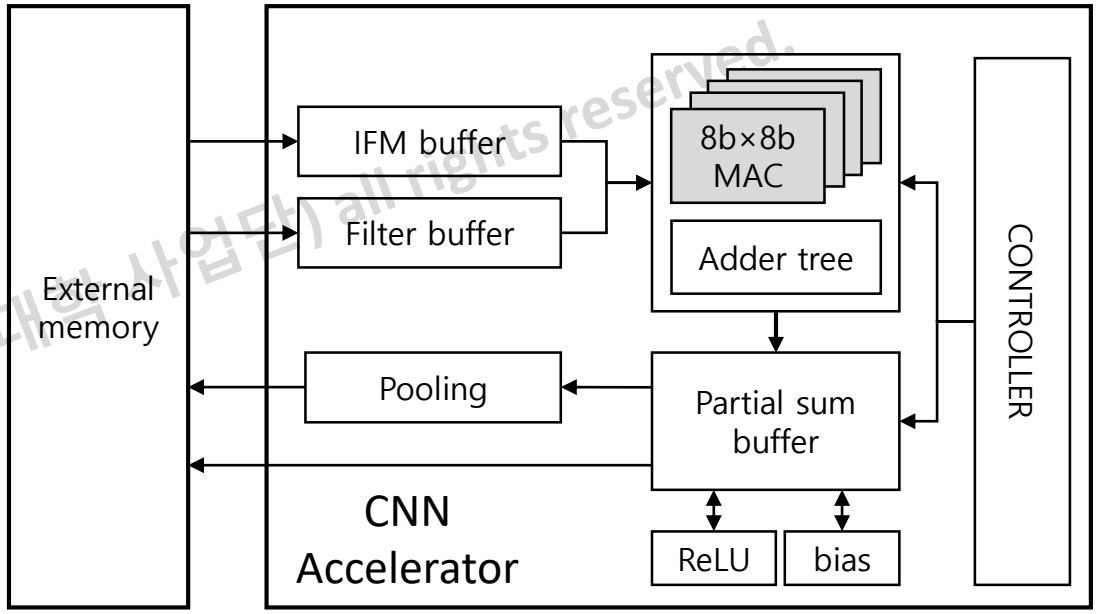
IO files

Controller

Sliding windows

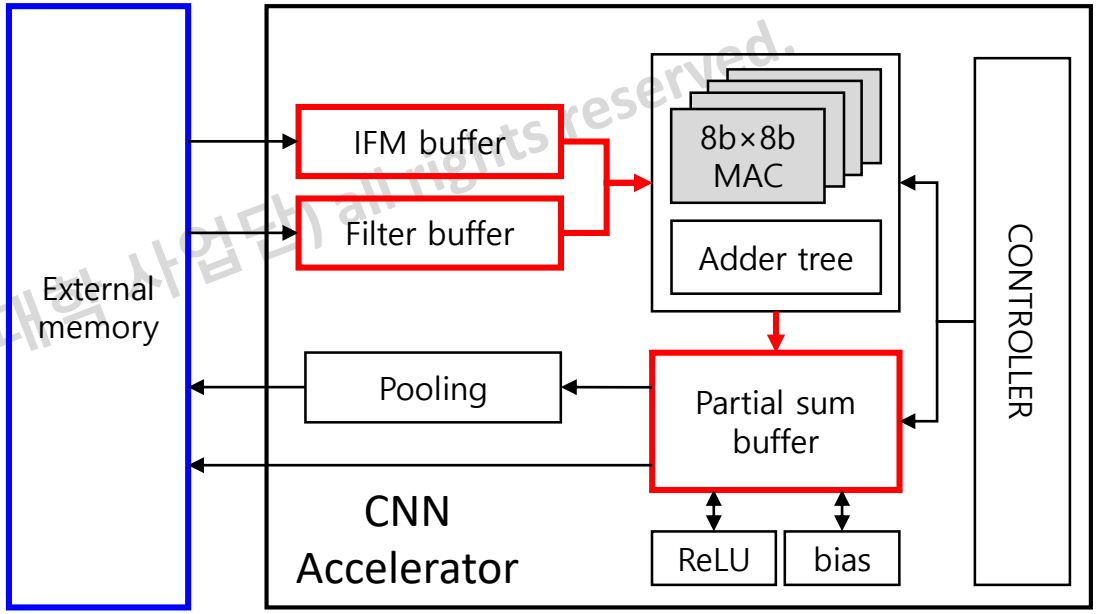
Inference engine

- Processing Element (PE): MAC, adder tree
- Buffers
  - Input feature map (IFM)
  - Filters
  - Intermediate results or partial sum
- Pooling
- Controller



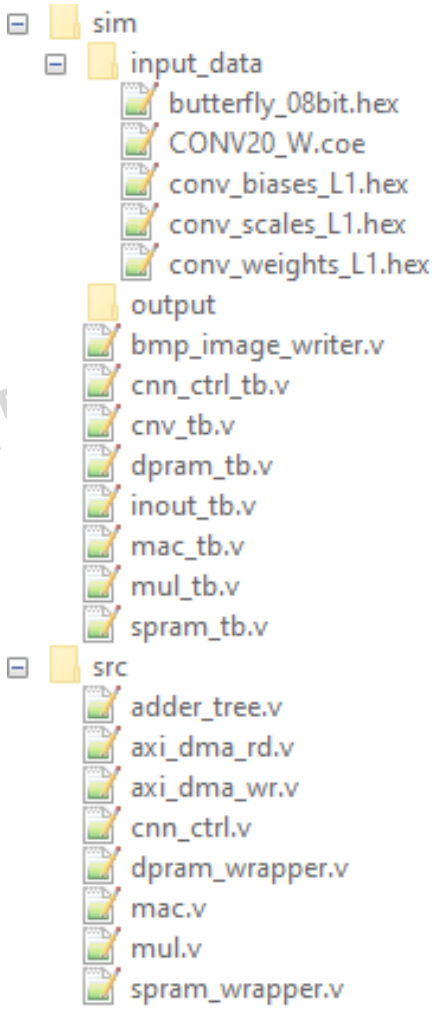
Buffers

- On-chip buffers (memory)
  - Input feature map (IFM)
  - Filters
  - Intermediate results or partial sum
- External memory



Verilog HDL code structure

- src/
  - Computing units
    - Multiplier with DSP (mul.v)
    - MAC (adder_tree.v, mac.v)
  - On-chip buffers
    - Single-port ram (spram_wrapper.v)
    - Dual-port ram (dpram_wrapper.v)
  - Controller (cnn_ctrl.v)
  - AXI bus interface
    - DMA read (axi_dma_rd.v)
    - DMA write (axi_write_wr.v)
- sim/
  - Input data: input image (butterfly_08bit.hex), CNN parameters (CONV20_W.coe, *_L1.hex)
  - Test benches



Scope

- The baseline code
  - Are designed provide the usage of components via examples or test benches
    - Processing elements: DSP, mul.v, mac.v
    - On-chip buffers: spram, dpram
    - Controller: cnn_ctrl.v
    - Load data from memory (axi_dma_rd.v) or store data to memory (axi_dma_wr.v)

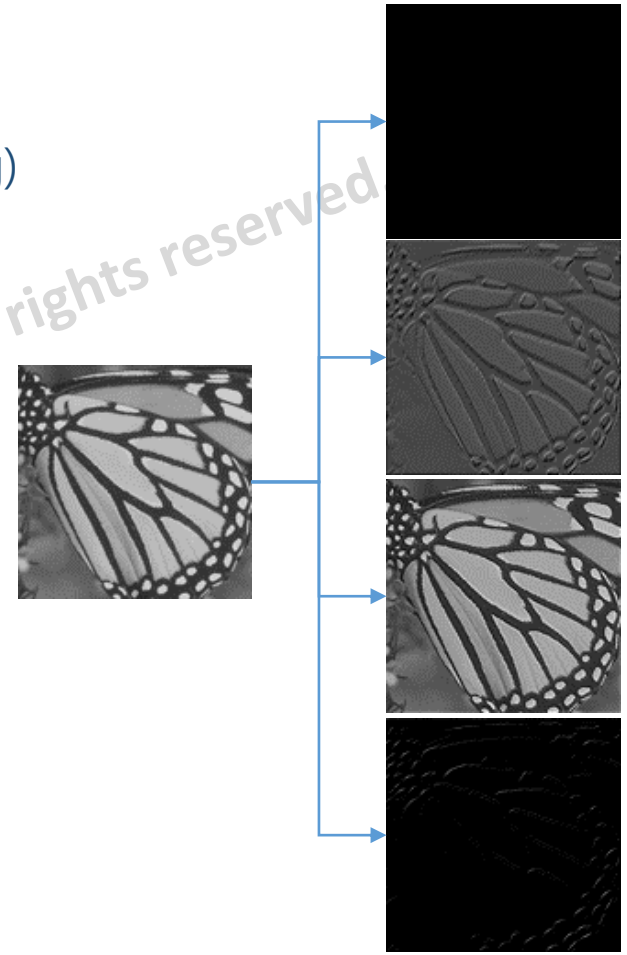
- AIX2022
  - Understand how to use modules
  - Modify/optimize parameters
  - Implementation

layer	type	filter	input	output	Remark
1	conv	3x3x1x16	128x128x1	128x128x16	Tutorial
0	conv	3x3x3x16	320x320x3	320x320x16	AIX2022 Mid-term
1	max		320x320x16	160x160x16	
2	conv	3x3x16x32	160x160x16	160x160x32	
3	max		160x160x32	80x80x32	
4	conv	3x3x32x64	80x80x32	80x80x64	
5	max		80x80x64	40x40x64	

Objective

- A simple task
  - Inputs
    - One gray image: 128x128x1
    - Four filters, biases and scales (batch normalization folding)
      - Filter size 3x3
      - Filters are stored in an 8-bit format
      - Biases and scales are stored in an 16-bit format
  - Outputs
    - Four feature maps

⇒ *Make an inference engine to execute the task*



Road map

Review

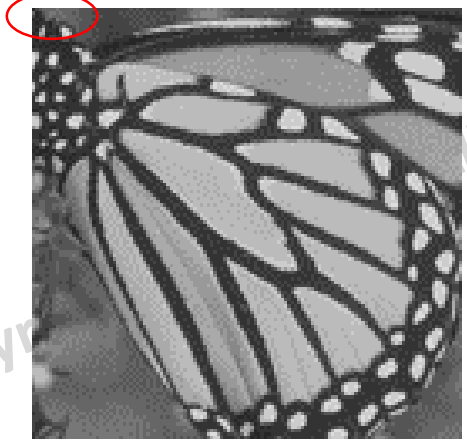
IO files

Controller

Sliding windows

Recap: What is an image?

