차세대 반도체 디지털 혁신공유대학 사업단
POLARIS

System Integration II
DMA, Top Module

2024.04.06 (Sat)

Objective

- In this tutorial, we show you
  - Make a convolution layer for Layer 00
  - Extend the controller to run multiple layers
  - Introduce the top module

Code structure

- Top level
  - /src Source code
  - /sim Testbench and data for simulation
    - sim/**conv_layer_tb.v**: Load data, do convolution, save inputs and outputs
    - sim/cnn_ctrl_multi_layer_tb.v: A simple controller to generate a loop (row, col indices)
    - sim/yolo_engine_tb.v: A test bench for top module
  - /arxiv Screen-captured results

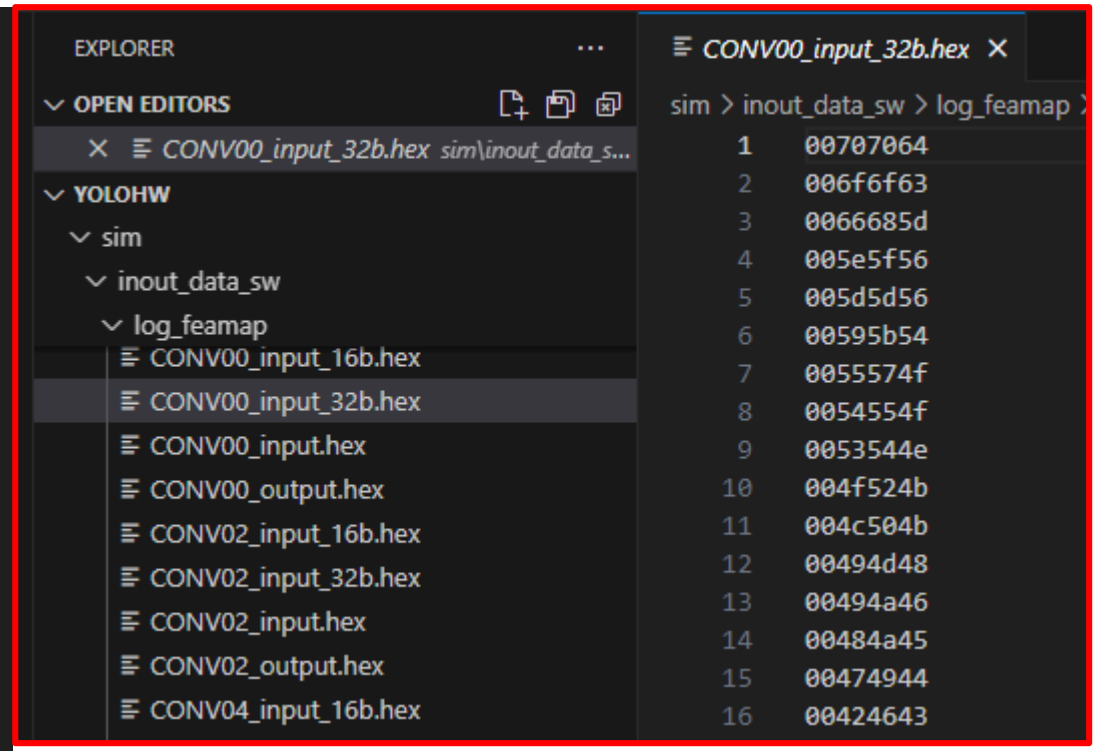
Simulation

/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**
/sim/sim_dram_model	**Code for a simple external memory model**

conv_layer_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
```

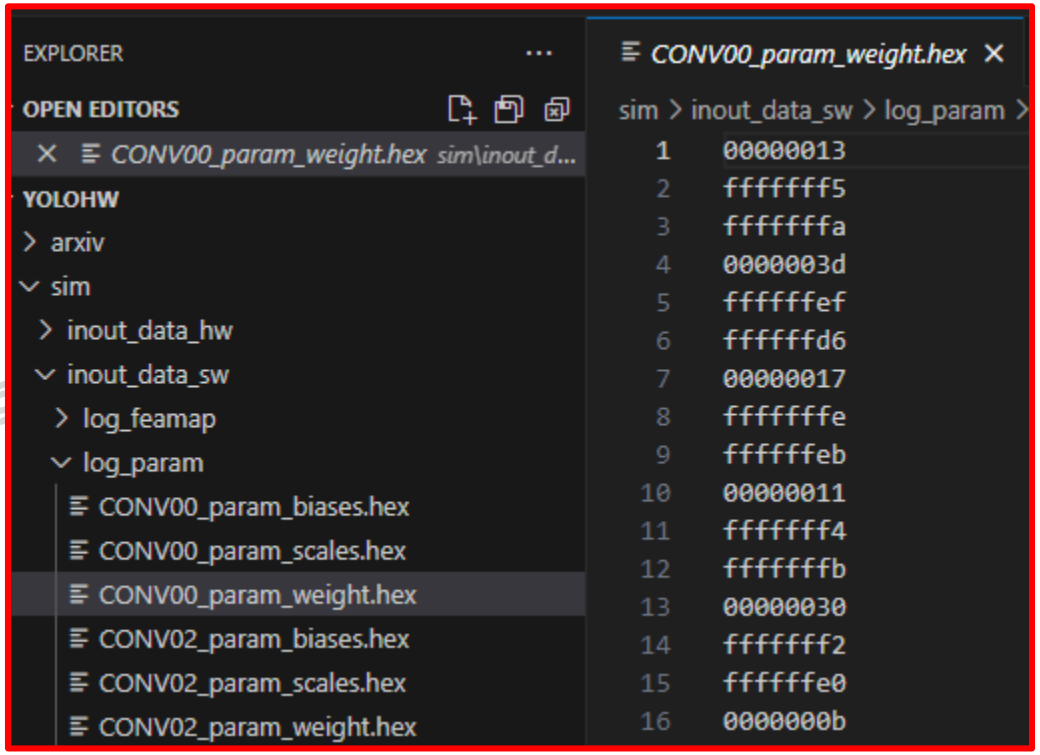


- Each line has 32 bits
- Color format: {0, ch2, ch1, ch0} for CONV00

conv_layer_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_layer_tb.v: Print out the filter

```
//-----------------------------------------------------------------------
// Test vector
//-----------------------------------------------------------------------
integer j;
integer row, col;
initial begin
    // Initialization
   rstn = 1'b0;          // Reset, low active
   preload = 1'b0;
   ctrl_data_run  = 1'b0;
    // Reset and check preloaded filters
   #(4*CLK_PERIOD) rstn = 1'b1;
   #(100*CLK_PERIOD)
        @(posedge clk)
            preload = 1'b1;
    // Show the filter
   #(100*CLK_PERIOD)
        @(posedge clk)
        for (j=0; j < No; j=j+1) begin
            $display("Filter och=%02d: \n",j);
            for(i = 0; i < 3; i = i + 1) begin
                $display("%d\t%d\t%d",
                    $signed(filter[(j*Fx*Fy*Ni) + (3*i  )][7:0]),
                    $signed(filter[(j*Fx*Fy*Ni) + (3*i+1)][7:0]),
                    $signed(filter[(j*Fx*Fy*Ni) + (3*i+2)][7:0]));
            end
            $display("\n");
        end
```

```
VSIM 186> run 3ms
# 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
#
```

conv_layer_tb.v: Use MACs

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

  $\Rightarrow$ 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 ($\sum_{i=0}^{15} w_i * d_i$)

```
//-----------------------------------------
// DUT: MACs
//-----------------------------------------
mac u_mac_00(
./*input         */clk  (clk     ),
./*input         */rstn (rstn    ),
./*input         */vld_i(vld_i   ),
./*input [127:0] */win  (win[0]  ),
./*input [127:0] */din  (din     ),
./*output[ 19:0] */acc_o(acc_o[0]),
./*output        */vld_o(vld_o[0])
);
mac u_mac_01(
./*input         */clk  (clk     ),
./*input         */rstn (rstn    ),
./*input         */vld_i(vld_i   ),
./*input [127:0] */win  (win[1]  ),
./*input [127:0] */din  (din     ),
./*output[ 19:0] */acc_o(acc_o[1]),
./*output        */vld_o(vld_o[1])
);
mac u_mac_02(
./*input         */clk  (clk     ),
./*input         */rstn (rstn    ),
./*input         */vld_i(vld_i   ),
./*input [127:0] */win  (win[2]  ),
./*input [127:0] */din  (din     ),
./*output[ 19:0] */acc_o(acc_o[2]),
./*output        */vld_o(vld_o[2])
);
mac u_mac_03(
./*input         */clk  (clk     ),
./*input         */rstn (rstn    ),
./*input         */vld_i(vld_i   ),
./*input [127:0] */win  (win[3]  ),
./*input [127:0] */din  (din     ),
./*output[ 19:0] */acc_o(acc_o[3]),
./*output        */vld_o(vld_o[3])
);
```

conv_layer_tb.v: Generate row, col, chn ctrl_data_run

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

```
// ****************************************
// Loop for convolutions
// ****************************************
//{{{
#(100*CLK_PERIOD)
    for(row = 0; row < IFM_HEIGHT; row = row + 1)   begin
        @(posedge clk)
            ctrl_data_run  = 0;
        #(100*CLK_PERIOD) @(posedge clk);
            ctrl_data_run  = 1;
        for (col = 0; col < IFM_WIDTH; col = col + 1) begin
            for (chn = 0; chn < IFM_CHANNEL; chn = chn + 1) begin
                @(posedge clk) begin
                    if((col == IFM_WIDTH-1) && (chn == IFM_CHANNEL-1)
                        ctrl_data_run = 0;
                end
            end
        end
    end
@(posedge clk)
        ctrl_data_run = 1'b0;
//}}}
```

conv_layer_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
- Use **chn**

Generate **four 3x3 filters**

NOTE: We only use four filters 0~3 now. You can change j to use other filters.

