

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
```

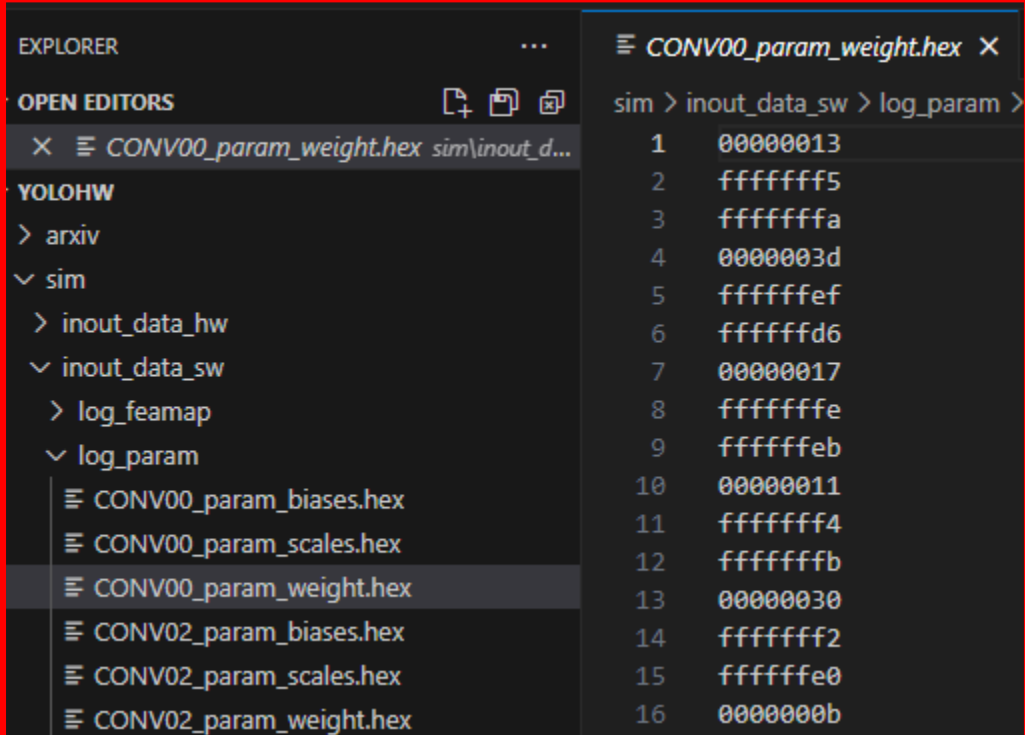
EXPLORER	
... CONV00_input_32b.hex X	
sim > inout_data_sw > log_feamap	
1 00707064	
2 006f6f63	
3 0066685d	
4 005e5f56	
5 005d5d56	
6 00595b54	
7 0055574f	
8 0054554f	
9 0053544e	
10 004f524b	
11 004c504b	
12 00494d48	
13 00494a46	
14 00484a45	
15 00474944	
16 00424643	

- 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
```



CONV00_param_weight.hex	
1	00000013
2	fffffffff5
3	ffffffffa
4	0000003d
5	ffffffef
6	ffffffd6
7	00000017
8	fffffffe
9	ffffffeb
10	00000011
11	fffffff4
12	fffffffb
13	00000030
14	fffffff2
15	ffffffe0
16	0000000b

- Each line has 32 bits
=> 8 LSB bits are used for one filter coefficient

conv_layer_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
78 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)
⇒ 64 multipliers
- Input/Output
 - Inputs:
 - vld_i Input valid signal
 - win Filters => four filters
 - din Window data
 - Output
 - vld_o Output valid signal
 - acc_o Accumulated results ($\sum_{i=0}^{15} w_i * d_i$)

```
152 //-----  
153 // DUT: MACs  
154 //-----  
155 mac u_mac_00(  
156   /*input      */clk (clk   ),  
157   /*input      */rstn (rstn  ),  
158   /*input      */vld_i(vld_i ),  
159   /*input [127:0]*/win (win[0] ),  
160   /*input [127:0]*/din (din  ),  
161   /*output [ 19:0]*/acc_o(acc_o[0]),  
162   /*output      */vld_o(vld_o[0])  
163 );  
164 mac u_mac_01(  
165   /*input      */clk (clk   ),  
166   /*input      */rstn (rstn  ),  
167   /*input      */vld_i(vld_i ),  
168   /*input [127:0]*/win (win[1] ),  
169   /*input [127:0]*/din (din  ),  
170   /*output [ 19:0]*/acc_o(acc_o[1]),  
171   /*output      */vld_o(vld_o[1])  
172 );  
173 mac u_mac_02(  
174   /*input      */clk (clk   ),  
175   /*input      */rstn (rstn  ),  
176   /*input      */vld_i(vld_i ),  
177   /*input [127:0]*/win (win[2] ),  
178   /*input [127:0]*/din (din  ),  
179   /*output [ 19:0]*/acc_o(acc_o[2]),  
180   /*output      */vld_o(vld_o[2])  
181 );  
182 mac u_mac_03(  
183   /*input      */clk (clk   ),  
184   /*input      */rstn (rstn  ),  
185   /*input      */vld_i(vld_i ),  
186   /*input [127:0]*/win (win[3] ),  
187   /*input [127:0]*/din (din  ),  
188   /*output [ 19:0]*/acc_o(acc_o[3]),  
189   /*output      */vld_o(vld_o[3])  
190 );
```