- An image is an artifact that depicts visual perception, such as a photograph or other two-dimensional picture, that resembles a subject—usually a physical object—and thus provides a depiction of it.
- In the context of signal processing, an image is a distributed amplitude of color(s)
- Example: Each pixel in a gray image is stored in an 8-bit format
  - Pixel value: {BLACK=0, 1, …, 255=WHITE}



42	69	91	99	106	108
105	42	56	84	106	113
72	43	42	68	109	112

Width

height

Pixel

Lab 1: IO files

- Lab 1:
  - Read an input image from a hex file
  - Generate pixel data
  - Store a pixel data in a BMP image

- Goal
  - Understand an image format
  - Visually verification and debugging

BMP file

- A Windows bitmap image starts with 54-byte header data.
- Example: an RGB image 768*512*3
  - **Image size** = 768*512*3= 1179648 bytes

    BMP header = 54 bytes

    BMP File size = Image size + BMP Header = 1179702 Bytes
    - 1179702_{10} = 120036_{hex} Then 4-byte size of BMP file: 12_{hex}= 18, 36_{hex}= 54

      BMP_header[2] = 54; BMP_header[3] = 0 ; BMP_header[4] = 18; BMP_header[5] = 0 ;
  - **Image width** = 768 = $0x0300_{hex}$. The 4 bytes of the image width are 0, 3, 0, 0.

    BMP_header[18] = 0; BMP_header[19] = 3; BMP_header[20] = 0; BMP_header[21] = 0;
  - **Image height** = 512 = $0x0200_{hex}$. The 4 bytes of the image width are 0, 2, 0, 0.

    BMP_header[22] = 0; BMP_header[23] = 2; BMP_header[24] = 0; BMP_header[25] = 0;

Data preparation: Input file

- Verilog can not read an image directly from a BMP image file
- Data preparation
  - Read an image file (stb_image.c)
  - Convert it to a hexadecimal file
    - 8/32-bit format
    - The 32-bit format can be easily extended for an RGB image.

butterfly_08bit.hex

1	2a
2	45
3	5b
4	63
5	6a
6	6c
7	6f
8	6d
9	6d
10	6a
11	67
12	64
13	62
14	61
15	5c
16	58

8-bit

butterfly_32bit.hex

1	0000002a
2	00000045
3	0000005b
4	00000063
5	0000006a
6	0000006c
7	0000006f
8	0000006d
9	0000006d
10	0000006a
11	00000067
12	00000064
13	00000062
14	00000061
15	0000005c
16	00000058

32-bit

Bmp writer (bmp_image_writer.v)

- Write an BMP file from incoming pixels
  - Simulation, verification, debugging
- Ports:
  - Inputs:
    - Clock (clk), reset (rstn)
    - Pixel input (din[7:0])
    - Valid signal (vld)
  - Output:
    - frame_done
- Parameters
  - Pixel size (WI) = 8
  - BMP header size = 54
  - WIDTH=HEIGHT=128
  - OUTFILE: location to store the output file

```
1  `timescale 1ns/1ps
2
3  module bmp_image_writer
4  #(parameter WI = 8,
5  parameter BMP_HEADER_NUM = 54,
6  parameter WIDTH    = 128,
7  parameter HEIGHT   = 128,
8  parameter OUTFILE   = "./out/oonvout.bmp")(
9      input clk,
10     input rstn,
11     input [WI-1:0] din,
12     input vld,
13     output reg frame_done
14 );
```

Bmp writer (bmp_image_writer.v)

- Local parameters: frame/image size
- Internal signals
  - Pixel counter (pixel_count)
    - Get the index of an incoming pixel
    - 0→1→…→frame_size-1
  - Pixel buffer (out_img)
    - Store all pixel data
  - Registers for an BMP header
  - Indexes: k, l, h, w
  - File pointe: fd

```
16  // Image parameters
17  localparam FRAME_SIZE = WIDTH*HEIGHT;
18  localparam FRAME_SIZE_W = $clog2(FRAME_SIZE);
19  reg [WI-1:0] out_img[0:FRAME_SIZE-1];   // Output feature map
20  reg [FRAME_SIZE_W-1:0] pixel_count;
21  reg [31:0] IW;
22  reg [31:0] IH;
23  reg [31:0] SZ;
24  reg [7:0] BMP_header [0 : BMP_HEADER_NUM - 1];
25  integer k;
26  integer fd;
27  integer i;
28  integer h, w;
```

Bmp writer (bmp_image_writer.v)

- Initialization
  - Reset all registers (pixel counter, frame done)
- For an incoming pixel (vld==1)
  - Update the pixel index (pixel_count)
    - If all pixels are received (pixel_count==FRAME_SIZE-1)
      - Reset the index
      - Set frame_done to HIGH
  - Based on the index, update the buffer
    - out_img[pixel_count] ← din

```
29  //------------------------------------------------------------
30  // Update the internal buffers.
31  //------------------------------------------------------------
32  always@(posedge clk, negedge rstn) begin
33      if(!rstn) begin
34          for(k=0;k<FRAME_SIZE;k=k+1) begin
35              out_img[k] <= 0;
36          end
37          pixel_count <= 0;
38          frame_done <= 1'b0;
39      end else begin
40          if(vld) begin
41              if(pixel_count == FRAME_SIZE-1) begin
42                  pixel_count <= 0;
43                  frame_done <= 1'b1;
44              end
45              else begin
46                  pixel_count <= pixel_count + 1;
47              end
48              out_img[pixel_count] <= din;
49          end
50      end
51  end
```

Bmp writer (bmp_image_writer.v)

- When all pixels are stored in a buffer
  - frame_done == 1

  ⇒ write an BMP image file
- $open(OUTFILE, "wb+"):
  - Open a binary file for writing
  - The integer fd is the file identifier
- $fwrite(fd, "%c", ...)
  - Write a character to the file
- Debug: Open a txt file and write data in a hex file

```
116  initial begin
117      // Open file
118      fd = $fopen(OUTFILE, "wb+");
119      h = 0;
120      w = 0;
121  end
122
123  always@(frame_done) begin
124      if(frame_done == 1'b1) begin
125      // Write header
126          for(i=0; i<BMP_HEADER_NUM; i=i+1) begin
127              $fwrite(fd, "%o", BMP_header[i][7:0]);
128          end
129
130          // Write data
131          for(h = 0; h < HEIGHT; h = h + 1) begin
132              for(w = 0; w < WIDTH; w = w + 1) begin
133                  $fwrite(fd, "%o", out_img[(HEIGHT-1-h)*WIDTH + w][7:0]);
134                  $fwrite(fd, "%o", out_img[(HEIGHT-1-h)*WIDTH + w][7:0]);
135                  $fwrite(fd, "%o", out_img[(HEIGHT-1-h)*WIDTH + w][7:0]);
136              end
137          end
138          $fclose(fd);
139
140      end
141  end
```

Open a binary file

Write Header

Write data

Test bench (inout_tb.v)

- Parameters: WIDTH, HEIGHT, INFILE, VSYNC_DELAY, HSYNC_DELAY
- Signals: in_img, vld, din

```
1  `timescale 1ns / 1ps
2
3  module inout_tb;
4  parameter WIDTH    = 128;
5  parameter HEIGHT   = 128;
6  parameter INFILE    = "C:/AIX2022/aix2022/sim/input_data/butterfly_08bit.hex";
7  parameter OUTFILE    = "C:/AIX2022/aix2022/sim/output/butterfly.bmp";
8  parameter VSYNC_DELAY = 200;
9  parameter HSYNC_DELAY = 160;
10 parameter WI = 8;
11 localparam FRAME_SIZE = WIDTH * HEIGHT;
12 localparam FRAME_SIZE_W = $clog2(FRAME_SIZE);
13 reg [7:0] in_img [0:FRAME_SIZE-1]; // Input image
14 reg clk;
15 reg rstn;
16 reg vld;
17 reg [7:0] din;
18 wire frame_done;
```

Need to check it for a valid directory

Test bench (inout_tb.v)

- Design under test (DUT):
  - bmp_image_writer
  - Change the directory for the output image
    - Set OUTFILE
- Clock 100MHz

```
20  //------------------------------------------------------------
21  // DUT: bmp_image_writer.v
22  //------------------------------------------------------------
23  bmp_image_writer #(.OUTFILE(OUTFILE))
24  u_bmp_image_writer(
25  ./*input     */clk(clk),
26  ./*input     */rstn(rstn),
27  ./*input[7:0]*/din(din),
28  ./*input     */vld(vld),
29  ./*output    */frame_done(frame_done)
30  );
31  // Clock
32  parameter CLK_PERIOD = 10; //100MHz
33  initial begin
34      clk = 1'b1;
35      forever #(CLK_PERIOD/2) clk = ~clk;
36  end
37  integer row, col;
```