```
// Generate din, win
wire is_first_row = (row == 0) ? 1'b1: 1'b0;
wire is_last_row  = (row == IFM_HEIGHT-1) ? 1'b1: 1'b0;
wire is_first_col = (col == 0) ? 1'b1: 1'b0;
wire is_last_col  = (col == IFM_WIDTH-1) ? 1'b1 : 1'b0;

always@(*) begin
    vld_i = 0;
    din = 128'd0;
    win[0] = 0;
    win[1] = 0;
    win[2] = 0;
    win[3] = 0;
    if(ctrl_data_run) begin
        vld_i = 1;
        // Tiled IFM data
        din[ 7: 0] = (is_first_row || is_first_col) ? 8'd0 : in_img[(row-1) * IFM_WIDTH + (col-1)][chn*8+:8];
        din[15: 8] = (is_first_row                ) ? 8'd0 : in_img[(row-1) * IFM_WIDTH +  col   ][chn*8+:8];
        din[23:16] = (is_first_row || is_last_col ) ? 8'd0 : in_img[(row-1) * IFM_WIDTH + (col+1)][chn*8+:8];

        din[31:24] = (               is_first_col) ? 8'd0 : in_img[ row    * IFM_WIDTH + (col-1)][chn*8+:8];
        din[39:32] =                                        in_img[ row    * IFM_WIDTH +  col   ][chn*8+:8];
        din[47:40] = (               is_last_col ) ? 8'd0 : in_img[ row    * IFM_WIDTH + (col+1)][chn*8+:8];

        din[55:48] = (is_last_row ||  is_first_col) ? 8'd0 : in_img[(row+1) * IFM_WIDTH + (col-1)][chn*8+:8];
        din[63:56] = (is_last_row                ) ? 8'd0 : in_img[(row+1) * IFM_WIDTH +  col   ][chn*8+:8];
        din[71:64] = (is_last_row ||  is_last_col ) ? 8'd0 : in_img[(row+1) * IFM_WIDTH + (col+1)][chn*8+:8];
        // Tiled Filters
        for(j = 0; j < 4; j=j+1) begin  // Four sets <=> Four output channels
            win[j][ 7: 0] = filter[(j*Fx*Fy*Ni) + chn*9    ][7:0];
            win[j][15: 8] = filter[(j*Fx*Fy*Ni) + chn*9 + 1][7:0];
            win[j][23:16] = filter[(j*Fx*Fy*Ni) + chn*9 + 2][7:0];
            win[j][31:24] = filter[(j*Fx*Fy*Ni) + chn*9 + 3][7:0];
            win[j][39:32] = filter[(j*Fx*Fy*Ni) + chn*9 + 4][7:0];
            win[j][47:40] = filter[(j*Fx*Fy*Ni) + chn*9 + 5][7:0];
            win[j][55:48] = filter[(j*Fx*Fy*Ni) + chn*9 + 6][7:0];
            win[j][63:56] = filter[(j*Fx*Fy*Ni) + chn*9 + 7][7:0];
            win[j][71:64] = filter[(j*Fx*Fy*Ni) + chn*9 + 8][7:0];
        end
    end
end
```

conv_layer_tb.v:

- **Accumulator:** Accumulate the results, for example
  - In this case, assume that each cycle convolves a filter 3x3 with a fmap window 3x3
  - In Layer 00, we must convolve a filter 3x3x3 with a fmap window 3x3x3.
  - Therefore, we may compute three times across the channel direction.
  - The output results from MACs are accumulated at the partial sum (psum)

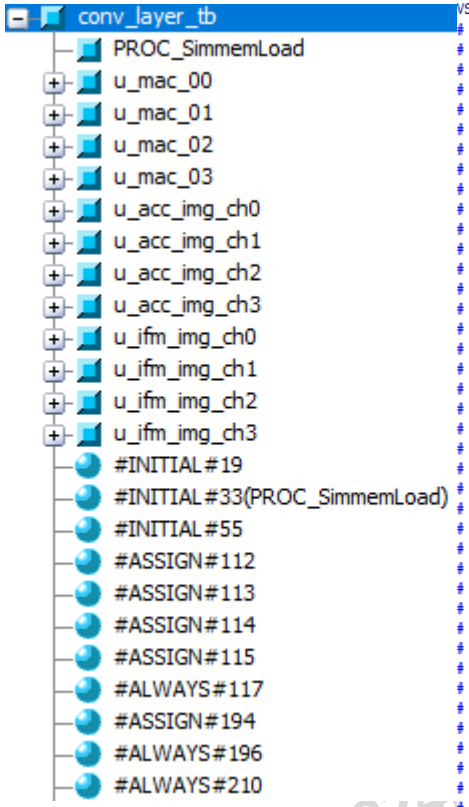
```
reg [31:0] psum[0:3];
wire valid_out = vld_o[0];

always@(posedge clk, negedge rstn) begin
    if(!rstn) begin
        chn_idx <= 0;
    end
    else begin
        if(valid_out) begin
            if(chn_idx == IFM_CHANNEL-1)
                chn_idx <= 0;
            else
                chn_idx <= chn_idx + 1;
        end
    end
end
reg write_pixel_ena;
always@(posedge clk, negedge rstn) begin
    if(!rstn) begin
        psum[0] <= 0;
        psum[1] <= 0;
        psum[2] <= 0;
        psum[3] <= 0;
        write_pixel_ena <= 0;
    end
    else begin
        if(valid_out) begin
            if(chn_idx == 0) begin
                psum[0] <= $signed(acc_o[0]);
                psum[1] <= $signed(acc_o[1]);
                psum[2] <= $signed(acc_o[2]);
                psum[3] <= $signed(acc_o[3]);
            end
            else begin
                psum[0] <= $signed(psum[0]) + $signed(acc_o[0]);
                psum[1] <= $signed(psum[1]) + $signed(acc_o[1]);
                psum[2] <= $signed(psum[2]) + $signed(acc_o[2]);
                psum[3] <= $signed(psum[3]) + $signed(acc_o[3]);
            end

            if(chn_idx == IFM_CHANNEL-1)
                write_pixel_ena <= 1;
            else
                write_pixel_ena <= 0;
        end
        else
            write_pixel_ena <= 0;
    end
end
```

conv_layer_tb.v: Descaling/Dequantization

- After doing convolution, it does activation and descaling or dequantization
  - Activation: Use RELU
  - Descaling: Generate eight-bit output pixels used as the inputs for the next layer

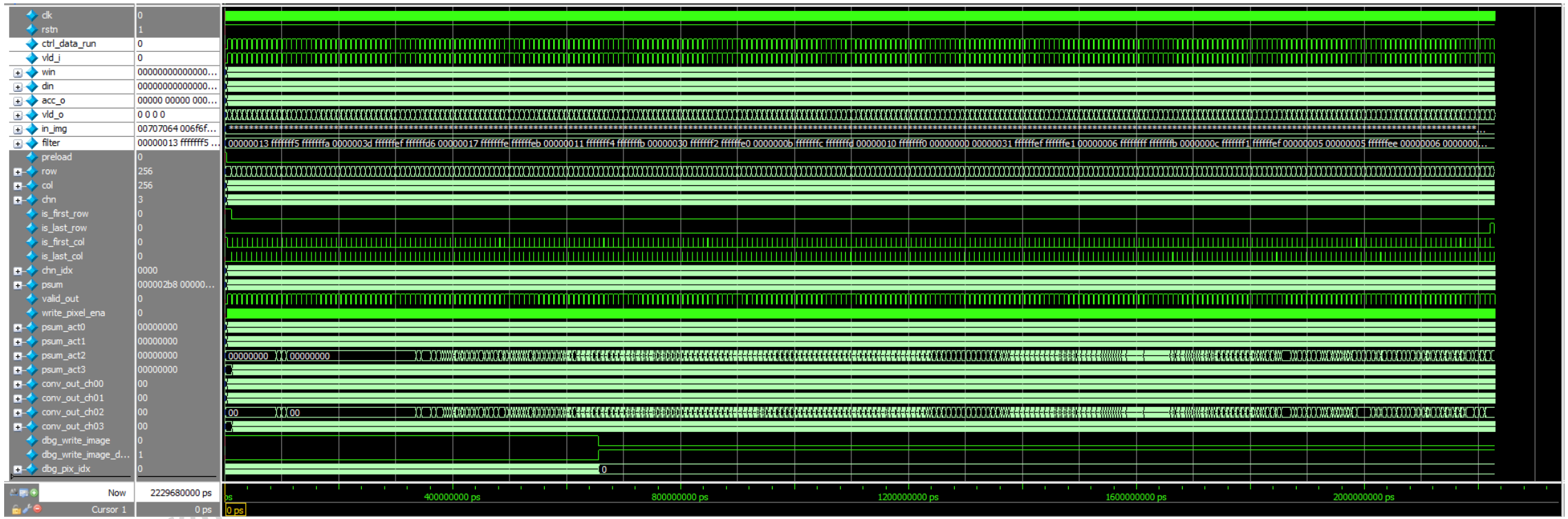
```
            if(chn_idx == IFM_CHANNEL-1)
                write_pixel_ena <= 1;
            else
                write_pixel_ena <= 0;
        end
        else
            write_pixel_ena <= 0;
    end
end
//-----------------------------------------------------------------
// For debugging: INPUTs/Outputs
//-----------------------------------------------------------------
// Activation + Quantization (Descaling)
//{{{
wire [31:0]  psum_act0 = write_pixel_ena ? ((psum[0][31]==1)?0:psum[0]): 0;
wire [31:0]  psum_act1 = write_pixel_ena ? ((psum[1][31]==1)?0:psum[1]): 0;
wire [31:0]  psum_act2 = write_pixel_ena ? ((psum[2][31]==1)?0:psum[2]): 0;
wire [31:0]  psum_act3 = write_pixel_ena ? ((psum[3][31]==1)?0:psum[3]): 0;