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)

```
84 // *****
85 // Loop for convolutions
86 // *****
87 //{{{
88 #(100*CLK_PERIOD)
89   for(row = 0; row < IFM_HEIGHT; row = row + 1) begin
90     @(posedge clk)
91       ctrl_data_run = 0;
92     #(100*CLK_PERIOD) @(posedge clk);
93     ctrl_data_run = 1;
94     for (col = 0; col < IFM_WIDTH; col = col + 1) begin
95       for (chn = 0; chn < IFM_CHANNEL; chn = chn + 1) begin
96         @(posedge clk) begin
97           if((col == IFM_WIDTH-1) && (chn == IFM_CHANNEL-1))
98             ctrl_data_run = 0;
99         end
100       end
101     end
102   end
103   @(posedge clk)
104   ctrl_data_run = 1'b0;
105 //}}}
```


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.

```
111 // Generate din, win
112 wire is_first_row = (row == 0) ? 1'b1: 1'b0;
113 wire is_last_row = (row == IFM_HEIGHT-1) ? 1'b1: 1'b0;
114 wire is_first_col = (col == 0) ? 1'b1: 1'b0;
115 wire is_last_col = (col == IFM_WIDTH-1) ? 1'b1 : 1'b0;
116
117 always@(*) begin
118     vld_i = 0;
119     din = 128'd0;
120     win[0] = 0;
121     win[1] = 0;
122     win[2] = 0;
123     win[3] = 0;
124     if(ctrl_data_run) begin
125         vld_i = 1;
126         // Tiled IFM data
127         din[ 7: 0] = (is_first_row || is_first_col) ? 8'd0 : in_img[(row-1) * IFM_WIDTH + (col-1)][chn*8+8];
128         din[15: 8] = (is_first_row || is_first_col) ? 8'd0 : in_img[(row-1) * IFM_WIDTH + col][chn*8+8];
129         din[23:16] = (is_first_row || is_last_col) ? 8'd0 : in_img[(row-1) * IFM_WIDTH + (col+1)][chn*8+8];
130
131         din[31:24] = (is_first_col) ? 8'd0 : in_img[ row * IFM_WIDTH + (col-1)][chn*8+8];
132         din[39:32] = in_img[ row * IFM_WIDTH + col][chn*8+8];
133         din[47:40] = (is_last_col) ? 8'd0 : in_img[ row * IFM_WIDTH + (col+1)][chn*8+8];
134
135         din[55:48] = (is_last_row || is_first_col) ? 8'd0 : in_img[(row+1) * IFM_WIDTH + (col-1)][chn*8+8];
136         din[63:56] = (is_last_row || is_first_col) ? 8'd0 : in_img[(row+1) * IFM_WIDTH + col][chn*8+8];
137         din[71:64] = (is_last_row || is_last_col) ? 8'd0 : in_img[(row+1) * IFM_WIDTH + (col+1)][chn*8+8];
138         // Tiled Filters
139         for(j = 0; j < 4; j=j+1) begin // Four sets <=> Four output channels
140             win[j][ 7: 0] = filter[(j*Fx*Fy*Ni) + chn*9][7:0];
141             win[j][15: 8] = filter[(j*Fx*Fy*Ni) + chn*9 + 1][7:0];
142             win[j][23:16] = filter[(j*Fx*Fy*Ni) + chn*9 + 2][7:0];
143             win[j][31:24] = filter[(j*Fx*Fy*Ni) + chn*9 + 3][7:0];
144             win[j][39:32] = filter[(j*Fx*Fy*Ni) + chn*9 + 4][7:0];
145             win[j][47:40] = filter[(j*Fx*Fy*Ni) + chn*9 + 5][7:0];
146             win[j][55:48] = filter[(j*Fx*Fy*Ni) + chn*9 + 6][7:0];
147             win[j][63:56] = filter[(j*Fx*Fy*Ni) + chn*9 + 7][7:0];
148             win[j][71:64] = filter[(j*Fx*Fy*Ni) + chn*9 + 8][7:0];
149         end
150     end
151 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)