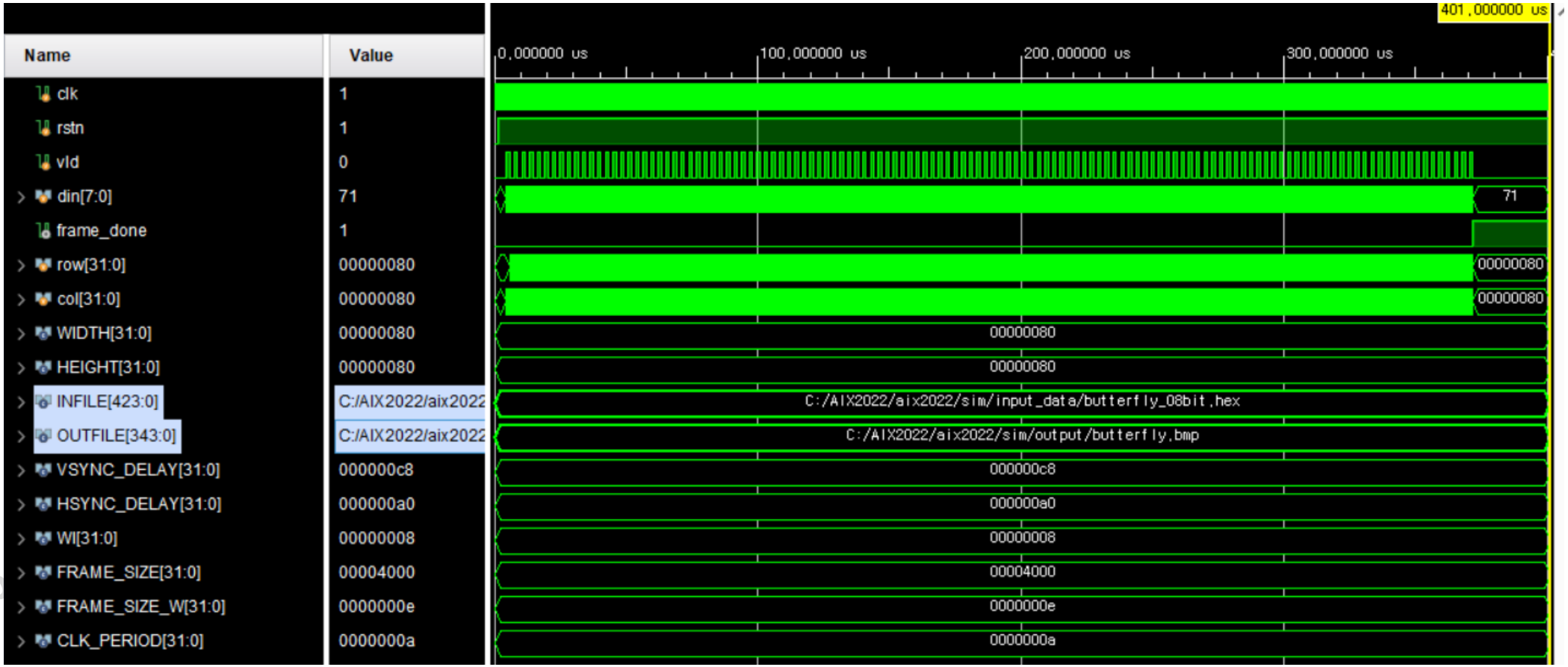
Test bench (inout_tb.v)

- $readmemh: Read a hex file (INFILE) to a buffer (in_img)
- Pseudo code for test cases
  - Wait VSYNC_DELAY cycles before starting a frame
    - vld=1
  - for row = 0→HEIGHT-1
    - for col = 0→WIDTH-1
      - vld = 1'b1
      - din ← in_img[row*WIDTH+col]
    - end for
  - end for
  - vld=1'b0

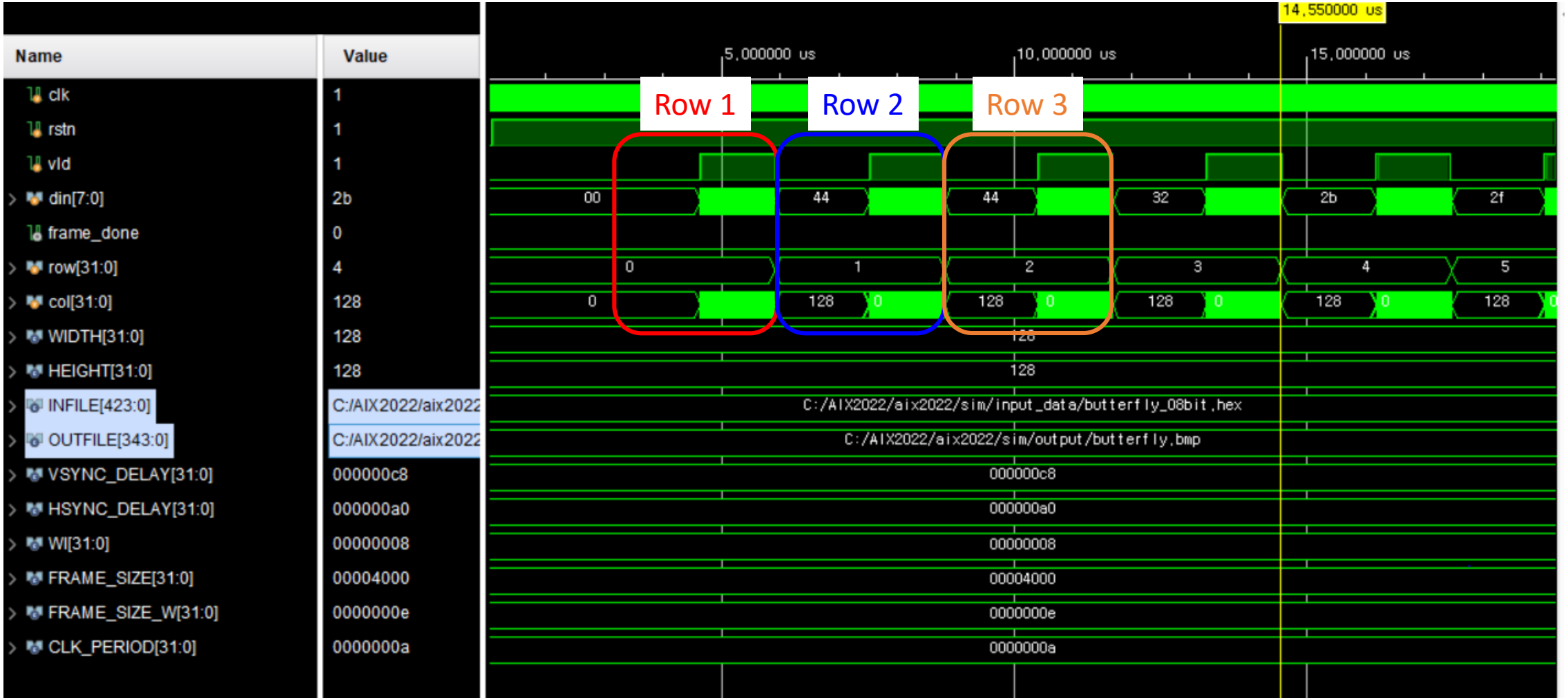
```
42  // Read the input file to memory
43  initial begin
44      $readmemh(INFILE, in_img ,0,FRAME_SIZE-1);
45  end
46
47  initial begin
48      rstn = 1'b0;            // Reset, low active
49      vld = 0;
50      din = 0;
51      row = 0;
52      col = 0;
53
54      // Reset
55      repeat(100) @(posedge clk);
56      #(4*CLK_PERIOD) rstn = 1'b1;
57
58      // Frame/layer sync
59      repeat(VSYNC_DELAY)@(posedge clk);
60      for(row = 0; row < HEIGHT; row = row + 1) begin
61          // Line sync
62          repeat(HSYNC_DELAY)@(posedge clk) vld = 1'b0;
63          // Pixel data
64          for(col = 0; col < WIDTH; col = col + 1) begin
65            @(posedge clk) vld = 1'b1;
66              din = in_img[row * WIDTH + col];
67          end
68      end
69
70      // Layer don
71      @(posedge clk) vld = 0;
72      $display("Layer done!");
73  end
```

Waveform (inout_tb.v)

- Simulation with time 400 us

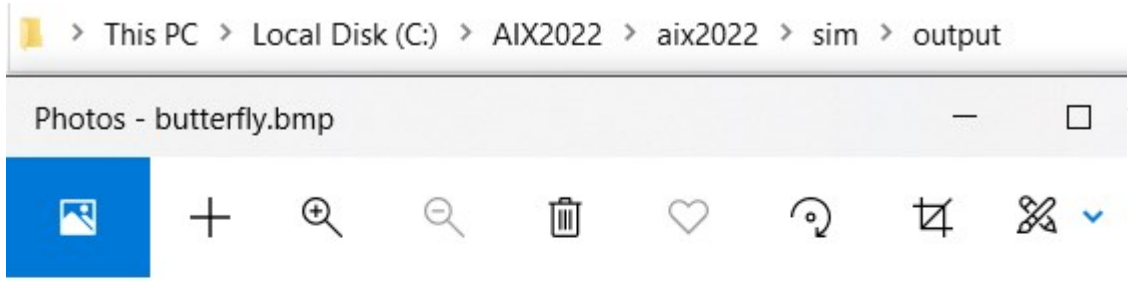


Waveform (inout_tb.v)



Waveform (inout_tb.v)

- The output image is stored at the predefined directory (OUTFILE)



Road map

Review

IO files

Controller

Sliding windows

Motivation

- A pseudo code to generate an output order
  - Wait VSYNC_DELAY cycles before starting a frame
    - vld=1
  - for row = 0→HEIGHT-1
    - for col = 0→WIDTH-1
      - vld = 1'b1
      - din ← in_img[row*WIDTH+col]
    - end for
  - end for
  - vld=1'b0

layer	type	filter	input	output	Remark
1	conv	3x3x1x16	128x128x1	128x128x16	Tutorial
0	conv	3x3x3x16	320x320x3	320x320x16	AIX2022 Mid-term
1	max		320x320x16	160x160x16	
2	conv	3x3x16x32	160x160x16	160x160x32	
3	max		160x160x32	80x80x32	
4	conv	3x3x32x64	80x80x32	80x80x64	
5	max		80x80x64	40x40x64	

Lab 2: controller (cnn_ctrl.v)

- Generate control signals: Loop generator
- Inputs
  - Clock, reset
  - q_width, q_height, q_frame_size
  - Synchronization delay
    - Frame/layer synchronization (vsync_delay)
    - Row/line synchronization (hsync_delay)
  - Trigger (q_start)
- Outputs
  - Synchronization signals (o_ctrl_vsync_run, o_ctrl_hsync_run, o_ctrl_data_run)
  - Row, column, and pixel index (o_row, o_col, o_data_count)
  - Frame/layer done (o_end_frame)

```
1   `timescale 1ns / 1ps
2
3   module cnn_ctrl(
4   clk,
5   rstn,
6   // Inputs
7   q_width,
8   q_height,
9   q_vsync_delay,
10  q_hsync_delay,
11  q_frame_size,
12  q_start,
13  //output
14  o_ctrl_vsync_run,
15  o_ctrl_vsync_cnt,
16  o_ctrl_hsync_run,
17  o_ctrl_hsync_cnt,
18  o_ctrl_data_run,
19  o_row,
20  o_col,
21  o_data_count,
22  o_end_frame
23  );
```