wire [7:0] conv_out_ch00 = write_pixel_ena?((psum_act0[31:7]>255)?255:psum_act0[14:7]):0; // Descaling: * 1/2^11
wire [7:0] conv_out_ch01 = write_pixel_ena?((psum_act1[31:7]>255)?255:psum_act1[14:7]):0; // Descaling: * 1/2^11
wire [7:0] conv_out_ch02 = write_pixel_ena?((psum_act2[31:7]>255)?255:psum_act2[14:7]):0; // Descaling: * 1/2^11
wire [7:0] conv_out_ch03 = write_pixel_ena?((psum_act3[31:7]>255)?255:psum_act3[14:7]):0; // Descaling: * 1/2^11
```

conv_layer_tb.v: Simulation time = 3ms




```
VSIM 186> run 3ms
# 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=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
# Layer done !!!
# ** Note: $stop    : C:/yolohw/sim/conv_layer_tb.v(108)
#    Time: 2229680 ns  Iteration: 0  Instance: /conv_layer_tb
# Break in Module conv_layer_tb at C:/yolohw/sim/conv_layer_tb.v line 108
```

conv_layer_tb.v: Simulation time = 3ms



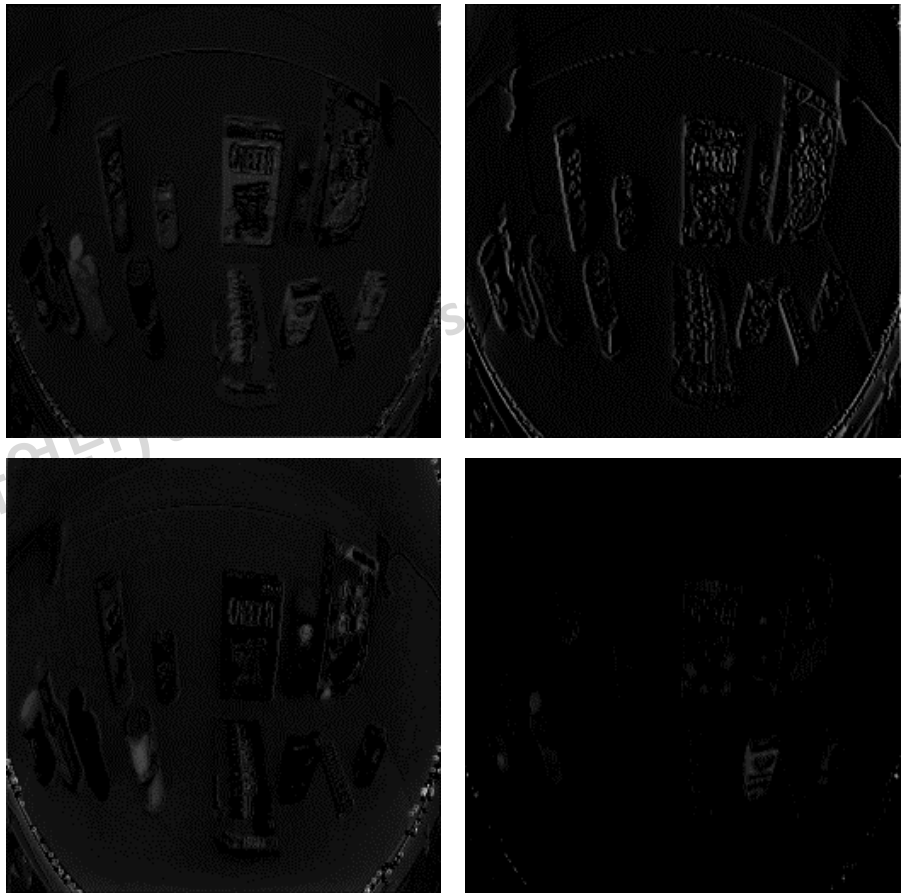
conv_layer_tb.v: Checking inputs

- We can save the input in an BMP image

```
bmp_image_writer #(.OUTFILE(CONV_INPUT_IMG00),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
u_ifm_img_ch0(
    ./*input            */clk       (clk                      ),
    ./*input            */rstn      (rstn                     ),
    ./*input [WI-1:0]   */din       (in_img[dbg_pix_idx][7:0]  ),
    ./*input            */vld       (dbg_write_image          ),
    ./*output reg       */frame_done(                         )
);
bmp_image_writer #(.OUTFILE(CONV_INPUT_IMG01),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
u_ifm_img_ch1(
    ./*input            */clk       (clk                      ),
    ./*input            */rstn      (rstn                     ),
    ./*input [WI-1:0]   */din       (in_img[dbg_pix_idx][15:8] ),
    ./*input            */vld       (dbg_write_image          ),
    ./*output reg       */frame_done(                         )
);
bmp_image_writer #(.OUTFILE(CONV_INPUT_IMG02),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
u_ifm_img_ch2(
    ./*input            */clk       (clk                      ),
    ./*input            */rstn      (rstn                     ),
    ./*input [WI-1:0]   */din       (in_img[dbg_pix_idx][23:16]),
    ./*input            */vld       (dbg_write_image          ),
    ./*output reg       */frame_done(                         )
);
bmp_image_writer #(.OUTFILE(CONV_INPUT_IMG03),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
u_ifm_img_ch3(
    ./*input            */clk       (clk                      ),
    ./*input            */rstn      (rstn                     ),
    ./*input [WI-1:0]   */din       (in_img[dbg_pix_idx][31:24]),
    ./*input            */vld       (dbg_write_image          ),
    ./*output reg       */frame_done(                         )
);
```

conv_layer_tb.v: Visual Verification

```
258   bmp_image_writer #(.OUTFILE(CONV_OUTPUT_IMG00),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
259   u_acc_img_ch0(
260       ./*input            */clk       (clk            ),
261       ./*input            */rstn      (rstn           ),
262       ./*input [WI-1:0]   */din       (conv_out_ch00  ),
263       ./*input            */vld       (write_pixel_ena),
264       ./*output reg       */frame_done(               )
265   );
266   bmp_image_writer #(.OUTFILE(CONV_OUTPUT_IMG01),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
267   u_acc_img_ch1(
268       ./*input            */clk       (clk            ),
269       ./*input            */rstn      (rstn           ),
270       ./*input [WI-1:0]   */din       (conv_out_ch01  ),
271       ./*input            */vld       (write_pixel_ena),
272       ./*output reg       */frame_done(               )
273   );
274   bmp_image_writer #(.OUTFILE(CONV_OUTPUT_IMG02),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
275   u_acc_img_ch2(
276       ./*input            */clk       (clk            ),
277       ./*input            */rstn      (rstn           ),
278       ./*input [WI-1:0]   */din       (conv_out_ch02  ),
279       ./*input            */vld       (write_pixel_ena),
280       ./*output reg       */frame_done(               )
281   );
282   bmp_image_writer #(.OUTFILE(CONV_OUTPUT_IMG03),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
283   u_acc_img_ch3(
284       ./*input            */clk       (clk            ),
285       ./*input            */rstn      (rstn           ),
286       ./*input [WI-1:0]   */din       (conv_out_ch03  ),
287       ./*input            */vld       (write_pixel_ena),
288       ./*output reg       */frame_done(               )
289   );
```



IMPORTANT NOTE: It is only for **visual verification**
You must **write data in hex files** and **compare them with the outputs from the software**

Objective

- In this tutorial, we show you
  - Make a convolution layer for Layer 00
  - Extend the controller to run multiple layers
  - Introduce the top module

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

