차세대 반도체 디지털 혁신공유대학 사업단

POLARIS

System Integration I

Sliding Windows, CNN Controller

2024.03.22 (Friday)

Objective

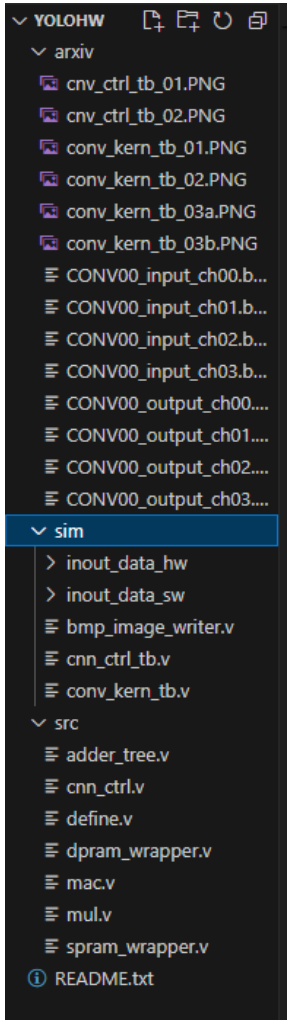
- In this tutorial, we show you
  - Load input data (input maps and filters) generated by the software (skeleton)
  - Print out the filters for visualization
  - Use four MACs module to do partial convolution
  - Save the input features and computed outputs in an image
  - Work on a simple controller

Code structure

- Top level
  - /src Source code
  - /sim Testbench and data for simulation
    - sim/conv_kern_tb.v: Load data, do convolution, save inputs and outputs
    - sim/cnn_ctrl_tb.v: A simple controller to generate a loop (row, col indices)
  - /arxiv Screen-captured results

Simulation

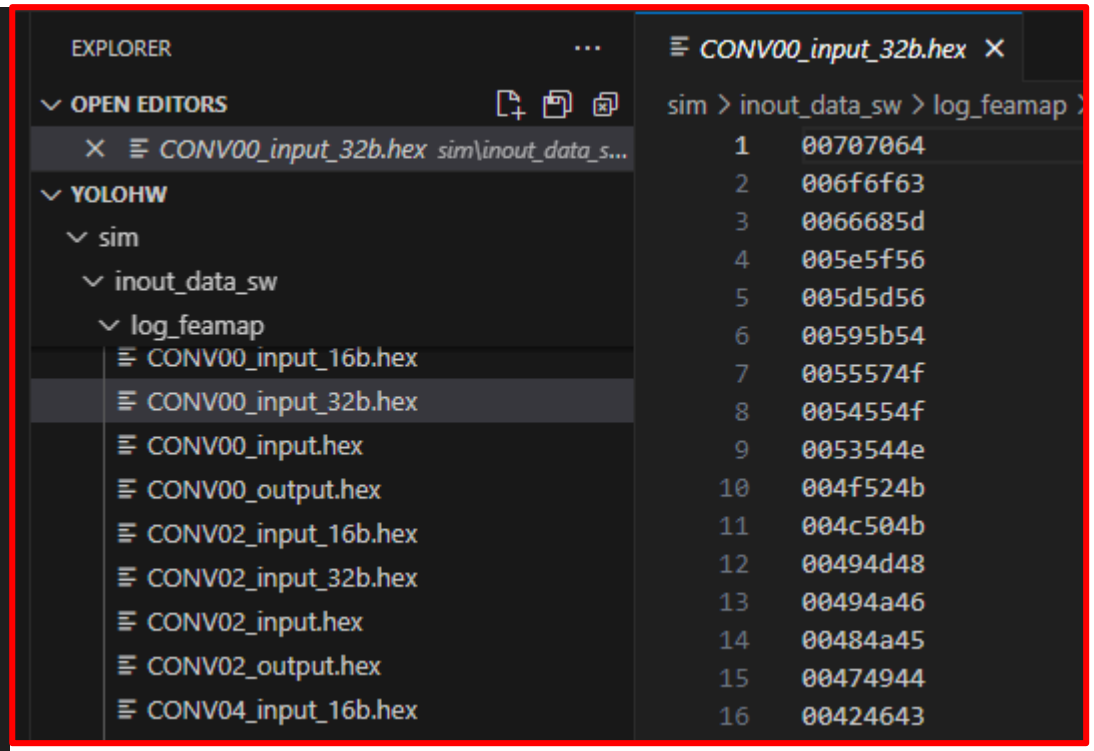
- /sim/inout_data_sw/log_feamap Feature maps from **SW simulation** (Hex format)
- /sim/inout_data_sw/log_param Weight maps from **SW simulation** (Hex format)
- /sim/inout_data_hw Output for **HW simulation**



conv_kern_tb.v: Load inputs from files

- Load **input features** generated from **Software**

```
23  //------------------------------------------------------------------
24  // Load input feature maps and parameters
25  //------------------------------------------------------------------
26  reg  [IFM_WORD_SIZE_32-1:0] in_img[0:IFM_DATA_SIZE_32-1];  // Infmap
27  reg  [IFM_WORD_SIZE_32-1:0] filter[0:WGT_DATA_SIZE   -1];   // Filter
28  reg  preload;
29
30
31  // Load memory from file
32  integer i;
33  initial begin: PROC_SimmemLoad
34
35      // Inputs
36      for (i = 0; i< IFM_DATA_SIZE_32; i=i+1) begin
37          in_img[i] = 0;
38      end
39      $display ("Loading input feature maps from file: %s", IFM_FILE_32);
40      $readmemh(IFM_FILE_32, in_img);
41
42      // Filters
43      for (i = 0; i< WGT_DATA_SIZE; i=i+1) begin
44          filter[i] = 0;
45      end
46      $display ("Loading input feature maps from file: %s", WGT_FILE);
47      $readmemh(WGT_FILE, filter);
48  end
```