Signals

- States
  - ST_IDLE: IDLE state,
    - Before data communication, computation
  - ST_VSYNC:
    - Frame/layer synchronization
    - Data preparation/transferring
    - May preload some filters or input pixels
  - ST_HYNC
    - Line/row synchronization
    - Data preparation/transferring
    - May preload some filters or input pixels
  - ST_DATA
    - Computation
- Registers: row, col, data_count, end_frame

```
46  //------------------------------------------------------------
47  // Internal signals
48  //------------------------------------------------------------
49  localparam      ST_IDLE     = 2'b00,
50                  ST_VSYNC    = 2'b01,
51                  ST_HSYNC    = 2'b10,
52                  ST_DATA     = 2'b11;
53  reg [1:0] cstate, nstate;
54  reg               ctrl_vsync_run;
55  reg [W_DELAY-1:0] ctrl_vsync_cnt;
56  reg               ctrl_hsync_run;
57  reg [W_DELAY-1:0] ctrl_hsync_cnt;
58  reg               ctrl_data_run;
59  reg [W_SIZE-1:0]  row;
60  reg [W_SIZE-1:0]  col;
61  reg [W_FRAME_SIZE-1:0] data_count;
62  wire end_frame;
```

Finite State Machine (FSM)

- Update the current state (cstate) by the next state (nstate)
  - Sequential logic
- Decide the next state based on the current state and other conditions.
  - Combinational logic

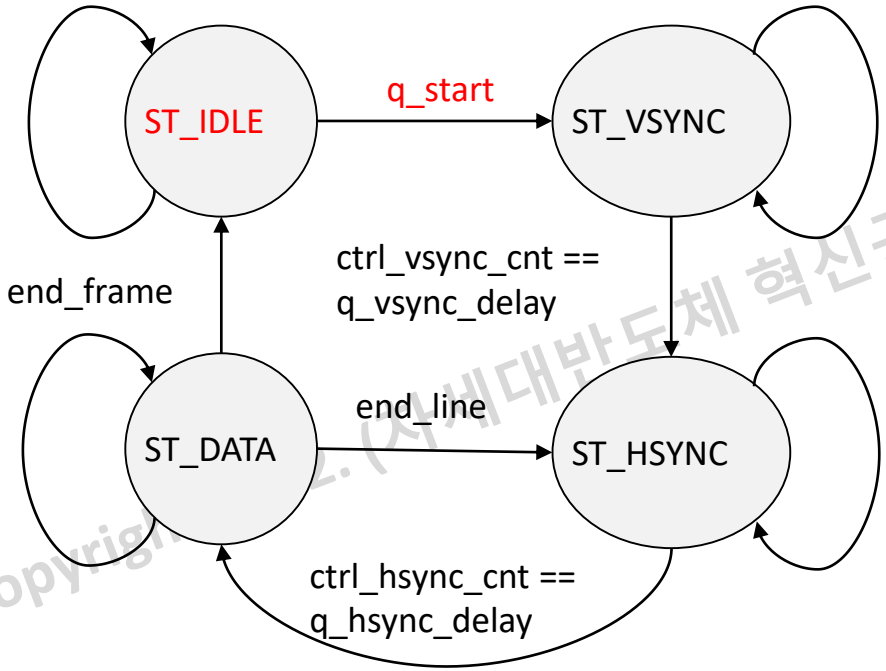
```
always @ (posedge clk, negedge rstn)
begin
   if(~rstn) begin
      cstate <= ST_IDLE;
   end
   else begin
      cstate <= nstate;
   end
end

always @ (*) begin
   case(cstate)
      ST_IDLE: begin
            if(start)
               nstate = ST_VSYNC;
            else
               nstate = ST_IDLE;
      end
      …
      default: nstate = ST_IDLE;
   endcase
end
```

Update the current state (cstate) by the next state (nstate)

Decide the next state based on the current state and other conditions.

Finite State Machine (FSM)

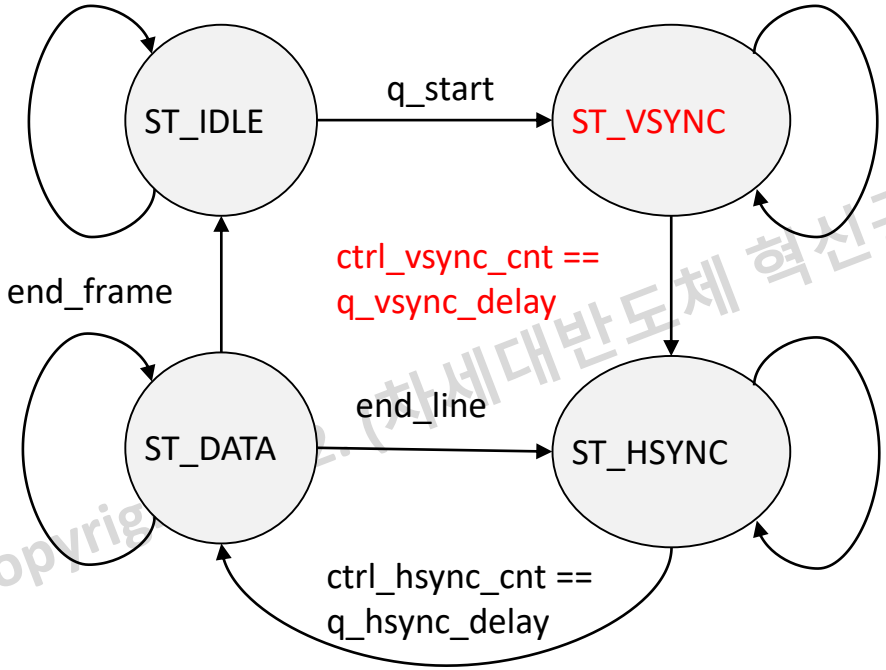
- ST_IDLE:
  - FSM is initialized at ST_IDLE
  - When "start" goes HIGH, FSM moves to ST_VSYNC.




```
always @(*) begin
  case(cstate)
      ST_IDLE: begin
            if(q_start)
               nstate = ST_VSYNC;
            else
               nstate = ST_IDLE;
      end
    ST_VSYNC: begin
        if(ctrl_vsync_cnt == q_vsync_delay)
           nstate = ST_HSYNC;
        else
           nstate = ST_VSYNC;
    end
    ST_HSYNC: begin
        if(ctrl_hsync_cnt == q_hsync_delay)
           nstate = ST_DATA;
        else
           nstate = ST_HSYNC;
    end
    ST_DATA: begin
        if(end_frame) begin   //end of frame
           nstate = ST_IDLE;
        end
        else begin
           if(col == q_width-1)   //end of line
              nstate = ST_HSYNC;
           else
              nstate = ST_DATA;
        end
    end
    default: nstate = ST_IDLE;
  endcase
end
```

Finite State Machine

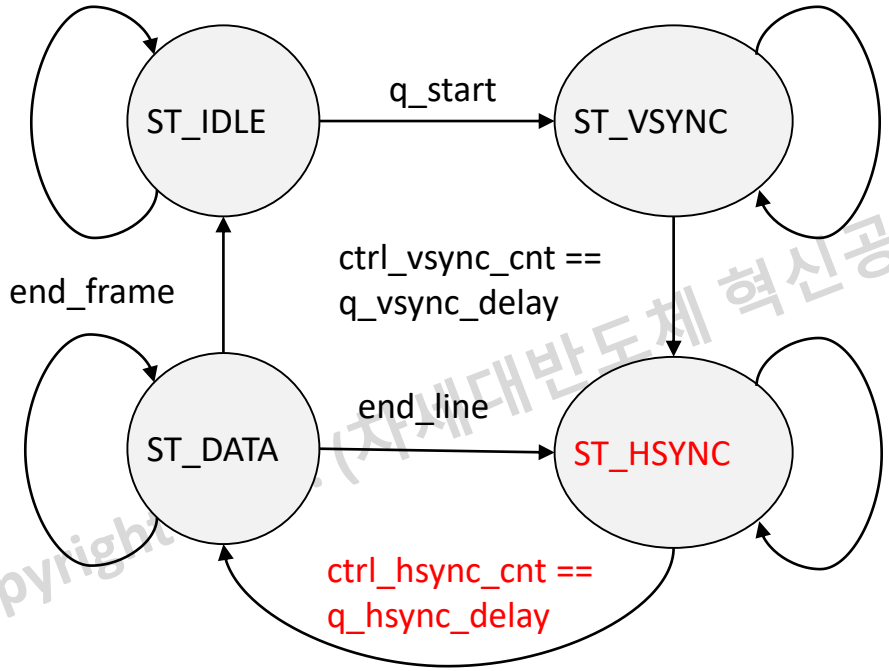
- ST_VSYNC: Frame synchronization
  - Start up for a frame
  - There is an VSYNC counter.
  - When the counter reaches to q_vsync_delay, FSM moves to ST_HSYNC.




```
always @(*) begin
  case(cstate)
    ST_IDLE: begin
        if (q_start)
          nstate = ST_VSYNC;
        else
          nstate = ST_IDLE;
    end
      ST_VSYNC: begin
            if(ctrl_vsync_cnt == q_vsync_delay)
              nstate = ST_HSYNC;
            else
              nstate = ST_VSYNC;
      end
    ST_HSYNC: begin
        if (ctrl_hsync_cnt == q_hsync_delay)
          nstate = ST_DATA;
        else
          nstate = ST_HSYNC;
    end
    ST_DATA: begin
        if (end_frame) begin   //end of frame
           nstate = ST_IDLE;
        end
        else begin
          if(col == q_width-1)    //end of line
            nstate = ST_HSYNC;
          else
            nstate = ST_DATA;
        end
    end
    default: nstate = ST_IDLE;
  endcase
end
```