```
timescale 1ns / 1ps

module cnn_ctrl(
clk,
rstn,
// Inputs
q_width,
q_height,
q_vsync_delay,
q_hsync_delay,
q_frame_size,
q_start,
//output
o_ctrl_vsync_run,
o_ctrl_vsync_cnt,
o_ctrl_hsync_run,
o_ctrl_hsync_cnt,
o_ctrl_data_run,
o_row,
o_col,
o_data_count,
o_end_frame
);
```

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

```
//-----------------------------------------------
// Internal signals
//-----------------------------------------------
localparam      ST_IDLE     = 2'b00,
                ST_VSYNC    = 2'b01,
                ST_HSYNC    = 2'b10,
                ST_DATA     = 2'b11;
reg [1:0] cstate, nstate;
reg                     ctrl_vsync_run;
reg [W_DELAY-1:0]       ctrl_vsync_cnt;
reg                     ctrl_hsync_run;
reg [W_DELAY-1:0]       ctrl_hsync_cnt;
reg                     ctrl_data_run;
reg [W_SIZE-1:0]        row;
reg [W_SIZE-1:0]        col;
reg [W_FRAME_SIZE-1:0] data_count;
wire end_frame;
```

Test bench (cnn_ctrl_multi_layer_tb.v)

Layer configurations
q_width, q_height, q_frame_size, …

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

```
//-----------------------------------------------
// Controller (FSM)
//-----------------------------------------------
cnn_ctrl u_cnn_ctrl (
.clk            (clk            ),
.rstn           (rstn           ),
// Inputs
.q_width        (q_width        ),
.q_height       (q_height       ),
.q_vsync_delay  (q_vsync_delay  ),
.q_hsync_delay  (q_hsync_delay  ),
.q_frame_size   (q_frame_size   ),
.q_start        (q_start        ),
//output
.o_ctrl_vsync_run(ctrl_vsync_run),
.o_ctrl_vsync_cnt(ctrl_vsync_cnt),
.o_ctrl_hsync_run(ctrl_hsync_run),
.o_ctrl_hsync_cnt(ctrl_hsync_cnt),
.o_ctrl_data_run(ctrl_data_run  ),
.o_row          (row            ),
.o_col          (col            ),
.o_data_count   (data_count     ),
.o_end_frame    (end_frame      )
);

// Clock
parameter CLK_PERIOD = 10;  //100MHz
initial begin
    clk = 1'b1;
    forever #(CLK_PERIOD/2) clk = ~clk;
end
```

Test cases: Layer 00

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

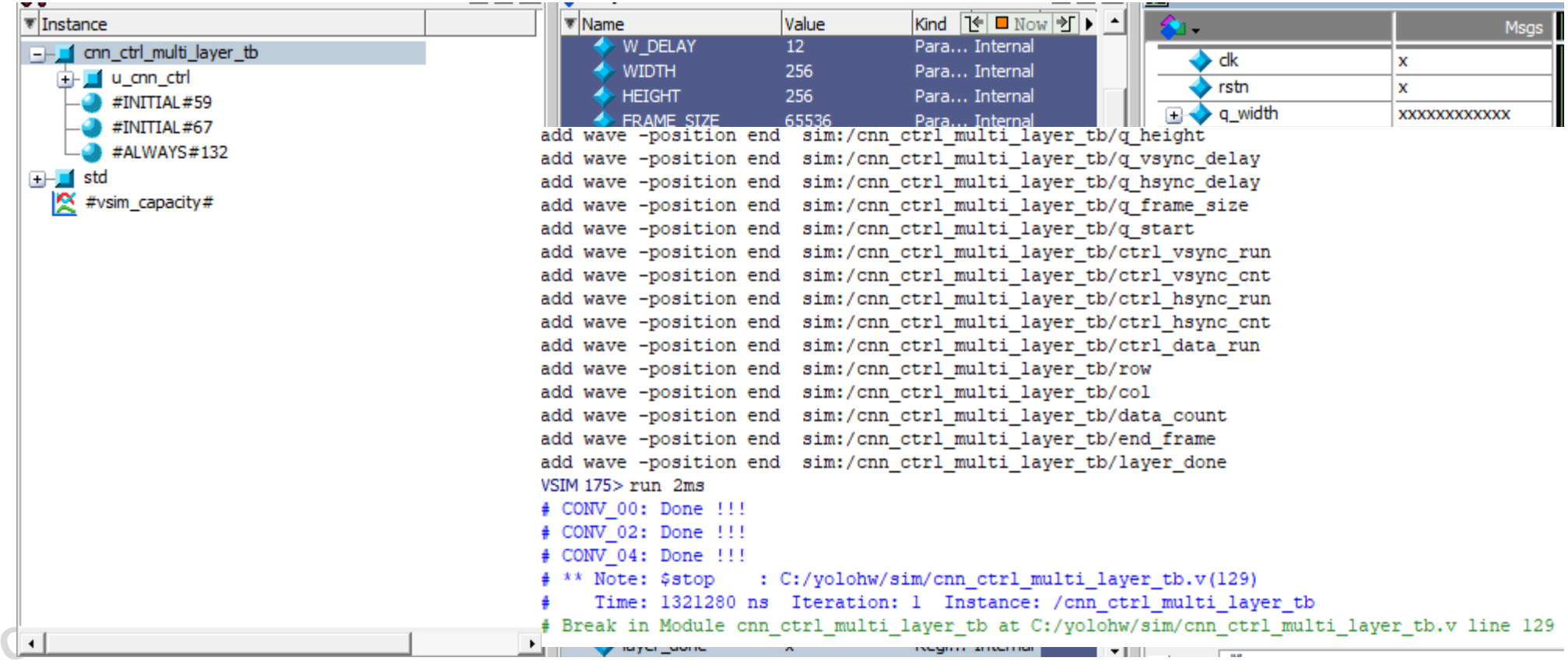
```
//-----------------------------------------------------------------
// Test cases
//-----------------------------------------------------------------
initial begin
    rstn = 1'b0;            // Reset, low active
    q_width         = WIDTH;
    q_height        = HEIGHT;
    q_vsync_delay   = VSYNC_DELAY;
    q_hsync_delay   = HSYNC_DELAY;
    q_frame_size    = FRAME_SIZE;
    q_start         = 1'b0;

    #(4*CLK_PERIOD) rstn = 1'b1;

    //-----------------------------------------------------------
    // Layer 1: width = height = 256
    //-----------------------------------------------------------
    q_width  = 256;
    q_height = 256;
    q_frame_size = 256*256;
    #(100*CLK_PERIOD)
        @(posedge clk)
            q_start = 1'b1;
    #(4*CLK_PERIOD)
        @(posedge clk)
            q_start = 1'b0;
    while(!layer_done) begin
        #(128*CLK_PERIOD) @(posedge clk);
    end
    $display("CONV_00: Done !!!");

```

Test cases: Layers 02, 04

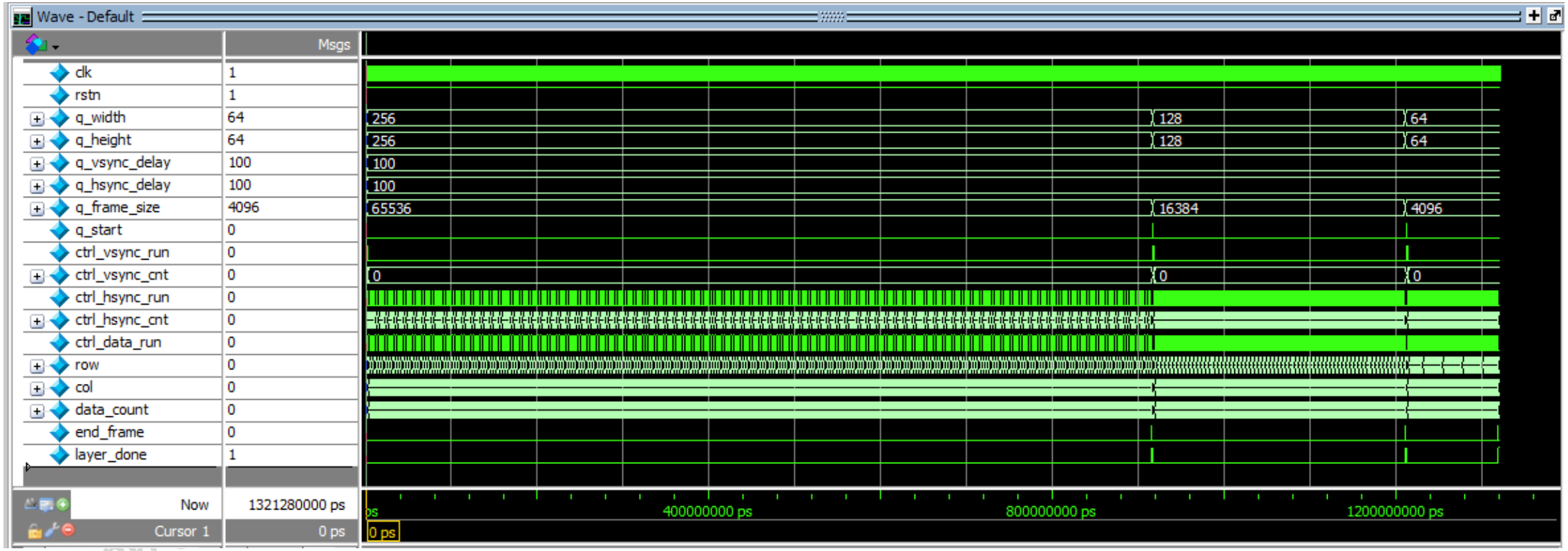
- Set parameters
  - q_width, q_height, q_frame_size
  - q_vsync_delay, q_hsync_delay
- Trigger with "q_start"