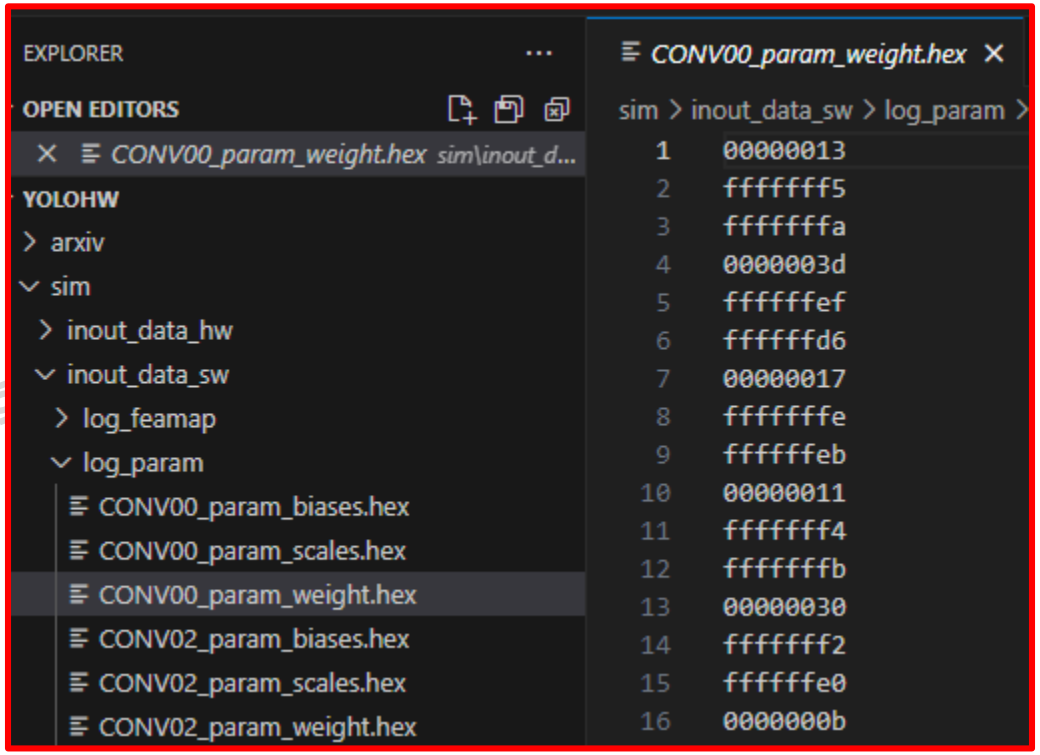


- Each line has 32 bits
- Color format: {0, ch2, ch1, ch0} for CONV00

conv_kern_tb.v: Load inputs from files

- Load **filters** generated from **Software**

```
23  //------------------------------------------------------------------
24  // Load input feature maps and parameters
25  //------------------------------------------------------------------
26  reg  [IFM_WORD_SIZE_32-1:0] in_img[0:IFM_DATA_SIZE_32-1];  // Infmap
27  reg  [IFM_WORD_SIZE_32-1:0] filter[0:WGT_DATA_SIZE   -1];   // Filter
28  reg  preload;
29
30
31  // Load memory from file
32  integer i;
33  initial begin: PROC_SimmemLoad
34
35      // Inputs
36      for (i = 0; i< IFM_DATA_SIZE_32; i=i+1) begin
37          in_img[i] = 0;
38      end
39      $display ("Loading input feature maps from file: %s", IFM_FILE_32);
40      $readmemh(IFM_FILE_32, in_img);
41
42      // Filters
43      for (i = 0; i< WGT_DATA_SIZE; i=i+1) begin
44          filter[i] = 0;
45      end
46      $display ("Loading input feature maps from file: %s", WGT_FILE);
47      $readmemh(WGT_FILE, filter);
48  end
```



- Each line has 32 bits

=> **8 LSB bits are used for one filter coefficient**

conv_kern_tb.v: Print out the filter

```
50  //------------------------------------------------------------------------
51  // Test vector
52  //------------------------------------------------------------------------
53  integer j;
54  integer row, col;
55  initial begin
56      // Initialization
57      rstn = 1'b0;          // Reset, low active
58      preload = 1'b0;
59      ctrl_data_run  = 1'b0;
60      // Reset and check preloaded filters
61      #(4*CLK_PERIOD) rstn = 1'b1;
62      #(100*CLK_PERIOD)
63          @(posedge clk)
64              preload = 1'b1;
65      // Show the filter
66      #(100*CLK_PERIOD)
67          @(posedge clk)
68          for (j=0; j < No; j=j+1) begin
69              $display("Filter och=%02d: \n",j);
70              for(i = 0; i < 3; i = i + 1) begin
71                  $display("%d\t%d\t%d",
72                      $signed(filter[(j*Fx*Fy*Ni) + (3*i  )][7:0]),
73                      $signed(filter[(j*Fx*Fy*Ni) + (3*i+1)][7:0]),
74                      $signed(filter[(j*Fx*Fy*Ni) + (3*i+2)][7:0]));
75              end
76              $display("\n");
77          end
```

```
VSIM 10> run 1ms
# Loading input feature maps from file: C:/yolohw/sim/inout_data_sw/log_feamap/CONV00_input_32b.hex
# Loading input feature maps from file: C:/yolohw/sim/inout_data_sw/log_param/CONV00_param_weight.hex
# Filter och= 0:
#
#   19   -11    -6
#   61   -17   -42
#   23    -2   -21
#
#
# Filter och= 1:
#
#   12   -15   -17
#    5     5   -18
#    6     2   -17
#
#
# Filter och= 2:
#
#    3     6    14
#   -8     1    21
#    5   -22     9
#
#
# Filter och= 3:
#
#   24    17    12
#   11    23    24
#   -4     9    -1
#
#
# Filter och= 4:
#
#  -10    -2     4
#   -9    -8    -9
#    8     0    10
#
#
# Filter och= 5:
#
#   25   -10    22
#   -7   -32     9
#   -6   -21    -8
#
#
# Filter och= 6:
#
#  -70   -35   -12
#   47    28     0
#   48    16     4
#
#
# Filter och= 7:
```

con_kern_tb.v

- Generate row, col ind

```
83      // ****************************************
84      // Loop for convolutions
85      // ****************************************
86      //{{{
87      #(100*CLK_PERIOD)
88          for(row = 0; row < IFM_HEIGHT; row = row + 1)   begin
89              @(posedge clk)
90                  ctrl_data_run  = 0;
91              #(100*CLK_PERIOD)
92              for (col = 0; col < IFM_WIDTH; col = col + 1) begin
93                  @(posedge clk)
94                      ctrl_data_run  = 1;
95              end
96          end
97      @(posedge clk)
98              ctrl_data_run = 1'b0;
99      //}}}
100     #(100*CLK_PERIOD)
101         @(posedge clk) $stop;
```

conv_kern_tb.v: Use MACs

- Use **four MAC groups** to do convolution
  - Each MAC has 16 multipliers (see Tutorial 5)

  ⇒ 64 multipliers

- Input/Ouput
  - Inputs:
    - vld_i Input valid signal
    - win Filters => four filters
    - din Window data
  - Output
    - vld_o Output valid signal
    - acc_o Accumulated results ($\sum_{i=0}^{15} w_i * d_i$)

```
143  //----------------------------------------
144  // DUT: MACs
145  //----------------------------------------
146  mac u_mac_00(
147  ./*input          */clk  (clk     ),
148  ./*input          */rstn (rstn    ),
149  ./*input          */vld_i(vld_i   ),
150  ./*input [127:0] */win  (win[0]  ),
151  ./*input [127:0] */din  (din     ),
152  ./*output[ 19:0] */acc_o(acc_o[0]),
153  ./*output         */vld_o(vld_o[0])
154  );
155  mac u_mac_01(
156  ./*input          */clk  (clk     ),
157  ./*input          */rstn (rstn    ),
158  ./*input          */vld_i(vld_i   ),
159  ./*input [127:0] */win  (win[1]  ),
160  ./*input [127:0] */din  (din     ),
161  ./*output[ 19:0] */acc_o(acc_o[1]),
162  ./*output         */vld_o(vld_o[1])
163  );
164  mac u_mac_02(
165  ./*input          */clk  (clk     ),
166  ./*input          */rstn (rstn    ),
167  ./*input          */vld_i(vld_i   ),
168  ./*input [127:0] */win  (win[2]  ),
169  ./*input [127:0] */din  (din     ),
170  ./*output[ 19:0] */acc_o(acc_o[2]),
171  ./*output         */vld_o(vld_o[2])
172  );
173  mac u_mac_03(
174  ./*input          */clk  (clk     ),
175  ./*input          */rstn (rstn    ),
176  ./*input          */vld_i(vld_i   ),
177  ./*input [127:0] */win  (win[3]  ),
178  ./*input [127:0] */din  (din     ),
179  ./*output[ 19:0] */acc_o(acc_o[3]),
180  ./*output         */vld_o(vld_o[3])
181  );
```