Finite State Machine

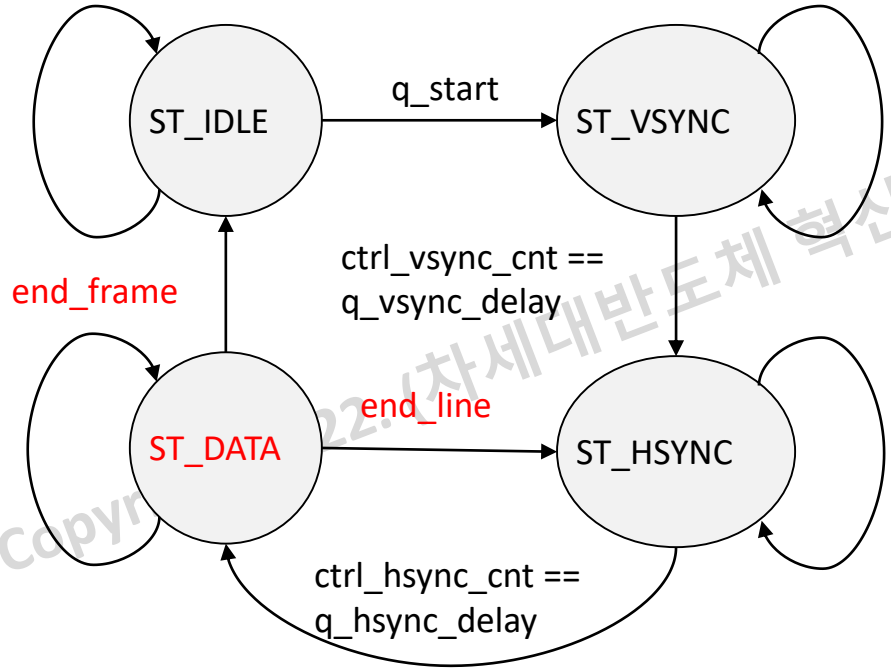
- ST_HSYNC: Line synchronization
  - There is an HSYNC counter.
  - When the counter reaches to q_hsync_delay, FSM moves to ST_DATA.




```
always @(*) begin
  case (cstate)
    ST_IDLE: begin
        if (q_start)
          nstate = ST_VSYNC;
        else
          nstate = ST_IDLE;
    end
    ST_VSYNC: begin
        if (ctrl_vsync_cnt == q_vsync_delay)
          nstate = ST_HSYNC;
        else
          nstate = ST_VSYNC;
    end
    ST_HSYNC: begin
          if (ctrl_hsync_cnt == q_hsync_delay)
            nstate = ST_DATA;
          else
            nstate = ST_HSYNC;
    end
    ST_DATA: begin
        if (end_frame) begin   //end of frame
           nstate = ST_IDLE:
        end
        else begin
          if (col == q_width-1)   //end of line
            nstate = ST_HSYNC;
          else
            nstate = ST_DATA;
        end
    end
    default: nstate = ST_IDLE;
  endcase
end
```

Finite State Machine

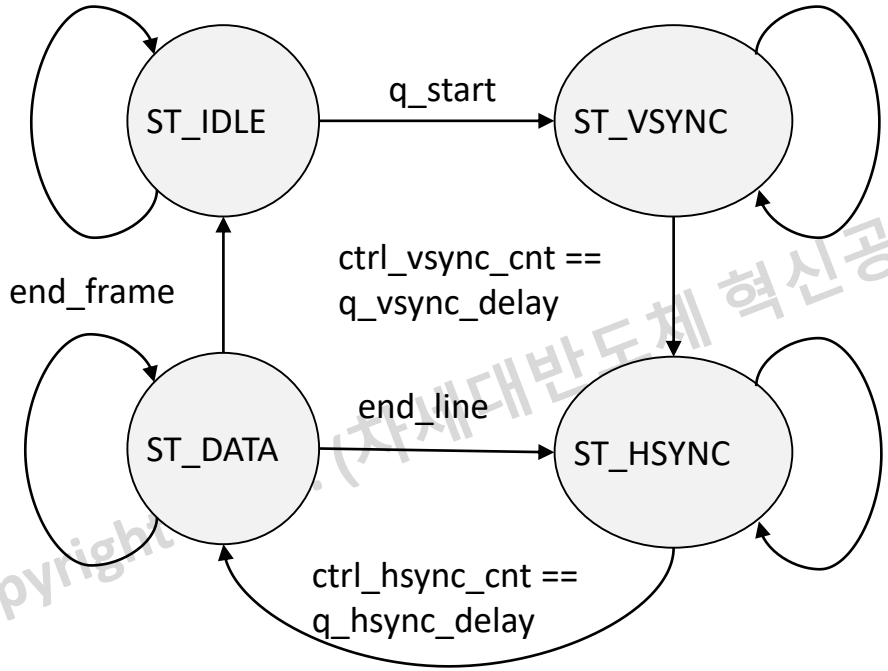
- ST_DATA: Sending pixel data
  - There are two counters
  - Line data counter: if it reaches to end of line, FSM moves to ST_HSYNC.
  - Frame data counter: if it reaches to end of frame, FSM moves to ST_IDLE.




```
always @(*) begin
  case(cstate)
    ST_IDLE: begin
        if (q_start)
          nstate = ST_VSYNC;
        else
          nstate = ST_IDLE;
    end
    ST_VSYNC: begin
        if (ctrl_vsync_cnt == q_vsync_delay)
          nstate = ST_HSYNC;
        else
          nstate = ST_VSYNC;
    end
    ST_HSYNC: begin
        if (ctrl_hsync_cnt == q_hsync_delay)
          nstate = ST_DATA;
        else
          nstate = ST_HSYNC;
    end
      ST_DATA: begin
            if (end_frame) begin   //end of frame
                nstate = ST_IDLE;
            end
            else begin
              if (col == q_width-1)    //end of line
                nstate = ST_HSYNC;
              else
                nstate = ST_DATA;
            end
      end
    default: nstate = ST_IDLE;
  endcase
end
```

States and sync. counters

- States: ctrl_vsync_run, ctrl_hsync_run, ctrl_data_run
- Synchronization counters
  - ctrl_vsync_cnt
  - ctrl_hsync_cnt




```
108  always @(*) begin
109      ctrl_vsync_run = 0;
110      ctrl_hsync_run = 0;
111      ctrl_data_run  = 0;
112      case(cstate)
113          ST_VSYNC:   begin ctrl_vsync_run = 1; end
114          ST_HSYNC:   begin ctrl_hsync_run = 1; end
115          ST_DATA:    begin ctrl_data_run  = 1; end
116      endcase
117  end
118  always@(posedge clk, negedge rstn)
119  begin
120      if(!rstn) begin
121          ctrl_vsync_cnt <= 0;
122          ctrl_hsync_cnt <= 0;
123      end
124      else begin
125          if(ctrl_vsync_run)
126              ctrl_vsync_cnt <= ctrl_vsync_cnt + 1;
127          else
128              ctrl_vsync_cnt <= 0;
129
130          if(ctrl_hsync_run)
131              ctrl_hsync_cnt <= ctrl_hsync_cnt + 1;
132          else
133              ctrl_hsync_cnt <= 0;
134      end
135  end
```

Row, column, pixel counters

- Three counters are related to computation (ST_DATA)
  - Updated when ctrl_data_run
  - Column (col)
    - 0 → 1 → … → q_width-1 → 0
    - Updated pixel by pixel
  - Row (row)
    - 0 → 1 → … → q_height-1 → 0
    - Updated row by row (col == q_width-1)
  - Pixel (data_count)
    - 0 → 1 → … → q_frame_size-1 → 0
    - Updated pixel by pixel
- The layer/frame done flag (end_frame) is raised to HIGH when the pixel counter reaches to q_frame_size-1

```
136  always@(posedge clk, negedge rstn)
137  begin
138      if(!rstn) begin
139          row <= 0;
140          col <= 0;
141      end
142      else begin
143          if(ctrl_data_run) begin
144              if(col == q_width - 1) begin
145                  if(end_frame)
146                      row <= 0;
147                  else
148                      row <= row + 1;
149              end
150              if(col == q_width - 1)
151                  col <= 0;
152              else
153                  col <= col + 1;
154          end
155      end
156  end
157  always@(posedge clk, negedge rstn)
158  begin
159      if(!rstn) begin
160          data_count <= 0;
161      end
162      else begin
163          if(ctrl_data_run) begin
164              if(!end_frame)
165                  data_count <= data_count + 1;
166              else
167                  data_count <= 0;
168          end
169      end
170  end
171  assign end_frame = (data_count == q_frame_size-1)? 1'b1: 1'b0;
```

Test bench (cnn_ctrl_tb.v)

- Parameters:
  - WIDTH, HEIGHT, FRAME_SIZE
  - VSYNC_DELAY, HSYNC_DELAY
- Signals
  - Registers
    - q_width, q_height, q_frame_size
    - q_vsync_delay, q_hsync_delay
    - q_start
  - Wires to monitor outputs of cnn_ctrl.v
    - ctrl_vsync_run, ctrl_hsync_run, ctrl_data_run
    - row, col, data_count, end_frame

```
1   `timescale 1ns / 1ps
2
3   module cnn_ctrl_tb;
4   parameter W_SIZE  = 12;
5   parameter W_FRAME_SIZE  = 2 * W_SIZE + 1;
6   parameter W_DELAY = 12;
7   parameter WIDTH   = 128;
8   parameter HEIGHT  = 128;
9   parameter FRAME_SIZE = WIDTH * HEIGHT;
10  parameter VSYNC_DELAY = 200;
11  parameter HSYNC_DELAY = 160;
12
13  reg clk, rstn;
14  reg [W_SIZE-1 :0] q_width;
15  reg [W_SIZE-1 :0] q_height;
16  reg [W_DELAY-1:0] q_vsync_delay;
17  reg [W_DELAY-1:0] q_hsync_delay;
18  reg [W_FRAME_SIZE-1:0] q_frame_size;
19  reg q_start;
20
21  wire              ctrl_vsync_run;
22  wire [W_DELAY-1:0]  ctrl_vsync_cnt;
23  wire              ctrl_hsync_run;
24  wire [W_DELAY-1:0]  ctrl_hsync_cnt;
25  wire              ctrl_data_run;
26  wire [W_SIZE-1:0]   row;
27  wire [W_SIZE-1:0]   col;
28  wire [W_FRAME_SIZE-1:0]data_count;
29  wire end_frame;
```

Test bench (cnn_ctrl_tb.v): DUT

layer	type	filter	input	output	Remark
1	conv	3x3x1x16	128x128x1	128x128x16	Tutorial
0	conv	3x3x3x16	320x320x3	320x320x16	
1	max		320x320x16	160x160x16	
2	conv	3x3x16x32	160x160x16	160x160x32	AIX2022
3	max		160x160x32	80x80x32	Mid-term
4	conv	3x3x32x64	80x80x32	80x80x64	
5	max		80x80x64	40x40x64	