```
//-----------------------------------------------------------------
// Layer 2: width = height = 128
//-----------------------------------------------------------------
q_width  = 128;
q_height = 128;
q_frame_size = 128*128;
#(100*CLK_PERIOD)
    @(posedge clk)
        q_start = 1'b1;
#(4*CLK_PERIOD)
    @(posedge clk)
        q_start = 1'b0;
while(!layer_done) begin
    #(128*CLK_PERIOD) @(posedge clk);
end
$display("CONV_02: Done !!!");

//-----------------------------------------------------------------
// Layer 3: width = height = 64
//-----------------------------------------------------------------
q_width  = 64;
q_height = 64;
q_frame_size = 64*64;
#(100*CLK_PERIOD)
    @(posedge clk)
        q_start = 1'b1;
#(4*CLK_PERIOD)
    @(posedge clk)
        q_start = 1'b0;
while(!layer_done) begin
    #(128*CLK_PERIOD) @(posedge clk);
end
$display("CONV_04: Done !!!");
#(100*CLK_PERIOD)
    @(posedge clk) $stop;
```

Simulation Results: Run 2ms

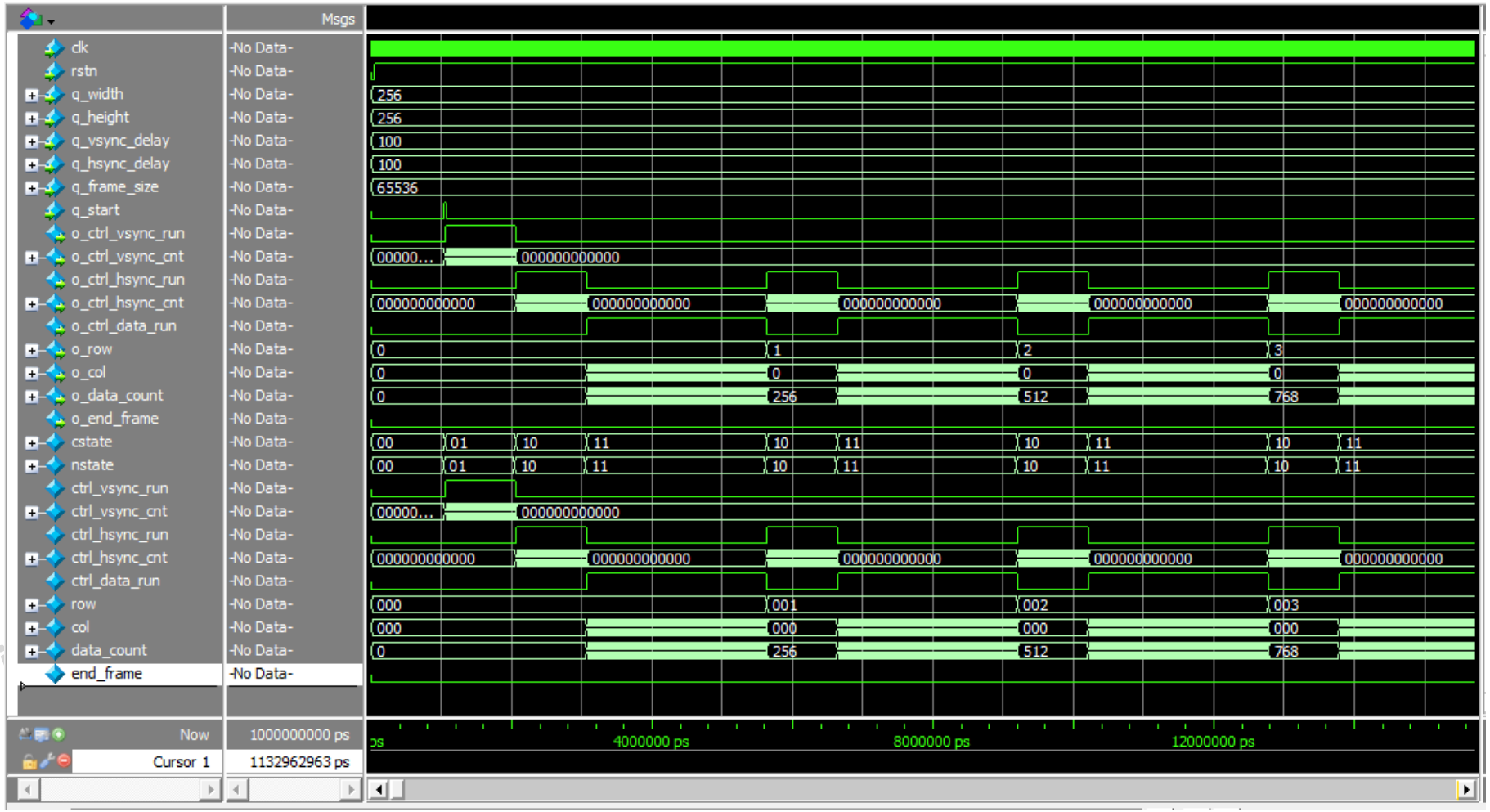


Simulation Results: Layers 0, 2, 4

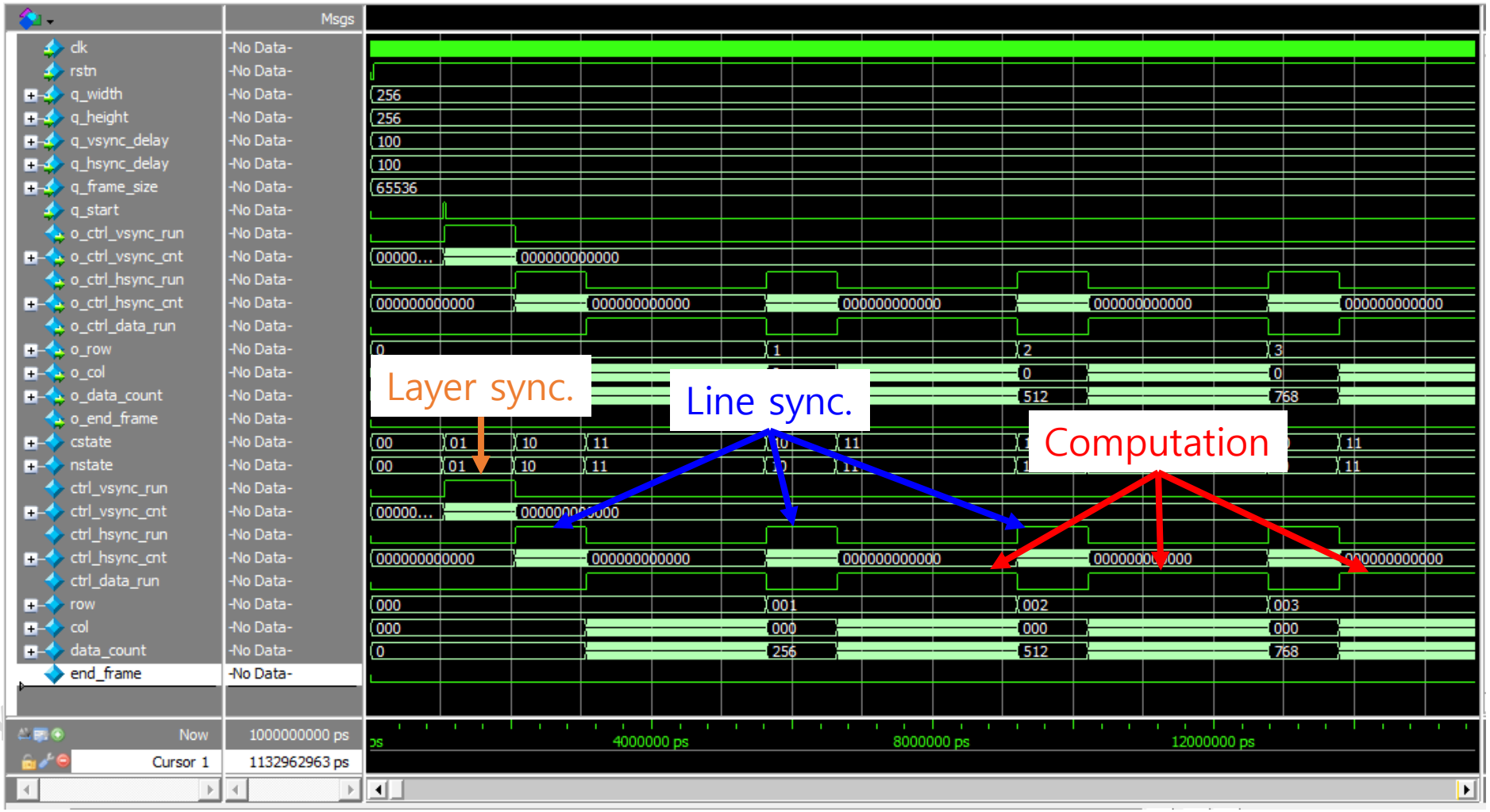


Waveform (cnn_ctrl_tb.v)

- Simulation with time = 1ms



Waveform (cnn_ctrl_tb.v)



Objective

- In this tutorial, we show you
  - Make a convolution layer for Layer 00
  - Extend the controller to run multiple layers
  - Introduce the top module

Top module: yolo_engine.v

- The top module
  - Control signals
    - i_ctrl_reg0~3
  - axi interfaces
  - Layer done signals
    - network_done
    - network_done_led