conv_kern_tb.v: Generate row, col, ctrl_data_run

- Use two loops to generate row, col, and ctrl_data_run
  - row, col are the spatial indices of accumulated results
  - ctrl_data_run: mark when we do calculation (vld_i = 1)

```
83      // ****************************************
84      // Loop for convolutions
85      // ****************************************
86      //{{{
87      #(100*CLK_PERIOD)
88          for(row = 0; row < IFM_HEIGHT; row = row + 1)   begin
89              @(posedge clk)
90                  ctrl_data_run  = 0;
91              #(100*CLK_PERIOD)
92              for (col = 0; col < IFM_WIDTH; col = col + 1) begin
93                  @(posedge clk)
94                      ctrl_data_run  = 1;
95              end
96          end
97      @(posedge clk)
98              ctrl_data_run = 1'b0;
99      //}}}
100     #(100*CLK_PERIOD)
101         @(posedge clk) $stop;
```

conv_kern_tb.v: Generate vld_i, win, din

Generate a request to do computation
- Use ctrl_run_data

Generate din from **a window 3x3** in the input feature map (in_img)
- Use **row, col** to form the window

Generate **four 3x3 filters**

```
104  // Generate din, win
105  wire is_first_row = (row == 0) ? 1'b1: 1'b0;
106  wire is_last_row  = (row == IFM_HEIGHT-1) ? 1'b1: 1'b0;
107  wire is_first_col = (col == 0) ? 1'b1: 1'b0;
108  wire is_last_col  = (col == IFM_WIDTH-1) ? 1'b1 : 1'b0;
109
110  always@(*) begin
111      vld_i = 0;
112      din = 128'd0;
113      win[0] = 0;
114      win[1] = 0;
115      win[2] = 0;
116      win[3] = 0;
117      if(ctrl_data_run) begin
118          vld_i = 1;
119          // Tiled IFM data
120          din[ 7: 0] = (is_first_row || is_first_col) ? 8'd0 : in_img[(row-1) * IFM_WIDTH + (col-1)];
121          din[15: 8] = (is_first_row                 ) ? 8'd0 : in_img[(row-1) * IFM_WIDTH +  col   ];
122          din[23:16] = (is_first_row || is_last_col ) ? 8'd0 : in_img[(row-1) * IFM_WIDTH + (col+1)];
123          din[31:24] = (                is_first_col) ? 8'd0 : in_img[ row    * IFM_WIDTH + (col-1)];
124          din[39:32] =                                         in_img[ row    * IFM_WIDTH +  col   ];
125          din[47:40] = (                is_last_col ) ? 8'd0 : in_img[ row    * IFM_WIDTH + (col+1)];
126          din[55:48] = (is_last_row ||  is_first_col) ? 8'd0 : in_img[(row+1) * IFM_WIDTH + (col-1)];
127          din[63:56] = (is_last_row                  ) ? 8'd0 : in_img[(row+1) * IFM_WIDTH + (col-1)];
128          din[71:64] = (is_last_row ||  is_last_col ) ? 8'd0 : in_img[(row+1) * IFM_WIDTH + (col+1)];
129          // Tiled Filters
130          for(j = 0; j < 4; j=j+1) begin  // Four sets <=> Four output channels
131              win[j][ 7: 0] = filter[(j*Fx*Fy*Ni)    ][7:0];
132              win[j][15: 8] = filter[(j*Fx*Fy*Ni) + 1][7:0];
133              win[j][23:16] = filter[(j*Fx*Fy*Ni) + 2][7:0];
134              win[j][31:24] = filter[(j*Fx*Fy*Ni) + 3][7:0];
135              win[j][39:32] = filter[(j*Fx*Fy*Ni) + 4][7:0];
136              win[j][47:40] = filter[(j*Fx*Fy*Ni) + 5][7:0];
137              win[j][55:48] = filter[(j*Fx*Fy*Ni) + 6][7:0];
138              win[j][63:56] = filter[(j*Fx*Fy*Ni) + 7][7:0];
139              win[j][71:64] = filter[(j*Fx*Fy*Ni) + 8][7:0];
140          end
141      end
142  end
```

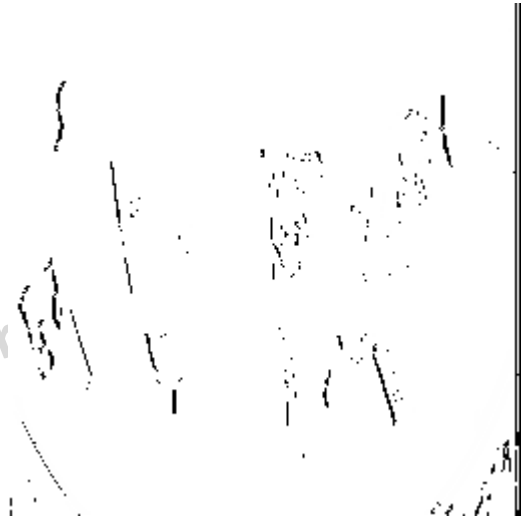
conv_kern_tb.v: Checking inputs

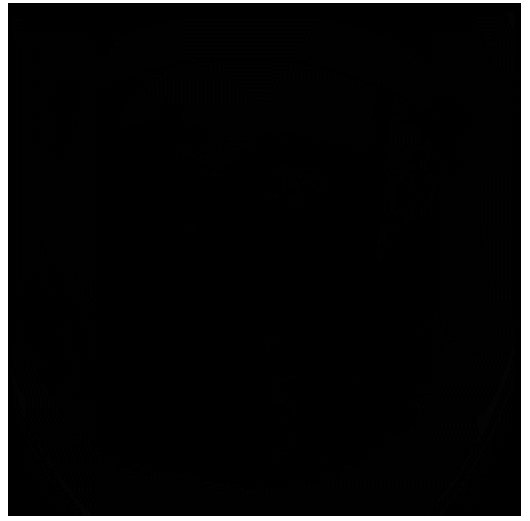
- We can save the input in an BMP image

```
256  bmp_image_writer #(.OUTFILE(CONV_INPUT_IMG00),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
257  u_ifm_img_ch0(
258      ./*input            */clk       (clk                        ),
259      ./*input            */rstn      (rstn                       ),
260      ./*input [WI-1:0]   */din       (in_img[dbg_pix_idx][7:0]   ),
261      ./*input            */vld       (dbg_write_image            ),
262      ./*output reg       */frame_done(                           )
263  );
264  bmp_image_writer #(.OUTFILE(CONV_INPUT_IMG01),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
265  u_ifm_img_ch1(
266      ./*input            */clk       (clk                        ),
267      ./*input            */rstn      (rstn                       ),
268      ./*input [WI-1:0]   */din       (in_img[dbg_pix_idx][15:8]  ),
269      ./*input            */vld       (dbg_write_image            ),
270      ./*output reg       */frame_done(                           )
271  );
272  bmp_image_writer #(.OUTFILE(CONV_INPUT_IMG02),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
273  u_ifm_img_ch2(
274      ./*input            */clk       (clk                        ),
275      ./*input            */rstn      (rstn                       ),
276      ./*input [WI-1:0]   */din       (in_img[dbg_pix_idx][23:16]),
277      ./*input            */vld       (dbg_write_image            ),
278      ./*output reg       */frame_done(                           )
279  );
280  bmp_image_writer #(.OUTFILE(CONV_INPUT_IMG03),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
281  u_ifm_img_ch3(
282      ./*input            */clk       (clk                        ),
283      ./*input            */rstn      (rstn                       ),
284      ./*input [WI-1:0]   */din       (in_img[dbg_pix_idx][31:24]),
285      ./*input            */vld       (dbg_write_image            ),
286      ./*output reg       */frame_done(                           )
287  );
```