```
193 reg [31:0] psum[0:3];
194 wire valid_out = vld_o[0];
195
196 always@(posedge clk, negedge rstn) begin
197     if(!rstn) begin
198         chn_idx <= 0;
199     end
200     else begin
201         if(valid_out) begin
202             if(chn_idx == IFM_CHANNEL-1)
203                 chn_idx <= 0;
204             else
205                 chn_idx <= chn_idx + 1;
206         end
207     end
208 end
209 reg write_pixel_ena;
210 always@(posedge clk, negedge rstn) begin
211     if(!rstn) begin
212         psum[0] <= 0;
213         psum[1] <= 0;
214         psum[2] <= 0;
215         psum[3] <= 0;
216         write_pixel_ena <= 0;
217     end
218     else begin
219         if(valid_out) begin
220             if(chn_idx == 0) begin
221                 psum[0] <= $signed(acc_o[0]);
222                 psum[1] <= $signed(acc_o[1]);
223                 psum[2] <= $signed(acc_o[2]);
224                 psum[3] <= $signed(acc_o[3]);
225             end
226             else begin
227                 psum[0] <= $signed(psum[0]) + $signed(acc_o[0]);
228                 psum[1] <= $signed(psum[1]) + $signed(acc_o[1]);
229                 psum[2] <= $signed(psum[2]) + $signed(acc_o[2]);
230                 psum[3] <= $signed(psum[3]) + $signed(acc_o[3]);
231             end
232
233             if(chn_idx == IFM_CHANNEL-1)
234                 write_pixel_ena <= 1;
235             else
236                 write_pixel_ena <= 0;
237         end
238     end
239     write_pixel_ena <= 0;
240 end
241 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

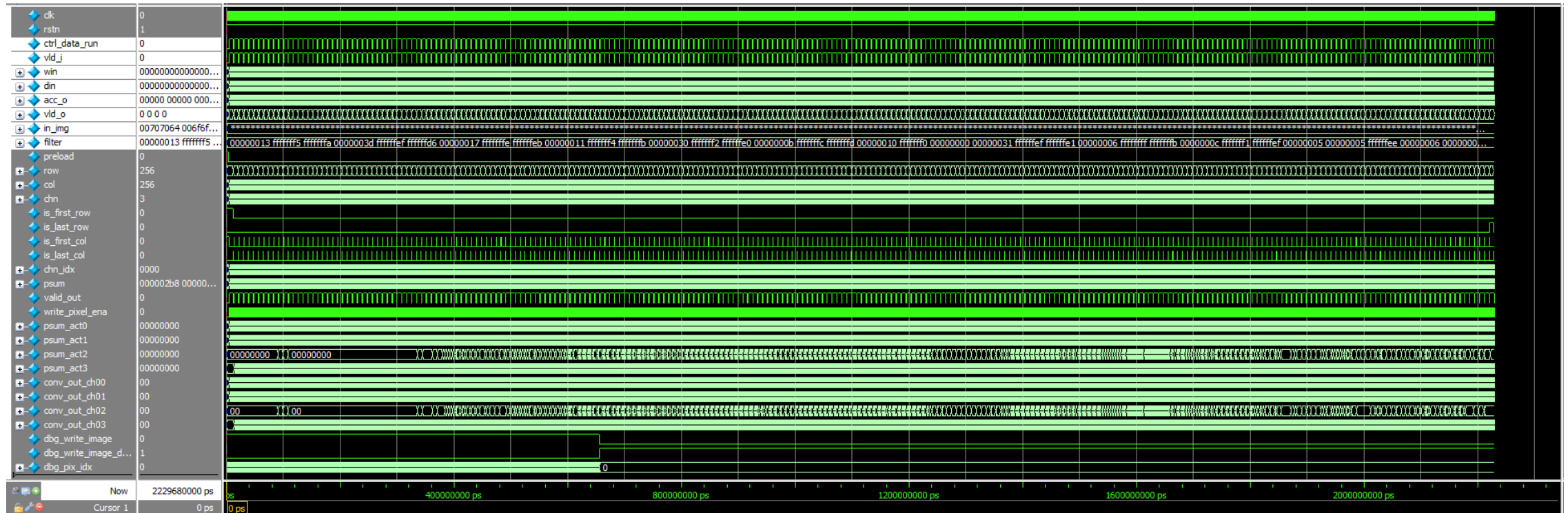
```
233         if(chn_idx == IFM_CHANNEL-1)
234             write_pixel_ena <= 1;
235         else
236             write_pixel_ena <= 0;
237         end
238     else
239         write_pixel_ena <= 0;
240     end
241 end
242 //-----
243 // For debugging: INPUTs/Outputs
244 //-----
245 // Activation + Quantization (Descaling)
246 //{{{
247 wire [31:0] psum_act0 = write_pixel_ena ? ((psum[0][31]==1)?0:psum[0]): 0;
248 wire [31:0] psum_act1 = write_pixel_ena ? ((psum[1][31]==1)?0:psum[1]): 0;
249 wire [31:0] psum_act2 = write_pixel_ena ? ((psum[2][31]==1)?0:psum[2]): 0;
250 wire [31:0] psum_act3 = write_pixel_ena ? ((psum[3][31]==1)?0:psum[3]): 0;
251
252 wire [7:0] conv_out_ch00 = write_pixel_ena?((psum_act0[31:7]>255)?255:psum_act0[14:7]):0; // Descaling: * 1/2^11
253 wire [7:0] conv_out_ch01 = write_pixel_ena?((psum_act1[31:7]>255)?255:psum_act1[14:7]):0; // Descaling: * 1/2^11
254 wire [7:0] conv_out_ch02 = write_pixel_ena?((psum_act2[31:7]>255)?255:psum_act2[14:7]):0; // Descaling: * 1/2^11
255 wire [7:0] conv_out_ch03 = write_pixel_ena?((psum_act3[31:7]>255)?255:psum_act3[14:7]):0; // Descaling: * 1/2^11
256
257
```