```
module yolo_engine #(
    parameter AXI_WIDTH_AD = 32,
    parameter AXI_WIDTH_ID = 4,
    parameter AXI_WIDTH_DA = 32,
    parameter AXI_WIDTH_DS = AXI_WIDTH_DA/8,
    parameter OUT_BITS_TRANS = 18,
    parameter WBUF_AW = 9,
    parameter WBUF_DW = 8*3*3*16,
    parameter WBUF_DS = WBUF_DW/8,
    parameter MEM_BASE_ADDR = 'h8000_0000,
    parameter MEM_DATA_BASE_ADDR = 4096
)
(
    input                          clk
    , input                          rstn

    , input [31:0] i_ctrl_reg0    // network_start, // {debug_big(1)
    , input [31:0] i_ctrl_reg1    // Read address base
    , input [31:0] i_ctrl_reg2    // Write address base
    , input [31:0] i_ctrl_reg3    // Write address base

    , output                         M_ARVALID
    , input                          M_ARREADY
    , output  [AXI_WIDTH_AD-1:0]     M_ARADDR
    , output  [AXI_WIDTH_ID-1:0]     M_ARID
    , output  [7:0]                  M_ARLEN
    , output  [2:0]                  M_ARSIZE
    , output  [1:0]                  M_ARBURST
    , output  [1:0]                  M_ARLOCK
    , output  [3:0]                  M_ARCACHE
    , output  [2:0]                  M_ARPROT
    , output  [3:0]                  M_ARQOS
    , output  [3:0]                  M_ARREGION
    , output  [3:0]                  M_ARUSER
    , input                          M_RVALID
    , output                         M_RREADY
    , input  [AXI_WIDTH_DA-1:0]      M_RDATA
    , input                          M_RLAST
    , input  [AXI_WIDTH_ID-1:0]      M_RID
    , input  [3:0]                   M_RUSER
    , input  [1:0]                   M_RRESP
```

```

    , output                         M_AWVALID
    , input                          M_AWREADY
    , output  [AXI_WIDTH_AD-1:0]     M_AWADDR
    , output  [AXI_WIDTH_ID-1:0]     M_AWID
    , output  [7:0]                  M_AWLEN
    , output  [2:0]                  M_AWSIZE
    , output  [1:0]                  M_AWBURST
    , output  [1:0]                  M_AWLOCK
    , output  [3:0]                  M_AWCACHE
    , output  [2:0]                  M_AWPROT
    , output  [3:0]                  M_AWQOS
    , output  [3:0]                  M_AWREGION
    , output  [3:0]                  M_AWUSER

    , output                         M_WVALID
    , input                          M_WREADY
    , output  [AXI_WIDTH_DA-1:0]     M_WDATA
    , output  [AXI_WIDTH_DS-1:0]     M_WSTRB
    , output                         M_WLAST
    , output  [AXI_WIDTH_ID-1:0]     M_WID
    , output  [3:0]                  M_WUSER

    , input                          M_BVALID
    , output                         M_BREADY
    , input  [1:0]                   M_BRESP
    , input  [AXI_WIDTH_ID-1:0]      M_BID
    , input                          M_BUSER

    , output network_done
    , output network_done_led
);
`include "define.v"
```

Top module (yolo_engine.v): Control signals

```
//-----------------------------------------------------------------
// Control signals
//-----------------------------------------------------------------
always @ (*) begin
    ap_done     = ctrl_write_done;
    ap_ready    = 1;
end
assign network_done     = interrupt;
assign network_done_led = interrupt;


always @ (posedge clk, negedge rstn) begin
    if(~rstn) begin
        ap_start <= 0;
    end
    else begin
        if(!ap_start && i_ctrl_reg0[0])
            ap_start <= 1;
        else if (ap_done)
            ap_start <= 0;
    end
end

always @(posedge clk, negedge rstn) begin
    if(~rstn) begin
        interrupt <= 0;
    end
    else begin
        if(i_ctrl_reg0[0])
            interrupt <= 0;
        else if (ap_done)
            interrupt <= 1;
    end
end
```

Top module (yolo_engine.v):

- Parse the configurations from the external
  - For example, the **base address** to read/write the external memory.

```
// Parse the control registers
always @ (posedge clk, negedge rstn) begin
    if(~rstn) begin
        dram_base_addr_rd <= 0;
        dram_base_addr_wr <= 0;
        reserved_register <= 0; // unused
    end
    else begin
        if(!ap_start && i_ctrl_reg0[0]) begin
            dram_base_addr_rd <= i_ctrl_reg1; // Base Address for READ  (Input image, Model parameters)
            dram_base_addr_wr <= i_ctrl_reg2; // Base Address for WRITE (Intermediate feature maps, Outputs)
            reserved_register <= i_ctrl_reg3; // reserved
        end
        else if (ap_done) begin
            dram_base_addr_rd <= 0;
            dram_base_addr_wr <= 0;
            reserved_register <= 0;
        end
    end
end
```

Top module: DMA controller

- Inputs
  - **DRAM base addresses** for Read/Write (dram_base_addr_rd, dram_base_addr_wr)
  - **Number of transactions** for a request (num_trans). WE FIXED IT TO 16 HERE.
  - **Maximum requests** (max_req_blk_idx). For example, it is (256*256)/16 when we want to read an image
- Outputs:
  - **Send trigger signals** for read/write (ctrl_read/ctrl_write) to u_dma_read (axi_dma_rd) and u_dma_write (axi_dma_wr)

```
//-----------------------------------------------------------------
// DUTs
//-----------------------------------------------------------------
// DMA Controller
axi_dma_ctrl #(.BIT_TRANS(BIT_TRANS))
u_dma_ctrl(
    .clk              (clk              )
   ,.rstn             (rstn             )
   ,.i_start          (i_ctrl_reg0[0]   )
   ,.i_base_address_rd(dram_base_addr_rd)
   ,.i_base_address_wr(dram_base_addr_wr)
   ,.i_num_trans      (num_trans        )
   ,.i_max_req_blk_idx(max_req_blk_idx  )
   // DMA Read
   ,.i_read_done      (read_done        )
   ,.o_ctrl_read      (ctrl_read        )
   ,.o_read_addr      (read_addr        )
   // DMA Write
   ,.i_indata_req_wr  (indata_req_wr    )
   ,.i_write_done     (write_done       )
   ,.o_ctrl_write     (ctrl_write       )
   ,.o_write_addr     (write_addr       )
   ,.o_write_data_cnt (write_data_cnt   )
   ,.o_ctrl_write_done(ctrl_write_done  )
);
```

Top module: DMA read

- The flow includes:

1. Receive a READ request signal from u_dma_ctrl (ctrl_read, num_trans, read_addr)
2. Send the read request to external memory via an axi interface
3. Receive the return data via an axi interface
4. Send the return data to a buffer
   - For example, request 16 x 32-bit (num_trans = 16), (e.g., 16th data)

```
u_dma_read(
    //AXI Master Interface
    //Read address channel
    .M_ARVALID  (M_ARVALID    ),  // address/control valid handshake
    .M_ARREADY  (M_ARREADY    ),  // Read addr ready
    .M_ARADDR   (M_ARADDR     ),  // Address Read
    .M_ARID     (M_ARID       ),  // Read addr ID
    .M_ARLEN    (M_ARLEN      ),  // Transfer length
    .M_ARSIZE   (M_ARSIZE     ),  // Transfer width
    .M_ARBURST  (M_ARBURST    ),  // Burst type
    .M_ARLOCK   (M_ARLOCK     ),  // Atomic access information
    .M_ARCACHE  (M_ARCACHE    ),  // Cachable/bufferable infor
    .M_ARPROT   (M_ARPROT     ),  // Protection info
    .M_ARQOS    (M_ARQOS      ),  // Quality of Service
    .M_ARREGION (M_ARREGION   ),  // Region signaling
    .M_ARUSER   (M_ARUSER     ),  // User defined signal