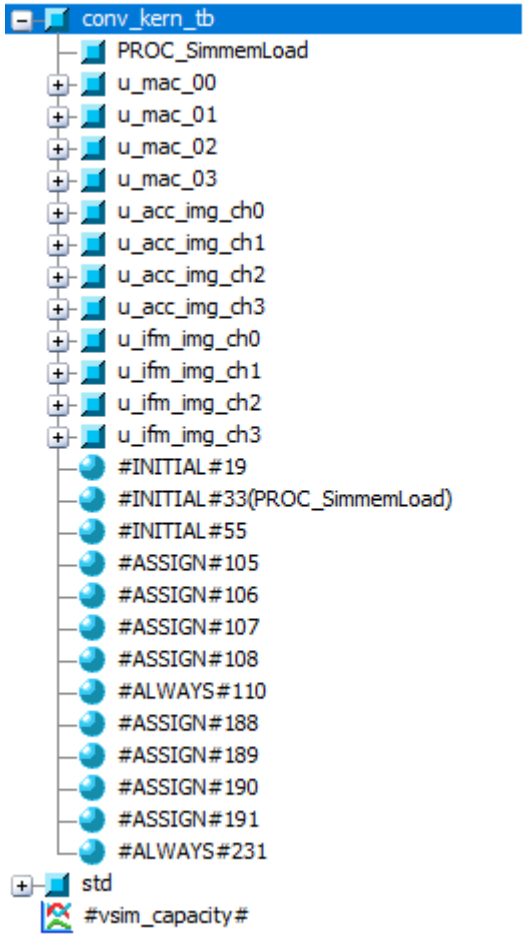
conv_kern_tb.v: Checking **outputs**

```
186   // Output feature maps
187   //{{{
188   wire [7:0] conv_out_ch00 = acc_o[0][19:12]; // Descaling: * 1/2^11
189   wire [7:0] conv_out_ch01 = acc_o[1][19:12]; // Descaling: * 1/2^11
190   wire [7:0] conv_out_ch02 = acc_o[2][19:12]; // Descaling: * 1/2^11
191   wire [7:0] conv_out_ch03 = acc_o[3][19:12]; // Descaling: * 1/2^11
192
193   bmp_image_writer #(.OUTFILE(CONV_OUTPUT_IMG00),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
194   u_acc_img_ch0(
195       ./*input            */clk       (clk          ),
196       ./*input            */rstn      (rstn         ),
197       ./*input [WI-1:0]   */din       (conv_out_ch00),
198       ./*input            */vld       (vld_o[0]     ),
199       ./*output reg       */frame_done(             )
200   );
201   bmp_image_writer #(.OUTFILE(CONV_OUTPUT_IMG01),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
202   u_acc_img_ch1(
203       ./*input            */clk       (clk          ),
204       ./*input            */rstn      (rstn         ),
205       ./*input [WI-1:0]   */din       (conv_out_ch01),
206       ./*input            */vld       (vld_o[0]     ),
207       ./*output reg       */frame_done(             )
208   );
209   bmp_image_writer #(.OUTFILE(CONV_OUTPUT_IMG02),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
210   u_acc_img_ch2(
211       ./*input            */clk       (clk          ),
212       ./*input            */rstn      (rstn         ),
213       ./*input [WI-1:0]   */din       (conv_out_ch02),
214       ./*input            */vld       (vld_o[0]     ),
215       ./*output reg       */frame_done(             )
216   );
217   bmp_image_writer #(.OUTFILE(CONV_OUTPUT_IMG03),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
218   u_acc_img_ch3(
219       ./*input            */clk       (clk          ),
220       ./*input            */rstn      (rstn         ),
221       ./*input [WI-1:0]   */din       (conv_out_ch03),
222       ./*input            */vld       (vld_o[0]     ),
223       ./*output reg       */frame_done(             )
224   );
225   //}}}
```