conv_layer_tb.v: Simulation time = 3ms

```
conv_layer_tb
+ PROC_SimmemLoad
+ u_mac_00
+ u_mac_01
+ u_mac_02
+ u_mac_03
+ u_acc_img_ch0
+ u_acc_img_ch1
+ u_acc_img_ch2
+ u_acc_img_ch3
+ u_jfm_img_ch0
+ u_jfm_img_ch1
+ u_jfm_img_ch2
+ u_jfm_img_ch3
+ #INITIAL#19
+ #INITIAL#33(PROC_SimmemLoad)
+ #INITIAL#55
+ #ASSIGN#112
+ #ASSIGN#113
+ #ASSIGN#114
+ #ASSIGN#115
+ #ALWAYS#117
+ #ASSIGN#194
+ #ALWAYS#196
+ #ALWAYS#210

VIM 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

```
321 bmp_image_writer #(.OUTFILE(CONV_INPUT_IMG00),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
322 u_ifm_img_ch0(
323     /*input      */clk      (clk      ),
324     /*input      */rstn     (rstn     ),
325     /*input [WI-1:0] */din    (in_img[dbg_pix_idx][7:0] ),
326     /*input      */vld      (dbg_write_image ),
327     /*output reg  */frame_done(
328 );
329 bmp_image_writer #(.OUTFILE(CONV_INPUT_IMG01),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
330 u_ifm_img_ch1(
331     /*input      */clk      (clk      ),
332     /*input      */rstn     (rstn     ),
333     /*input [WI-1:0] */din    (in_img[dbg_pix_idx][15:8] ),
334     /*input      */vld      (dbg_write_image ),
335     /*output reg  */frame_done(
336 );
337 bmp_image_writer #(.OUTFILE(CONV_INPUT_IMG02),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
338 u_ifm_img_ch2(
339     /*input      */clk      (clk      ),
340     /*input      */rstn     (rstn     ),
341     /*input [WI-1:0] */din    (in_img[dbg_pix_idx][23:16] ),
342     /*input      */vld      (dbg_write_image ),
343     /*output reg  */frame_done(
344 );
345 bmp_image_writer #(.OUTFILE(CONV_INPUT_IMG03),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
346 u_ifm_img_ch3(
347     /*input      */clk      (clk      ),
348     /*input      */rstn     (rstn     ),
349     /*input [WI-1:0] */din    (in_img[dbg_pix_idx][31:24] ),
350     /*input      */vld      (dbg_write_image ),
351     /*output reg  */frame_done(
352 );
```



conv_layer_tb.v: Visual Verification

```
258 bmp_image_writer #(.OUTFILE(CONV_OUTPUT_IMG00),.WIDTH(IFM_WIDTH),.HEIGHT(IFM_HEIGHT))
259 v 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 v 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 v 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 v 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

Copyright 2024. (차세대반도체 혁신공유대학 사업단) all rights reserved.