    //Read data channel
    .M_RVALID   (M_RVALID     ),  // Read data valid
    .M_RREADY   (M_RREADY     ),  // Read data ready (to Slave)
    .M_RDATA    (M_RDATA      ),  // Read data bus
    .M_RLAST    (M_RLAST      ),  // Last beat of a burst transfer
    .M_RID      (M_RID        ),  // Read ID
    .M_RUSER    (M_RUSER      ),  // User defined signal
    .M_RRESP    (M_RRESP      ),  // Read response

    //Functional Ports
    .start_dma  (ctrl_read    ),
    .num_trans  (num_trans    ), //Number of 128-bit words transferred
    .start_addr (read_addr    ), //iteration_num * 4 * 16 + read_address_d
    .data_o     (read_data    ),
    .data_vld_o (read_data_vld),
    .data_cnt_o (read_data_cnt),
    .done_o     (read_done    ),

    //Global signals
    .clk        (clk          ),
    .rstn       (rstn         )
);
```

Top module: Data buffer

- A buffer saves the data coming from u_dma_read

```
// dpram_256x32
dpram_wrapper #(
    .DEPTH  (BUFF_DEPTH     ),
    .AW     (BUFF_ADDR_W    ),
    .DW     (AXI_WIDTH_DA   ))
u_data_buffer(
    .clk    (clk            ),
    .ena    (1'd1           ),
    .addra  (read_data_cnt  ),
    .wea    (read_data_vld  ),
    .dia    (read_data      ),
    .enb    (1'd1           ),  // Always Read
    .addrb  (write_data_cnt ),
    .dob    (write_data     )
);
```

Top module: DMA write

- The flow includes:

1. Receive a WRITE request signal from u_dma_ctrl (ctrl_write, num_trans, write_addr)
2. Send the write request to external memory via an axi interface
3. Send the data via an axi interface
4. Send the return data to a buffer

- write_data 32-bit data
- indata_req_wr 1-bit request enable to u_data_buffer
- write_done Mark the last data (e.g., 16th data)

```
u_dma_write(
    .M_AWID     (M_AWID     ),  // Address ID
    .M_AWADDR   (M_AWADDR   ),  // Address Write
    .M_AWLEN    (M_AWLEN    ),  // Transfer length
    .M_AWSIZE   (M_AWSIZE   ),  // Transfer width
    .M_AWBURST  (M_AWBURST  ),  // Burst type
    .M_AWLOCK   (M_AWLOCK   ),  // Atomic access information
    .M_AWCACHE  (M_AWCACHE  ),  // Cachable/bufferable infor
    .M_AWPROT   (M_AWPROT   ),  // Protection info
    .M_AWREGION (M_AWREGION ),
    .M_AWQOS    (M_AWQOS    ),
    .M_AWVALID  (M_AWVALID  ),  // address/control valid handshake
    .M_AWREADY  (M_AWREADY  ),
    .M_AWUSER   (           ),
    //Write data channel
    .M_WID      (M_WID      ),  // Write ID
    .M_WDATA    (M_WDATA    ),  // Write Data bus
    .M_WSTRB    (M_WSTRB    ),  // Write Data byte lane strobes
    .M_WLAST    (M_WLAST    ),  // Last beat of a burst transfer
    .M_WVALID   (M_WVALID   ),  // Write data valid
    .M_WREADY   (M_WREADY   ),  // Write data ready
    .M_WUSER    (           ),
    .M_BUSER    (           ),
    //Write response chaDnel
    .M_BID      (M_BID      ),  // buffered response ID
    .M_BRESP    (M_BRESP    ),  // Buffered write response
    .M_BVALID   (M_BVALID   ),  // Response info valid
    .M_BREADY   (M_BREADY   ),  // Response info ready (to slave)
    //Read address channDl
    //User interface
    .start_dma  (ctrl_write     ),
    .num_trans  (num_trans      ), //Number of words transferred
    .start_addr (write_addr     ),
    .indata     (write_data     ),
    .indata_req_o(indata_req_wr ),
    .done_o     (write_done     ), //Blk transfer done
    .fail_check (               ),
    //User signals
    .clk        (clk            ),
    .rstn       (rstn           )
);
```

Test bench (sim/yolo_engine_tb.v)

An external memory model and a memory controller

```
//-----------------------------------------------------------------------
axi_sram_if #(  //New
    .MEM_ADDRW(MEM_ADDRW), .MEM_DW(MEM_DW),
    .A(A), .I(I), .L(L), .D(D), .M(M))
u_axi_ext_mem_if_input(
    .ACLK(clk), .ARESETn(rstn),

    //AXI Slave IF
    .AWID (M_AWID     ),        // Address ID
    .AWADDR  (M_AWADDR   ),      // Address Write
    .AWLEN   (M_AWLEN    ),       // Transfer length
    .AWSIZE  (M_AWSIZE   ),      // Transfer width
    .AWBURST (M_AWBURST  ),     // Burst type
    .AWLOCK  (M_AWLOCK   ),      // Atomic access information
    .AWCACHE (M_AWCACHE  ),     // Cachable/bufferable infor
    .AWPROT  (M_AWPROT   ),      // Protection info
    .AWVALID (M_AWVALID  ),     // address/control valid handshake
    .AWREADY (M_AWREADY  ),
    //Write data channel
    .WID     (M_WID      ),         // Write ID
    .WDATA   (M_WDATA    ),       // Write Data bus
    .WSTRB   (M_WSTRB    ),       // Write Data byte lane strobes
    .WLAST   (M_WLAST    ),       // Last beat of a burst transfer
    .WVALID  (M_WVALID   ),      // Write data valid
    .WREADY  (M_WREADY   ),      // Write data ready
    //Write response channel
    .BID     (M_BID      ),         // buffered response ID
    .BRESP   (M_BRESP    ),       // Buffered write response
    .BVALID  (M_BVALID   ),      // Response info valid
    .BREADY  (M_BREADY   ),      // Response info ready (from Master)
```

```
    .ARID    (M_ARID     ),   // Read addr ID
    .ARADDR  (M_ARADDR   ),   // Address Read
    .ARLEN   (M_ARLEN    ),   // Transfer length
    .ARSIZE  (M_ARSIZE   ),   // Transfer width
    .ARBURST (M_ARBURST  ),   // Burst type
    .ARLOCK  (M_ARLOCK   ),   // Atomic access information
    .ARCACHE (M_ARCACHE  ),   // Cachable/bufferable infor
    .ARPROT  (M_ARPROT   ),   // Protection info
    .ARVALID (M_ARVALID  ),   // address/control valid handshake
    .ARREADY (M_ARREADY  ),
    .RID     (M_RID      ),   // Read ID
    .RDATA   (M_RDATA    ),   // Read data bus
    .RRESP   (M_RRESP    ),   // Read response
    .RLAST   (M_RLAST    ),   // Last beat of a burst transfer
    .RVALID  (M_RVALID   ),   // Read data valid
    .RREADY  (M_RREADY   ),   // Read data ready (to Slave)

    //Interface to SRAM
    .mem_addr(mem_addr   ),
    .mem_we  (mem_we     ),
    .mem_di  (mem_di     ),
    .mem_do  (mem_do     )
);


// Input
// IMEM for SIM
// Inputs
sram #(
    .FILE_NAME(IFM_FILE),
    .SIZE(2**MEM_ADDRW),
    .WL_ADDR(MEM_ADDRW),
    .WL_DATA(MEM_DW))
u_ext_mem_input (
    .clk   (clk       ),
    .rst   (rstn      ),
    .addr  (mem_addr  ),
    .wdata (mem_di    ),
    .rdata (mem_do    ),
    .ena   (1'b0      )     // Read only
    );
```

Simulate the data in memory
For example
- Input images
- Model parameters
- Input/output feature maps

Test bench (sim/yolo_engine_tb.v)

- An instance of a CNN accelerator

```
//-----------------------------------------------------------
// CNN Accelerator
//-----------------------------------------------------------
reg [31:0] i_0;
reg [31:0] i_1;
reg [31:0] i_2;

yolo_engine #(
    .AXI_WIDTH_AD(A),
    .AXI_WIDTH_ID(4),
    .AXI_WIDTH_DA(D),
    .AXI_WIDTH_DS(M),
    .MEM_BASE_ADDR(2048),
    .MEM_DATA_BASE_ADDR(2048)
)
u_yolo_engine
(
    .clk(clk),
    .rstn(rstn),

    .i_ctrl_reg0(i_0     ), // network_start // {debug_big(1),
    .i_ctrl_reg1(i_1     ), // Read_address (INPUT)
    .i_ctrl_reg2(i_2     ), // Write_address
    .i_ctrl_reg3(32'd0   ), // Reserved
```