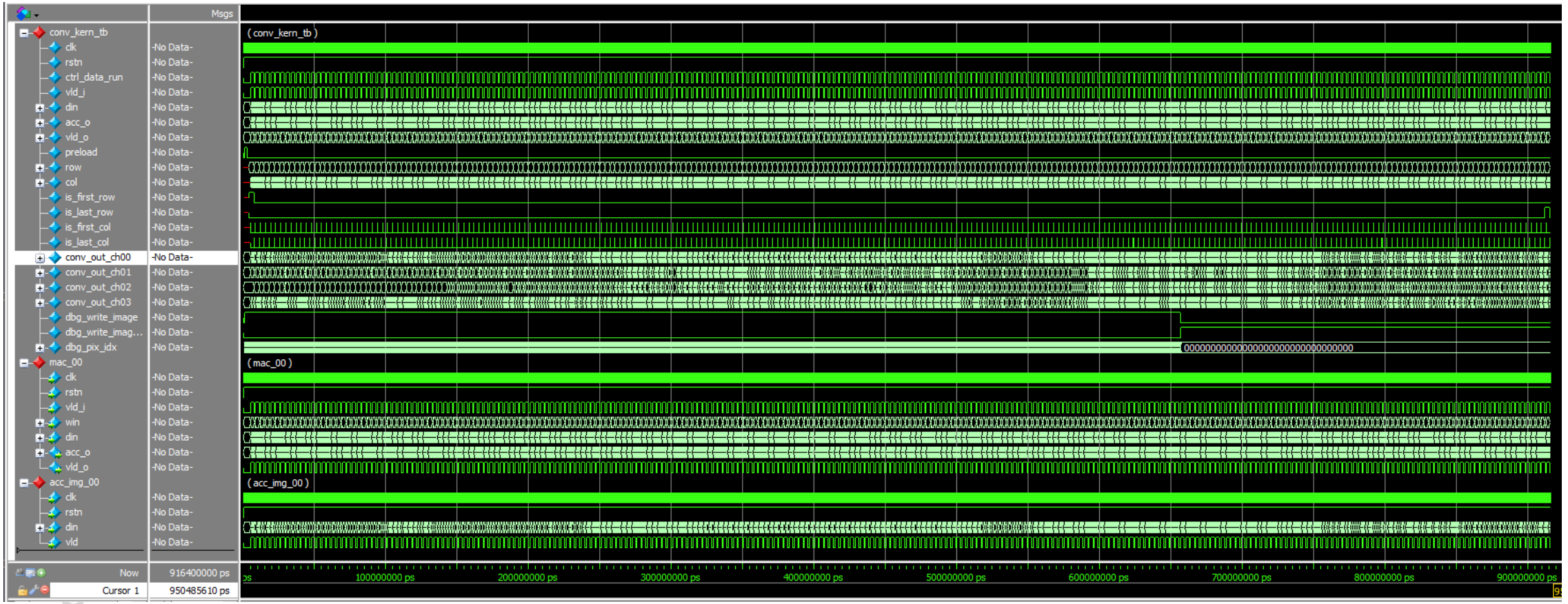


conv_kern_tb.v: Simulation time = 1ms




```
VSIM 10> run 1ms
# Loading input feature maps from file: C:/yolohw/sim/inout_data_sw/log_feamap/CONV00_input_32b.hex
# Loading input feature maps from file: C:/yolohw/sim/inout_data_sw/log_param/CONV00_param_weight.hex
# Filter och= 0:
#
#   19   -11     -6
#   61   -17    -42
#   23    -2    -21
#
#
# Filter och= 1:
#
#   12   -15    -17
#    5     5    -18
#    6     2    -17
#
#
# Filter och= 2:
#
#    3     6     14
#   -8     1     21
#    5   -22      9
#
#
# Filter och= 3:
#
#   24    17     12
#   11    23     24
#   -4     9     -1
#
#
# Filter och= 4:
#
#  -10    -2      4
#   -9    -8     -9
#    8     0     10
#
#
# Filter och= 5:
#
#   25   -10     22
#   -7   -32      9
#   -6   -21     -8
#
#
# Filter och= 6:
#
#  -70   -35    -12
#   47    28      0
#   48    16      4
#
#
# Filter och= 7:
#
#   53    29     30
#   25    -5     -1
#  -62   -56     -9
#
#
# Filter och=12:
#
#    6    53     26
#   23    70     46
#   -9    31     11
#
#
# Filter och=13:
#
#   20    45    -11
#   -7   -32    -25
#    4   -28    -17
#
#
# Filter och=14:
#
#    7   -65     10
#  -10     5      3
#    1   -22      2
#
#
# Filter och=15:
#
#   -8    -9    -18
#    0     2     -2
#    6    13      0
#
#
# Saving output images to file: C:/yolohw/sim/inout_data_hw/CONV00_input_ch03.bmp
# Saving output images to file: C:/yolohw/sim/inout_data_hw/CONV00_input_ch02.bmp
# Saving output images to file: C:/yolohw/sim/inout_data_hw/CONV00_input_ch01.bmp
# Saving output images to file: C:/yolohw/sim/inout_data_hw/CONV00_input_ch00.bmp
# Saving output images to file: C:/yolohw/sim/inout_data_hw/CONV00_output_ch03.bmp
# Saving output images to file: C:/yolohw/sim/inout_data_hw/CONV00_output_ch02.bmp
# Saving output images to file: C:/yolohw/sim/inout_data_hw/CONV00_output_ch01.bmp
# Saving output images to file: C:/yolohw/sim/inout_data_hw/CONV00_output_ch00.bmp
# ** Note: $stop    : C:/yolohw/sim/conv_kern_tb.v(101)
#    Time: 916400 ns  Iteration: 1  Instance: /conv_kern_tb
# Break in Module conv_kern_tb at C:/yolohw/sim/conv_kern_tb.v line 101
```

conv_kern_tb.v: Simulation time = 1ms



Objective

- In this tutorial, we show you
  - Load input data (input maps and filters) generated by the software (skeleton)
  - Print out the filters for visualization
  - Use four MACs module to do partial convolution
  - Save the input features and computed outputs in an image
  - Work on a simple controller

Motivation

- A pseudo code to generate an computation order for output calculation
  - 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

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)

conv_kern_tb.v | cnn_ctrl.v

src > cnn_ctrl.v

```
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

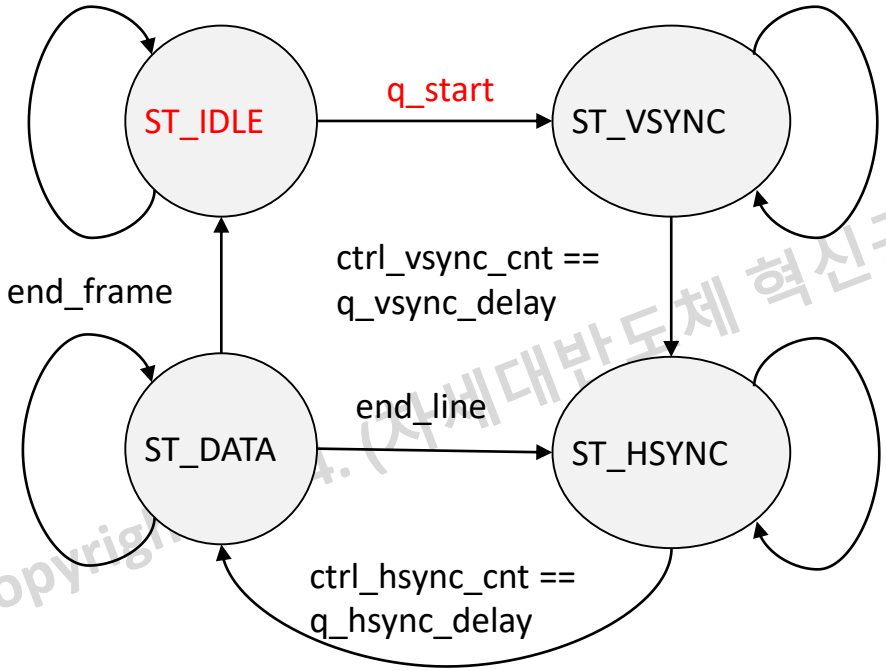
```
always @ (posedge clk, negedge rstn)
begin
   if(~rstn) begin
      cstate <= ST_IDLE;
   end
   else begin
      cstate <= nstate;
   end
end

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)