Layer configurations
q_width, q_height, q_frame_size, …

Control signals
ctrl_vync_run, ctrl_hsync_run, …
row, col, data_count, end_frame

```
31  //------------------------------------------------------------
32  // Controller (FSM)
33  //------------------------------------------------------------
34  cnn_ctrl u_cnn_ctrl (
35  .clk(clk),
36  .rstn(rstn),
37  // Inputs
38  .q_width(q_width),
39  .q_height(q_height),
40  .q_vsync_delay(q_vsync_delay),
41  .q_hsync_delay(q_hsync_delay),
42  .q_frame_size(q_frame_size),
43  .q_start(q_start),
44  //output
45  .o_ctrl_vsync_run(ctrl_vsync_run),
46  .o_ctrl_vsync_cnt(ctrl_vsync_cnt),
47  .o_ctrl_hsync_run(ctrl_hsync_run),
48  .o_ctrl_hsync_cnt(ctrl_hsync_cnt),
49  .o_ctrl_data_run(ctrl_data_run),
50  .o_row(row),
51  .o_col(col),
52  .o_data_count(data_count),
53  .o_end_frame(end_frame)
54  );
```

Test cases

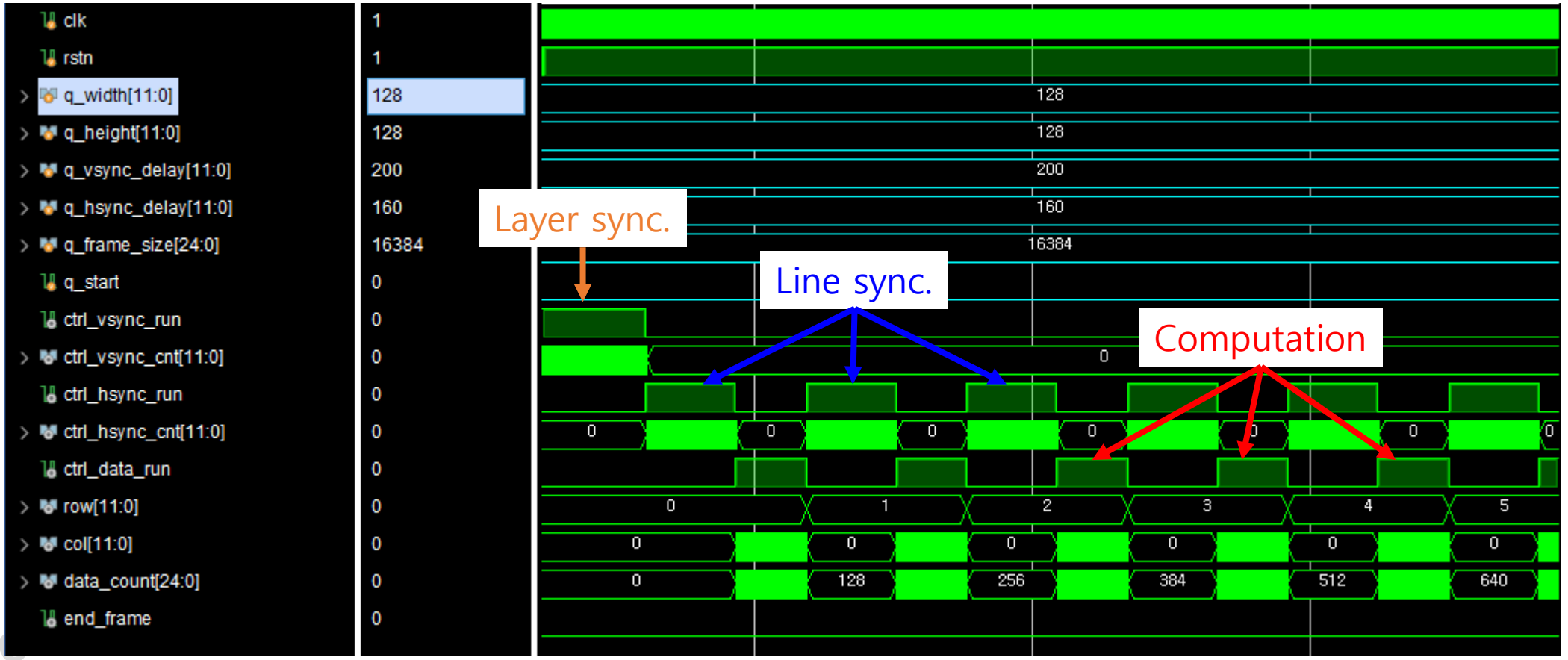
- Set parameters
  - q_width, q_height, q_frame_size
  - q_vsync_delay, q_hsync_delay
- Trigger with "q_start"

```
63  //------------------------------------------------------------
64  // Test cases
65  //------------------------------------------------------------
66  initial begin
67      rstn = 1'b0;            // Reset, low activ
68      q_width         = WIDTH;
69      q_height        = HEIGHT;
70      q_vsync_delay   = VSYNC_DELAY;
71      q_hsync_delay   = HSYNC_DELAY;
72      q_frame_size    = FRAME_SIZE;
73      q_start         = 1'b0;
74
75      #(4*CLK_PERIOD) rstn = 1'b1;
76
77      #(100*CLK_PERIOD)
78          @(posedge clk)
79              q_start = 1'b1;
80      #(4*CLK_PERIOD)
81          @(posedge clk)
82              q_start = 1'b0;
83
84  end
```

Waveform (cnn_ctrl_tb.v)

- Simulation with time = 400 microseconds

Waveform (cnn_ctrl_tb.v)



Road map

Review

IO files

Controller

Sliding windows

Lab 3: Sliding windows (cnv_tb.v)

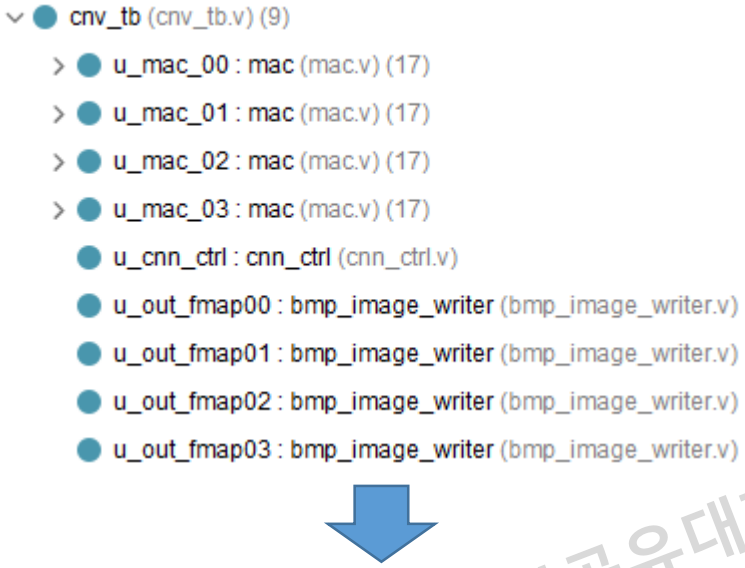
- Lab 3:
  - Use four convolutional kernels (mac.v)
  - Use a CNN controller (cnn_ctrl.v)
  - Generate din, vld
    - Do calculation
  - Store the output results (bmp_image_writer.v)

Convolutional kernels (cnv_tb.v)

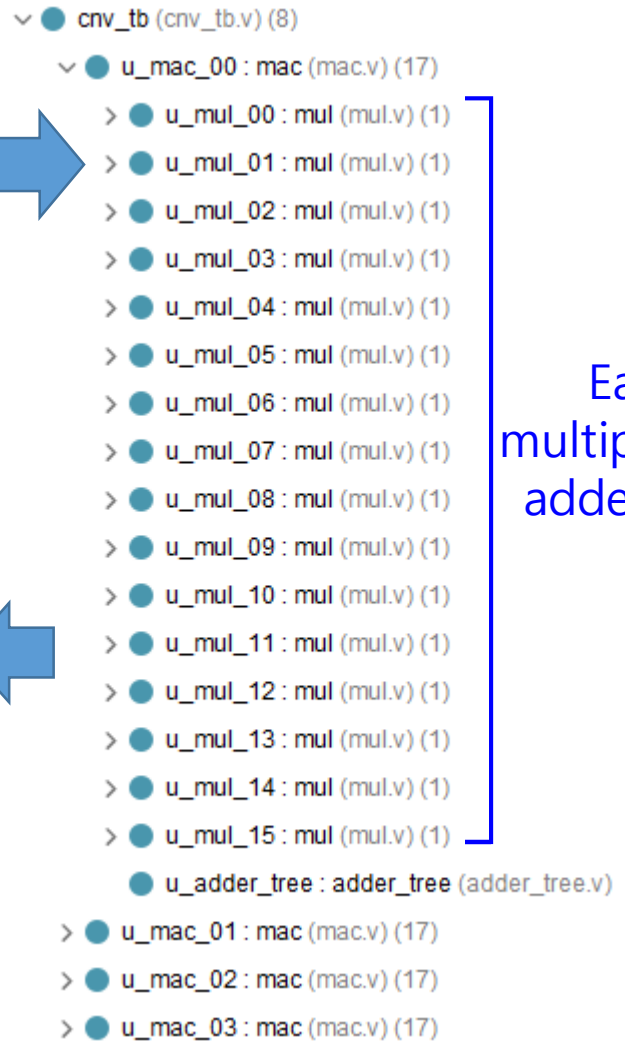
Four convolutiona kernels (mac.v)

CNN controller

Four output writer modules



Execute 64 multiplication operations in parallel



Each kernel has 16 multipliers (mul.v) and one adder tree (adder_tree.v)

Convolutional kernels (cnv_tb.v)