```
    .M_AWVALID (M_AWVALID),
    .M_AWREADY (M_AWREADY),
    .M_AWADDR  (M_AWADDR ),
    .M_AWID    (M_AWID   ),
    .M_AWLEN   (M_AWLEN  ),
    .M_AWSIZE  (M_AWSIZE ),
    .M_AWBURST (M_AWBURST),
    .M_AWLOCK  (M_AWLOCK ),
    .M_AWCACHE (M_AWCACHE),
    .M_AWPROT  (M_AWPROT ),
    .M_AWQOS   (         ),
    .M_AWREGION(         ),
    .M_AWUSER  (         ),

    .M_WVALID  (M_WVALID ),
    .M_WREADY  (M_WREADY ),
    .M_WDATA   (M_WDATA  ),
    .M_WSTRB   (M_WSTRB  ),
    .M_WLAST   (M_WLAST  ),
    .M_WID     (M_WID    ),
    .M_WUSER   (         ),

    .M_BVALID  (M_BVALID ),
    .M_BREADY  (M_BREADY ),
    .M_BRESP   (M_BRESP  ),
    .M_BID     (M_BID    ),
    .M_BUSER   (         ),

    .network_done(network_done),
    .network_done_led(network_done_led)
);
```

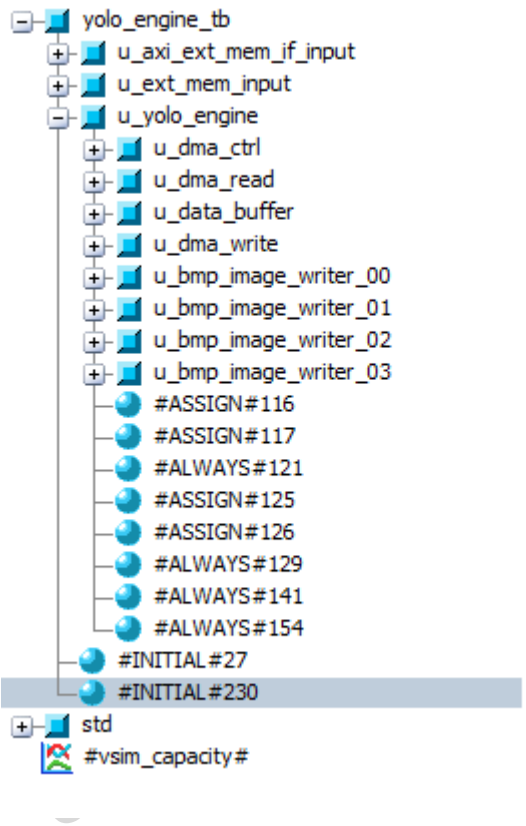
Test bench (sim/yolo_engine_tb.v)

- We can control and monitor the CNN accelerator
  - ✓ Start: i_0 ← 1
  - ✓ Send other configurations as you want
    - ✓ i_1, i_2
  - ✓ Preload:
    - ✓ You may test a particular layer
    - ✓ For example, compare the results from HW with that from the software.
  - ✓ Wait until the layer/network completes

```
223 initial begin
224     rstn = 1'b0;          // Reset, low active
225     i_0 = 0;
226     i_1 = 0;
227     i_2 = (4*256*256)*4;
228 `ifdef DEBUG
229     resume_counter = 0;
230 `endif
231 `ifdef PRELOAD
232     preload = 1'b0;
233     preload_layer_idx = 4;
234 `endif
235
236     #(4*CLK_PERIOD) rstn = 1'b1;
237     #(100*CLK_PERIOD)
238         @(posedge clk)
239             i_0 = 32'd1;
240      `ifdef PRELOAD
241             preload <= 1'b1;
242             preload_layer_idx <= 4;
243      `endif
244     #(100*CLK_PERIOD)
245         @(posedge clk)
246             i_0 = 32'd0;
247     while(!network_done) begin
248         #(128*CLK_PERIOD) @(posedge clk);
249     end
250     #(100*CLK_PERIOD)
251         @(posedge clk) $stop;
252 end
```

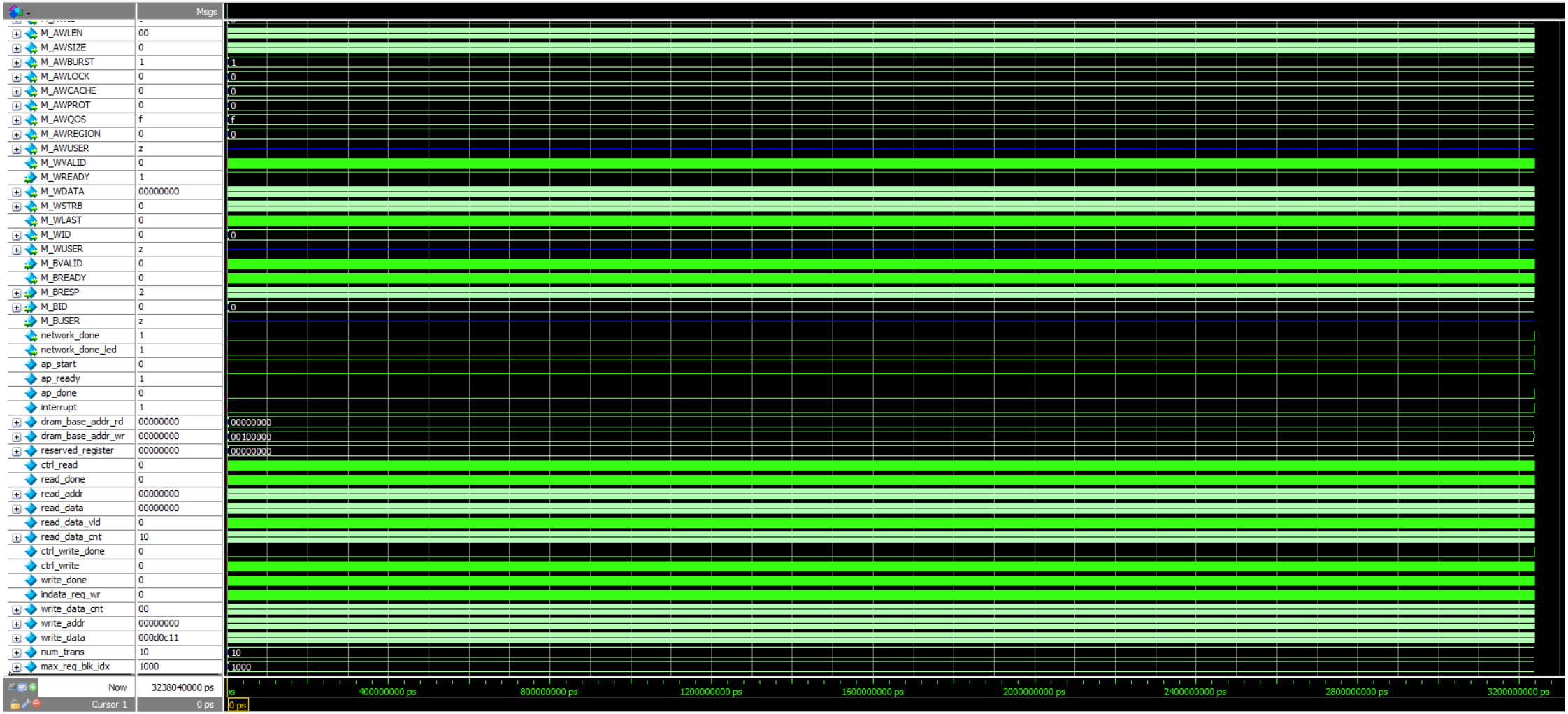
Test bench (sim/yolo_engine_tb.v)

- Simulate the top design with time = 4ms




```
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/dram_base_addr_rd
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/dram_base_addr_wr
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/reserved_register
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/ctrl_read
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/read_done
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/read_addr
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/read_data
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/read_data_vld
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/read_data_cnt
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/ctrl_write_done
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/ctrl_write
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/write_done
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/indata_req_wr
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/write_data_cnt
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/write_addr
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/write_data
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/num_trans
add wave -position end  sim:/yolo_engine_tb/u_yolo_engine/max_req_blk_idx
VSIM 263> run 4ms
# Initializing memory 'SimmemSync_rp0_wp0_cpl'..
# Loading memory 'SimmemSync_rp0_wp0_cpl' from file: C:/yolohw/sim/inout_data_sw/log_feamap/CONV00_input_16b.hex
# 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
# ** Note: $stop    : C:/yolohw/sim/yolo_engine_tb.v(251)
#    Time: 3238880 ns  Iteration: 1  Instance: /yolo_engine_tb
# Break in Module yolo_engine_tb at C:/yolohw/sim/yolo_engine_tb.v line 251
```

Test bench (sim/yolo_engine_tb.v)



Test bench (sim/yolo_engine_tb.v)