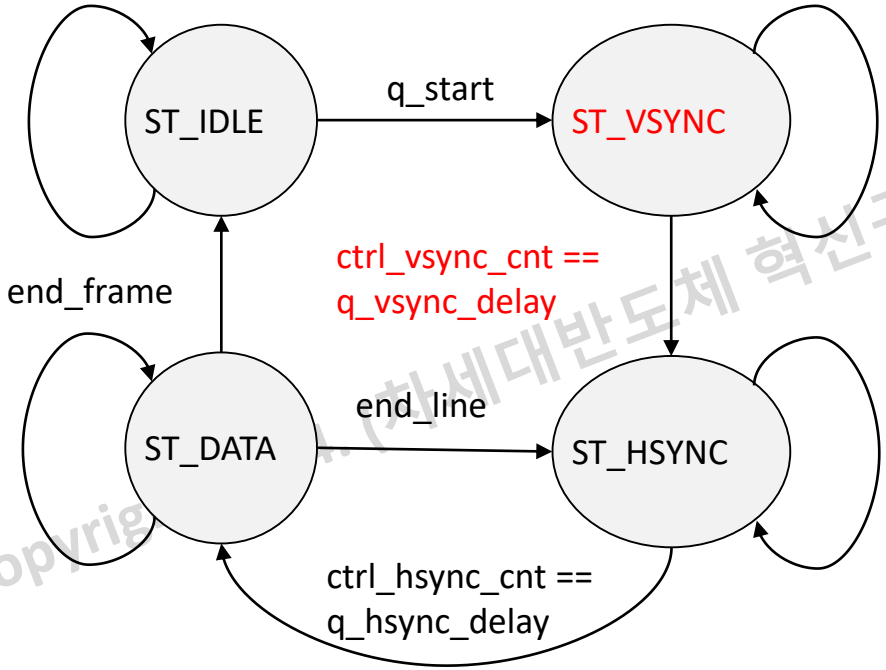
- ST_IDLE:
  - FSM is initialized at ST_IDLE
  - When "start" goes HIGH, FSM moves to ST_VSYNC.




```
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

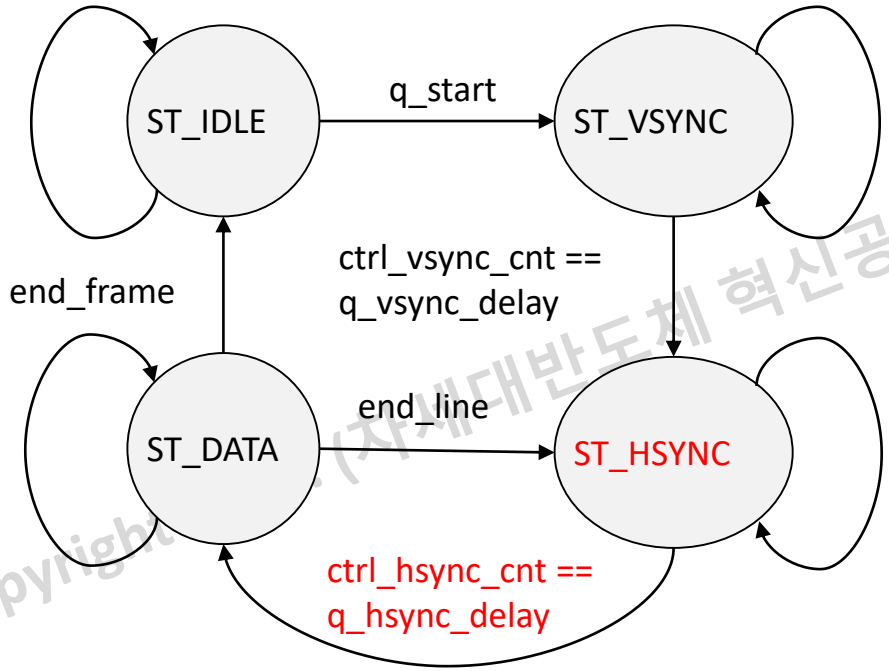
- ST_VSYNC: Frame synchronization
  - Start up for a frame
  - There is an VSYNC counter.
  - When the counter reaches to q_vsync_delay, FSM moves to ST_HSYNC.




```
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

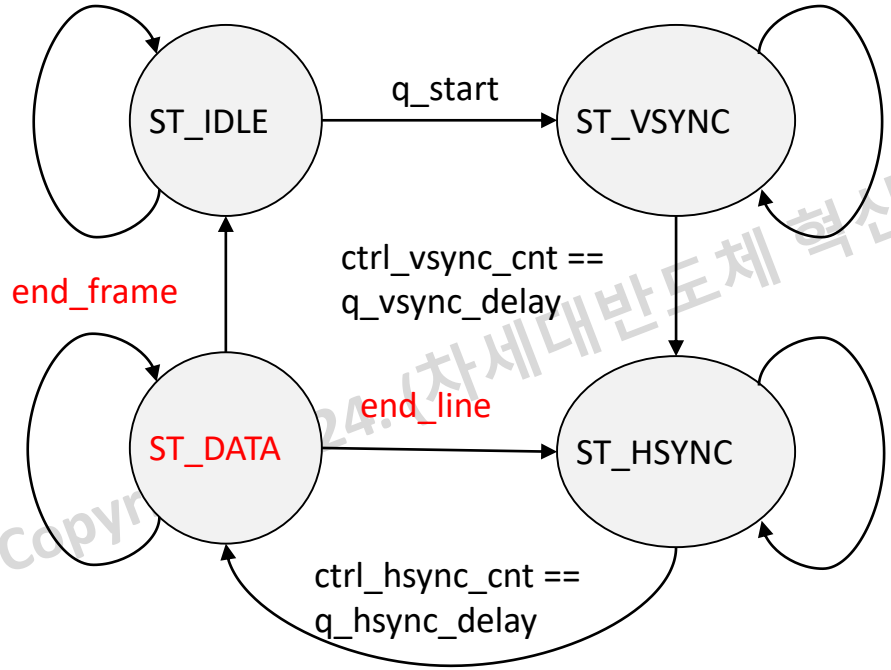
- ST_HSYNC: Line synchronization
  - There is an HSYNC counter.
  - When the counter reaches to q_hsync_delay, FSM moves to ST_DATA.




```
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

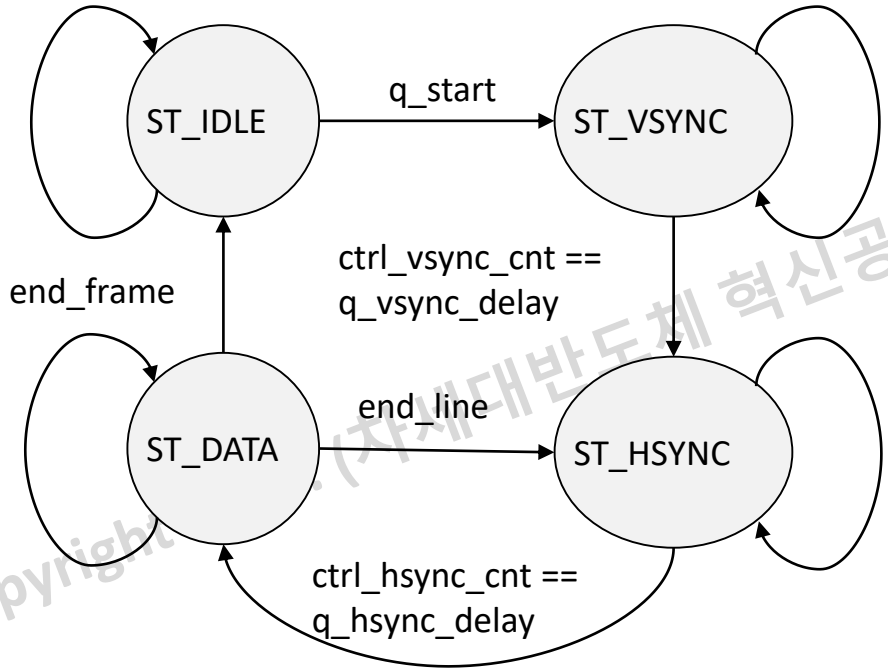
- ST_DATA: Sending pixel data
  - There are two counters
  - Line data counter: if it reaches to end of line, FSM moves to ST_HSYNC.
  - Frame data counter: if it reaches to end of frame, FSM moves to ST_IDLE.