- Predefine four 3x3 filters (win[0], win[1], win[2], and win[3])
  - Come from a quantized model

```
173     // CNN filters of four output channels
174     win[0][  7:  0] = 8'd142; win[1][  7:  0] = 8'd69 ; win[2][  7:  0] = 8'd13 ; win[3][  7:  0] = 8'd69 ;
175     win[0][ 15:  8] = 8'd151; win[1][ 15:  8] = 8'd181; win[2][ 15:  8] = 8'd244; win[3][ 15:  8] = 8'd135;
176     win[0][ 23: 16] = 8'd215; win[1][ 23: 16] = 8'd209; win[2][ 23: 16] = 8'd255; win[3][ 23: 16] = 8'd235;
177     win[0][ 31: 24] = 8'd127; win[1][ 31: 24] = 8'd19 ; win[2][ 31: 24] = 8'd241; win[3][ 31: 24] = 8'd128;
178     win[0][ 39: 32] = 8'd163; win[1][ 39: 32] = 8'd128; win[2][ 39: 32] = 8'd127; win[3][ 39: 32] = 8'd32 ;
179     win[0][ 47: 40] = 8'd205; win[1][ 47: 40] = 8'd95 ; win[2][ 47: 40] = 8'd240; win[3][ 47: 40] = 8'd90 ;
180     win[0][ 55: 48] = 8'd229; win[1][ 55: 48] = 8'd221; win[2][ 55: 48] = 8'd252; win[3][ 55: 48] = 8'd48 ;
181     win[0][ 63: 56] = 8'd255; win[1][ 63: 56] = 8'd121; win[2][ 63: 56] = 8'd237; win[3][ 63: 56] = 8'd52 ;
182     win[0][ 71: 64] = 8'd113; win[1][ 71: 64] = 8'd8  ; win[2][ 71: 64] = 8'd1  ; win[3][ 71: 64] = 8'd211;
183     win[0][ 79: 72] = 8'd0  ; win[1][ 79: 72] = 8'd0  ; win[2][ 79: 72] = 8'd0  ; win[3][ 79: 72] = 8'd0  ;
184     win[0][ 87: 80] = 8'd0  ; win[1][ 87: 80] = 8'd0  ; win[2][ 87: 80] = 8'd0  ; win[3][ 87: 80] = 8'd0  ;
185     win[0][ 95: 88] = 8'd0  ; win[1][ 95: 88] = 8'd0  ; win[2][ 95: 88] = 8'd0  ; win[3][ 95: 88] = 8'd0  ;
186     win[0][103: 96] = 8'd0  ; win[1][103: 96] = 8'd0  ; win[2][103: 96] = 8'd0  ; win[3][103: 96] = 8'd0  ;
187     win[0][111:104] = 8'd0  ; win[1][111:104] = 8'd0  ; win[2][111:104] = 8'd0  ; win[3][111:104] = 8'd0  ;
188     win[0][119:112] = 8'd0  ; win[1][119:112] = 8'd0  ; win[2][119:112] = 8'd0  ; win[3][119:112] = 8'd0  ;
189     win[0][127:120] = 8'd0  ; win[1][127:120] = 8'd0  ; win[2][127:120] = 8'd0  ; win[3][127:120] = 8'd0  ;
190
```

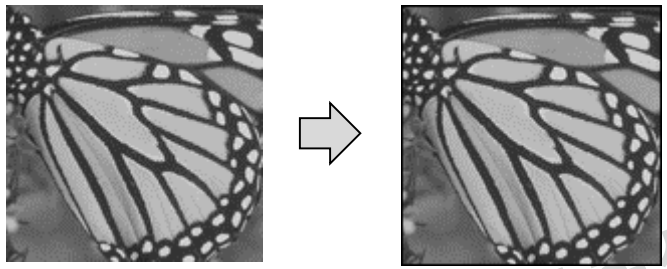
- Later, filters should be preloaded during ST_VSYNC, ST_HSYNC or ST_DATA.

Sliding windows and mapping

```
173     // CNN filters of four output channels
174     win[0][  7:  0] = 8'd142; win[1][  7:  0] = 8'd69 ; win[2][  7:  0] = 8'd13 ; win[3][  7:  0] = 8'd69 ;
175     win[0][ 15:  8] = 8'd151; win[1][ 15:  8] = 8'd181; win[2][ 15:  8] = 8'd244; win[3][ 15:  8] = 8'd135;
176     win[0][ 23: 16] = 8'd215; win[1][ 23: 16] = 8'd209; win[2][ 23: 16] = 8'd255; win[3][ 23: 16] = 8'd235;
177     win[0][ 31: 24] = 8'd127; win[1][ 31: 24] = 8'd19 ; win[2][ 31: 24] = 8'd241; win[3][ 31: 24] = 8'd128;
178     win[0][ 39: 32] = 8'd163; win[1][ 39: 32] = 8'd128; win[2][ 39: 32] = 8'd127; win[3][ 39: 32] = 8'd32 ;
179     win[0][ 47: 40] = 8'd205; win[1][ 47: 40] = 8'd95 ; win[2][ 47: 40] = 8'd240; win[3][ 47: 40] = 8'd90 ;
180     win[0][ 55: 48] = 8'd229; win[1][ 55: 48] = 8'd221; win[2][ 55: 48] = 8'd252; win[3][ 55: 48] = 8'd48 ;
181     win[0][ 63: 56] = 8'd255; win[1][ 63: 56] = 8'd121; win[2][ 63: 56] = 8'd237; win[3][ 63: 56] = 8'd52 ;
182     win[0][ 71: 64] = 8'd113; win[1][ 71: 64] = 8'd8  ; win[2][ 71: 64] = 8'd1  ; win[3][ 71: 64] = 8'd211;
183     win[0][ 79: 72] = 8'd0  ; win[1][ 79: 72] = 8'd0  ; win[2][ 79: 72] = 8'd0  ; win[3][ 79: 72] = 8'd0  ;
184     win[0][ 87: 80] = 8'd0  ; win[1][ 87: 80] = 8'd0  ; win[2][ 87: 80] = 8'd0  ; win[3][ 87: 80] = 8'd0  ;
185     win[0][ 95: 88] = 8'd0  ; win[1][ 95: 88] = 8'd0  ; win[2][ 95: 88] = 8'd0  ; win[3][ 95: 88] = 8'd0  ;
186     win[0][103: 96] = 8'd0  ; win[1][103: 96] = 8'd0  ; win[2][103: 96] = 8'd0  ; win[3][103: 96] = 8'd0  ;
187     win[0][111:104] = 8'd0  ; win[1][111:104] = 8'd0  ; win[2][111:104] = 8'd0  ; win[3][111:104] = 8'd0  ;
188     win[0][119:112] = 8'd0  ; win[1][119:112] = 8'd0  ; win[2][119:112] = 8'd0  ; win[3][119:112] = 8'd0  ;
189     win[0][127:120] = 8'd0  ; win[1][127:120] = 8'd0  ; win[2][127:120] = 8'd0  ; win[3][127:120] = 8'd0  ;
190
```

Input image

- The input image is 128x128x1
  - Zero padding (S/W)
    - To generate an output image (128x128x4)
    - Set boundary pixels to zeros

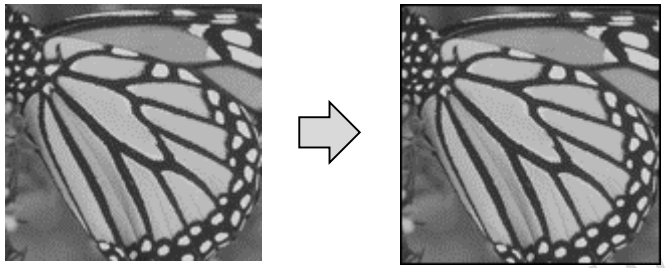


0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

```
157    // Read the input file to memory
158    initial begin
159        $readmemh(INFILE, in_img ,0,FRAME_SIZE-1);
160    end
```

Sliding windows and mapping

- Generate din
  - A window of a filter is sliding
    - Left to right, top to bottom
  - In this case, only 9 multipliers are actually used.



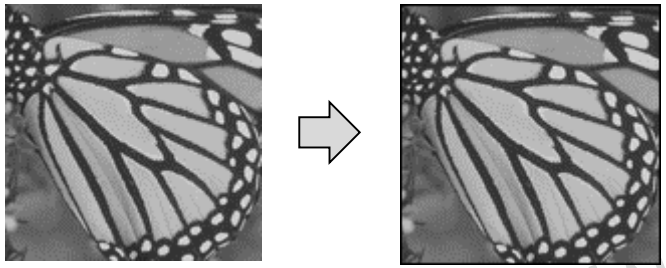
0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

out(0,0,k)[1]

[1] In C or Verilog, loop indexes are started from 0. In Matlab, they start from 1.

Sliding windows and mapping

- Generate din
  - A window of a filter is sliding
    - Left to right, top to bottom
  - In this case, only 9 multipliers are actually used.

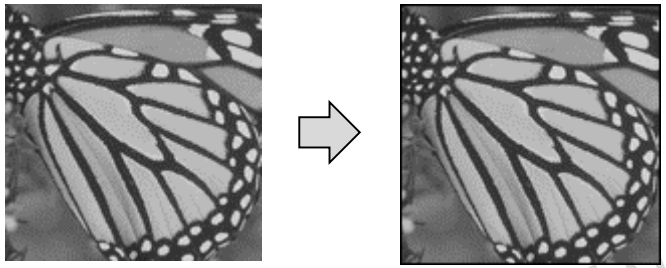


0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

out(0,1,k)

Sliding windows and mapping

- Generate din
  - A window of a filter is sliding
    - Left to right, top to bottom
  - In this case, only 9 multipliers are actually used.

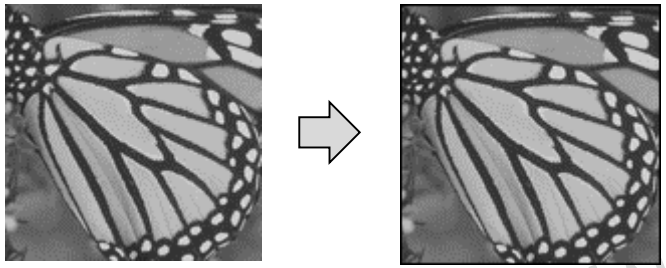


0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

out(0,2,k)

Sliding windows and mapping

- Generate din
  - A window of a filter is sliding
    - Left to right, top to bottom
  - In this case, only 9 multipliers are actually used.