```
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




```
75  always @(*) begin
76      case(cstate)
77          ST_IDLE: begin
78              if(q_start)
79                  nstate = ST_VSYNC;
80              else
81                  nstate = ST_IDLE;
82          end
83          ST_VSYNC: begin
84              if(ctrl_vsync_cnt == q_vsync_delay)
85                  nstate = ST_HSYNC;
86              else
87                  nstate = ST_VSYNC;
88          end
89          ST_HSYNC: begin
90              if(ctrl_hsync_cnt == q_hsync_delay)
91                  nstate = ST_DATA;
92              else
93                  nstate = ST_HSYNC;
94          end
95          ST_DATA: begin
96              if(end_frame)       //end of frame
97                  nstate = ST_IDLE;
98              else begin
99                  if(col == q_width-1)    //end of line
100                 nstate = ST_HSYNC;
101             else
102                 nstate = ST_DATA;
103             end
104         end
105         default: nstate = ST_IDLE;
106     endcase
107 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

conv_kern_tb.v | cnn_ctrl_tb.v

sim > cnn_ctrl_tb.v

```
1  `timescale 1ns / 1ps
2
3  module cnn_ctrl_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     = 256;
8  parameter HEIGHT    = 256;
9  parameter FRAME_SIZE = WIDTH * HEIGHT;
10 parameter VSYNC_DELAY = 100;
11 parameter HSYNC_DELAY = 100;
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 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  );
```

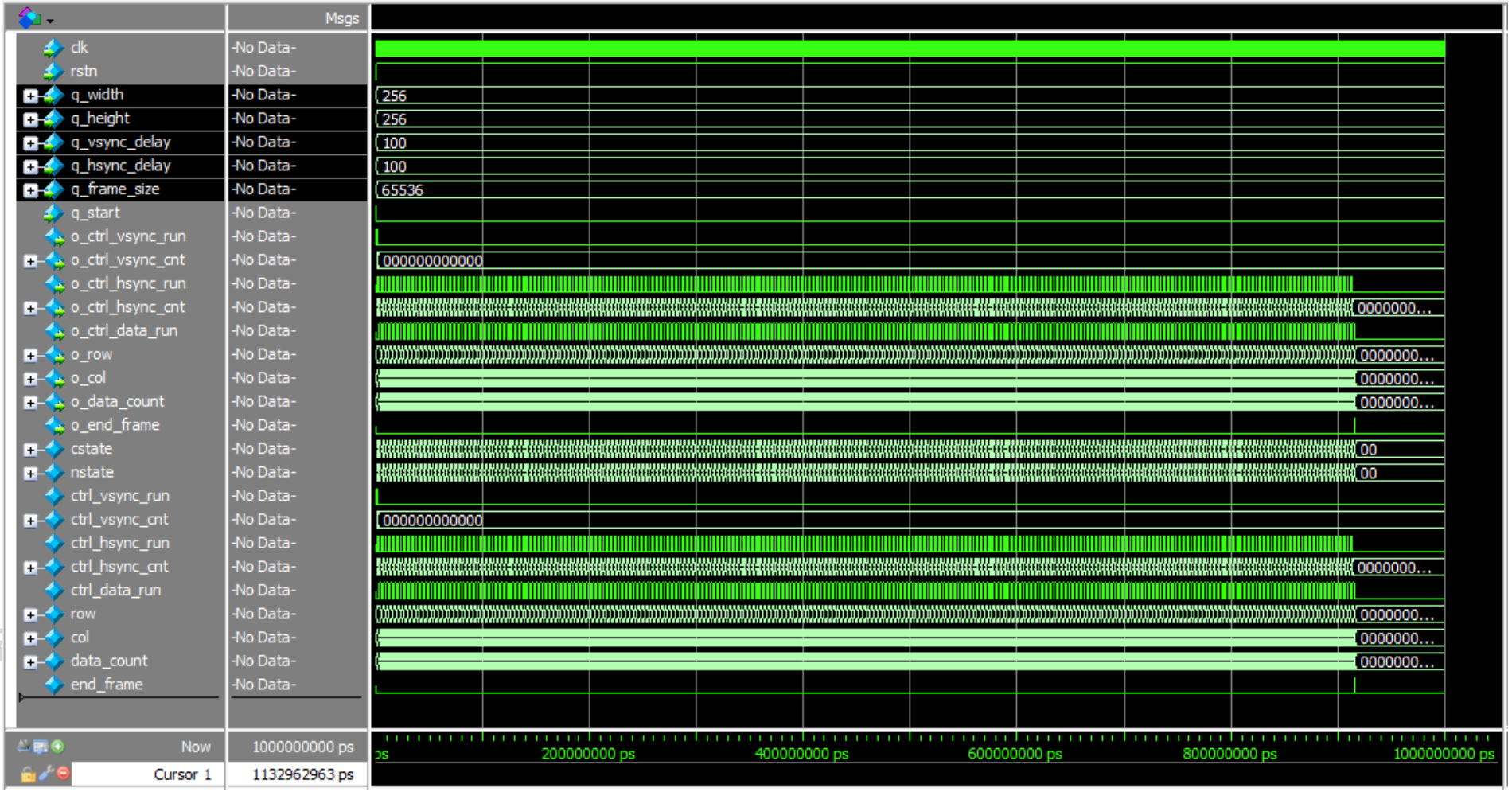
Test cases

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

```
56  // Clock
57  parameter CLK_PERIOD = 10;  //100MHz
58  initial begin
59      clk = 1'b1;
60      forever #(CLK_PERIOD/2) clk = ~clk;
61  end
62
63  //---------------------------------------------------------
64  // Test cases
65  //---------------------------------------------------------
66  initial begin
67      rstn = 1'b0;            // Reset, low active
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
85
```

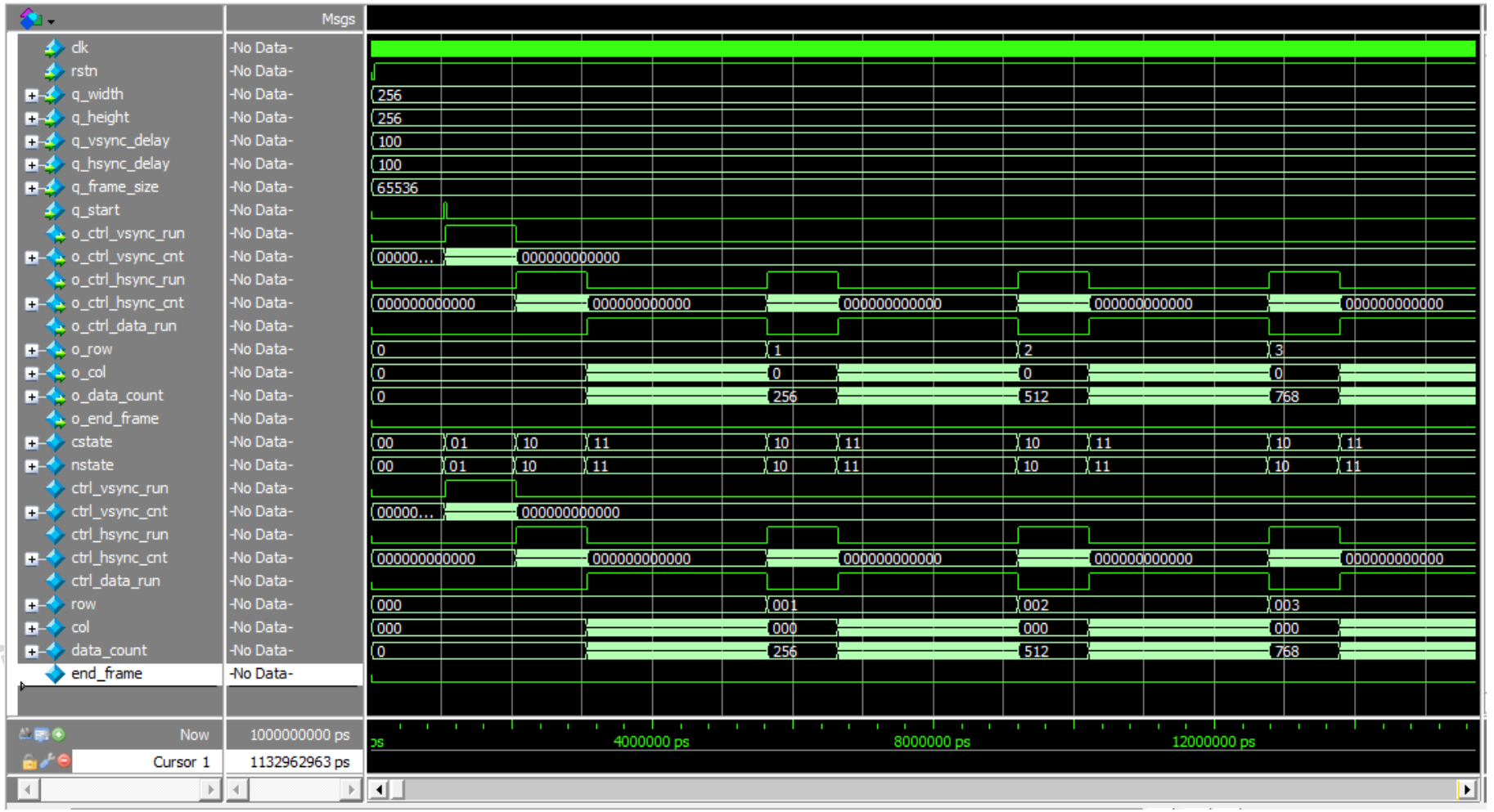
Waveform (cnn_ctrl_tb.v)

- Simulation with time = 1ms

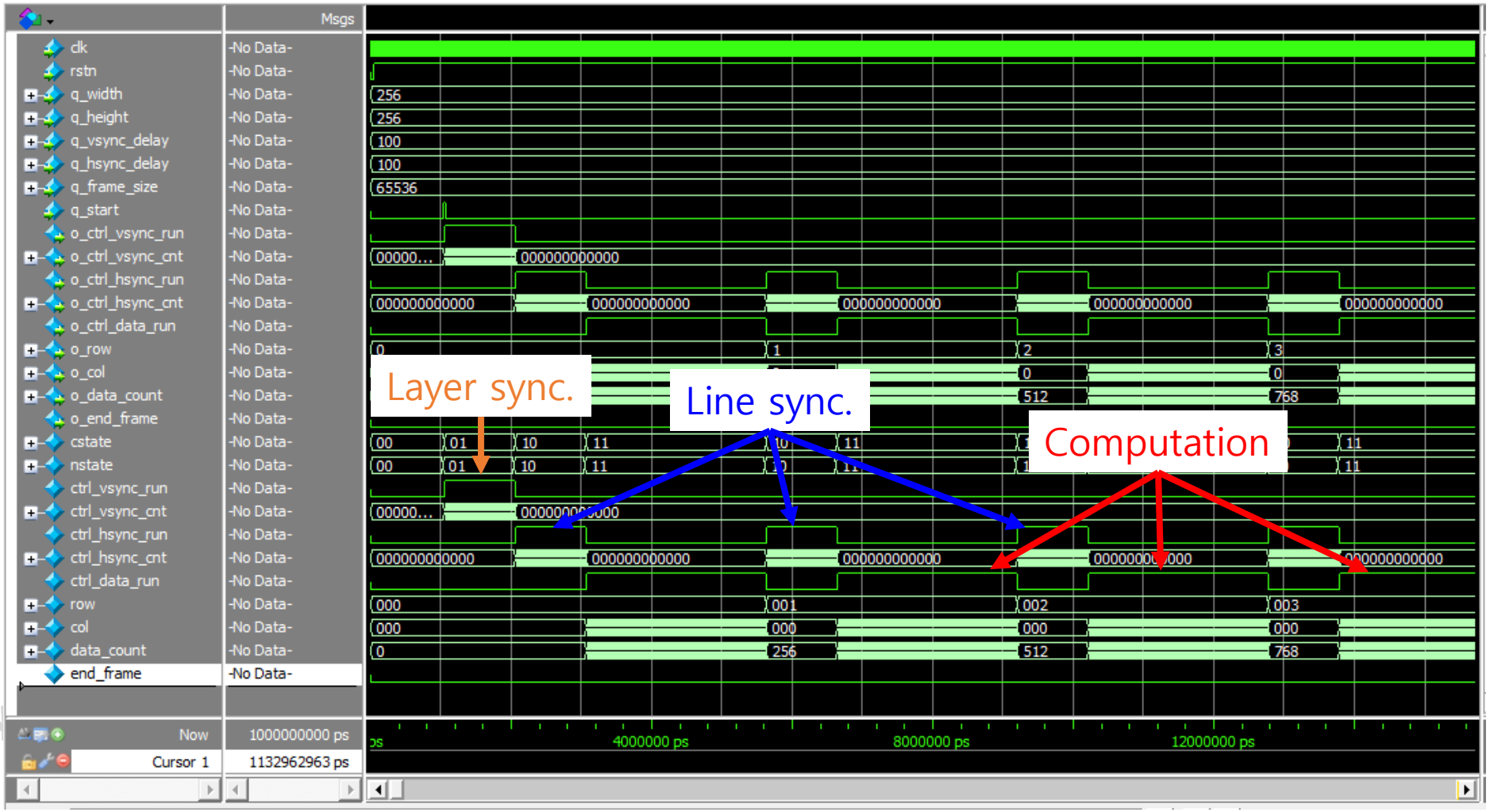


Waveform (cnn_ctrl_tb.v)

- Simulation with time = 1ms

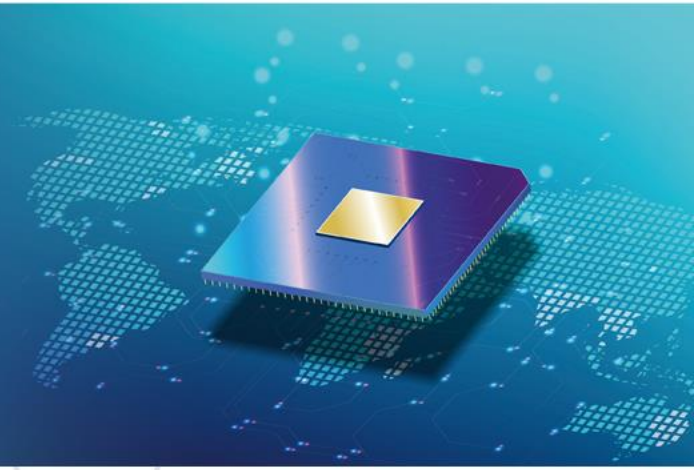


Waveform (cnn_ctrl_tb.v)



Backup Slides

Bmp writer (bmp_image_writer.v)



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