0	0	0	0	0	0	0	0
0	42	69	91	99	106	108	111
0	105	42	56	84	106	113	112
0	72	43	42	68	109	112	104

out(0,3,k)

Test bench (cnv_tb.v)

Parameters

Signals for Convolutional kernels

```
1  `timescale 1ns / 1ps
2
3  module cnv_tb;
4      parameter W_SIZE  = 12;                    // Max 4K QHD (3840x1920).
5      parameter W_FRAME_SIZE  = 2 * W_SIZE + 1;  // Max 4K QHD (3840x1920).
6      parameter W_DELAY = 12;
7      parameter WIDTH   = 128;
8      parameter HEIGHT  = 128;
9      parameter INFILE    = "C:/AIX2022/aix2022/sim/input_data/butterfly_08bit.hex";
10     parameter OUTFILE0  = "C:/AIX2022/aix2022/sim/output/fmap00.bmp";
11     parameter OUTFILE1  = "C:/AIX2022/aix2022/sim/output/fmap01.bmp";
12     parameter OUTFILE2  = "C:/AIX2022/aix2022/sim/output/fmap02.bmp";
13     parameter OUTFILE3  = "C:/AIX2022/aix2022/sim/output/fmap03.bmp";
14     parameter FRAME_SIZE = WIDTH * HEIGHT;
15     parameter VSYNC_DELAY = 200;
16     parameter HSYNC_DELAY = 160;
17     localparam FRAME_SIZE_W = $clog2(FRAME_SIZE);
18     reg [7:0] in_img [0:FRAME_SIZE-1]; // Input image
19     reg clk;
20     reg rstn;
21     // MACs, Conv kernels
22     reg vld_i;
23     reg [127:0] win[0:3];
24     reg [127:0] din;
25     wire[ 19:0] acc_o[0:3];
26     wire        vld_o[0:3];
27     wire frame_done[0:3];
```

Check if the directories are correct!!!

Generate din

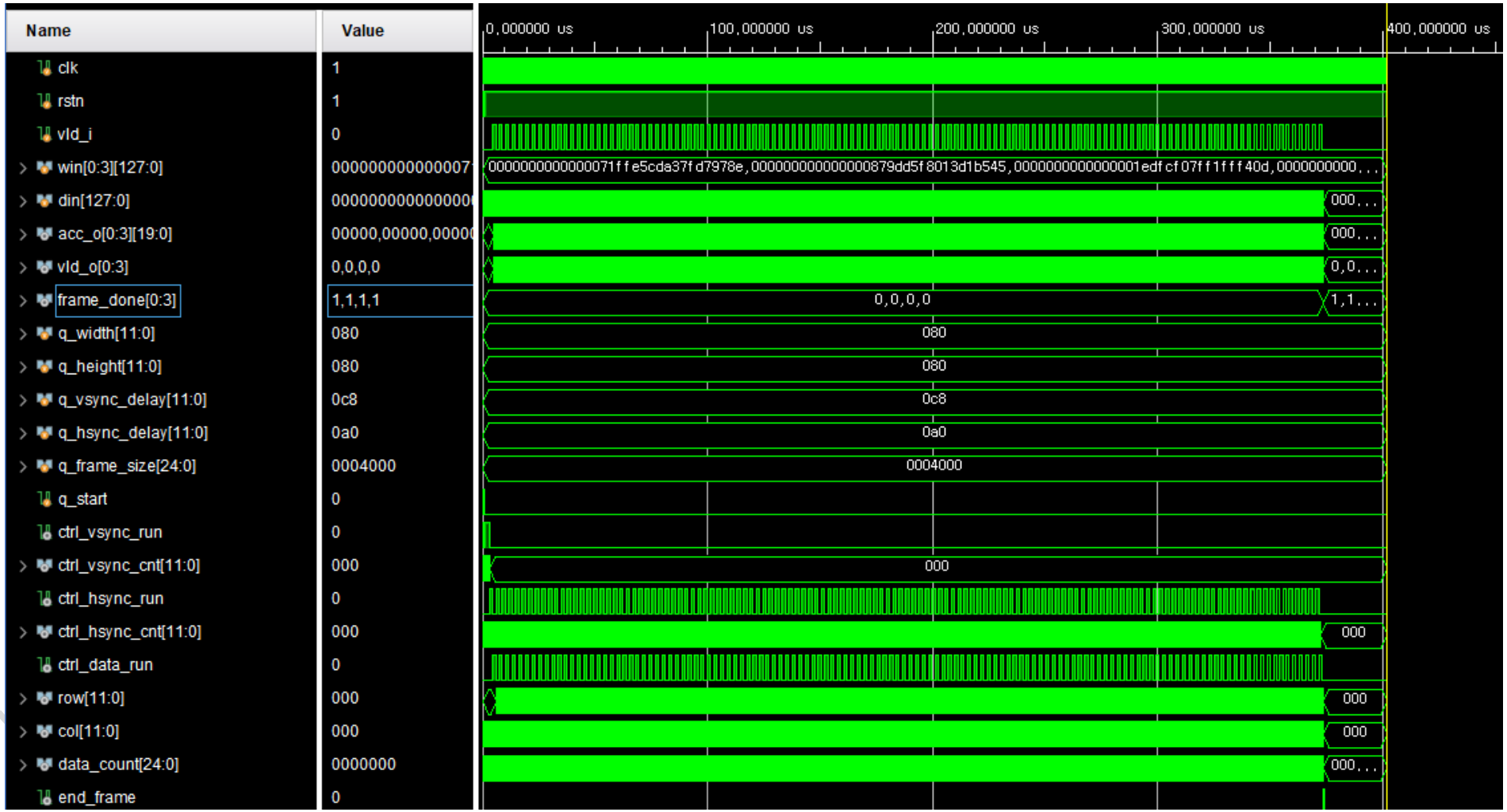
Boundary checking

Generate din, vld_i

```
201     // Generate din
202     wire is_first_row = (row == 0) ? 1'b1: 1'b0;
203     wire is_last_row  = (row == HEIGHT-1) ? 1'b1: 1'b0;
204     wire is_first_col = (col == 0) ? 1'b1: 1'b0;
205     wire is_last_col  = (col == WIDTH-1) ? 1'b1 : 1'b0;
206
207     always@(*) begin
208         din = 128'd0;
209         vld_i = 1'b0;
210         if(ctrl_data_run) begin
211             vld_i = 1'b1;
212             din[ 7: 0] = (is_first_row | is_first_col) ? 8'd0 : in_img[(row-1) * WIDTH + (col-1)];
213             din[15: 8] = (is_first_row              ) ? 8'd0 : in_img[(row-1) * WIDTH +  col   ];
214             din[23:16] = (is_first_row | is_last_col ) ? 8'd0 : in_img[(row-1) * WIDTH + (col+1)];
215             din[31:24] = (              is_first_col) ? 8'd0 : in_img[ row    * WIDTH + (col-1)];
216             din[39:32] =                               in_img[ row    * WIDTH +  col   ];
217             din[47:40] = (              is_last_col ) ? 8'd0 : in_img[ row    * WIDTH + (col+1)];
218             din[55:48] = (is_last_row | is_first_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col-1)];
219             din[63:56] = (is_last_row | is_first_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col-1)];
220             din[71:64] = (is_last_row | is_first_col ) ? 8'd0 : in_img[(row+1) * WIDTH + (col+1)];
221         end
222     end
```

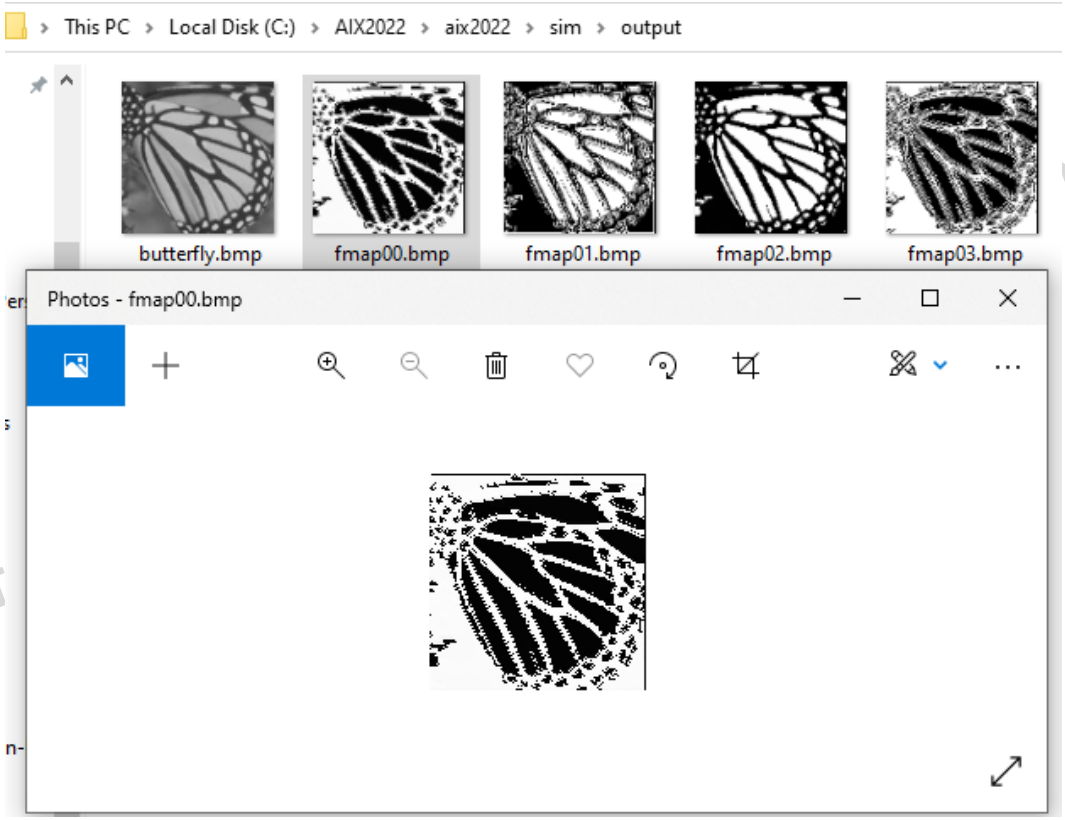
Do computation when ctrl_data_run = 1

Waveform



Output results

- The output images are stored at the predefined directories
  - Update directories based on your environment



Incoming lectures

- System integration
  - Use buffers
  - Use DMA