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
 - $din \leftarrow in_img[row*WIDTH+col]$
 - 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 x  cnn_ctrl.v x
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;
```

Test bench (cnn_ctrl_multi_layer_tb.v)

Layer configurations
q_width, q_height, q_frame_size, ...



Control signals
ctrl_vsync_run, ctrl_hsync_run, ...
row, col, data_count, end_frame

```
32 //-----
33 // Controller (FSM)
34 //-----
35 cnn_ctrl u_cnn_ctrl (
36   .clk      (clk      ),
37   .rstn     (rstn     ),
38   // Inputs
39   .q_width  (q_width  ),
40   .q_height (q_height ),
41   .q_vsync_delay (q_vsync_delay ),
42   .q_hsync_delay (q_hsync_delay ),
43   .q_frame_size (q_frame_size ),
44   .q_start   (q_start ),
45   //output
46   .o_ctrl_vsync_run(ctrl_vsync_run),
47   .o_ctrl_vsync_cnt(ctrl_vsync_cnt),
48   .o_ctrl_hsync_run(ctrl_hsync_run),
49   .o_ctrl_hsync_cnt(ctrl_hsync_cnt),
50   .o_ctrl_data_run(ctrl_data_run ),
51   .o_row      (row      ),
52   .o_col      (col      ),
53   .o_data_count (data_count ),
54   .o_end_frame (end_frame ),
55 );
56
57 // Clock
58 parameter CLK_PERIOD = 10; //100MHz
59 initial begin
60     clk = 1'b1;
61     forever #(CLK_PERIOD/2) clk = ~clk;
62 end
```

Test cases: Layer 00

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

```
64 //-----  
65 // Test cases  
66 //-----  
67 initial begin  
68     rstn = 1'b0;           // Reset, low active  
69     q_width    = WIDTH;  
70     q_height   = HEIGHT;  
71     q_vsync_delay = VSYNC_DELAY;  
72     q_hsync_delay = HSYNC_DELAY;  
73     q_frame_size = FRAME_SIZE;  
74     q_start    = 1'b0;  
75  
76     #(4*CLK_PERIOD) rstn = 1'b1;  
77  
78     //-----  
79     // Layer 1: width = height = 256  
80     //-----  
81     q_width  = 256;  
82     q_height = 256;  
83     q_frame_size = 256*256;  
84     #(100*CLK_PERIOD)  
85     |@ (posedge clk)  
86     |    q_start = 1'b1;  
87     #(4*CLK_PERIOD)  
88     |@ (posedge clk)  
89     |    q_start = 1'b0;  
90     while(!layer_done) begin  
91         #(128*CLK_PERIOD) @(posedge clk);  
92     end  
93     $display("CONV_00: Done !!!");  
94
```

Test cases: Layers 02, 04

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

```
95 //-----  
96 // Layer 2: width = height = 128  
97 //-----  
98 q_width = 128;  
99 q_height = 128;  
100 q_frame_size = 128*128;  
101 # (100*CLK_PERIOD)  
102   @(posedge clk)  
103     q_start = 1'b1;  
104 # (4*CLK_PERIOD)  
105   @(posedge clk)  
106     q_start = 1'b0;  
107 while(!layer_done) begin  
108   # (128*CLK_PERIOD) @(posedge clk);  
109 end  
110 $display("CONV_02: Done !!!");  
111  
112 //-----  
113 // Layer 3: width = height = 64  
114 //-----  
115 q_width = 64;  
116 q_height = 64;  
117 q_frame_size = 64*64;  
118 # (100*CLK_PERIOD)  
119   @(posedge clk)  
120     q_start = 1'b1;  
121 # (4*CLK_PERIOD)  
122   @(posedge clk)  
123     q_start = 1'b0;  
124 while(!layer_done) begin  
125   # (128*CLK_PERIOD) @(posedge clk);  
126 end  
127 $display("CONV_04: Done !!!");  
128 # (100*CLK_PERIOD)  
129   @(posedge clk) $stop;
```

Simulation Results: Run 2ms

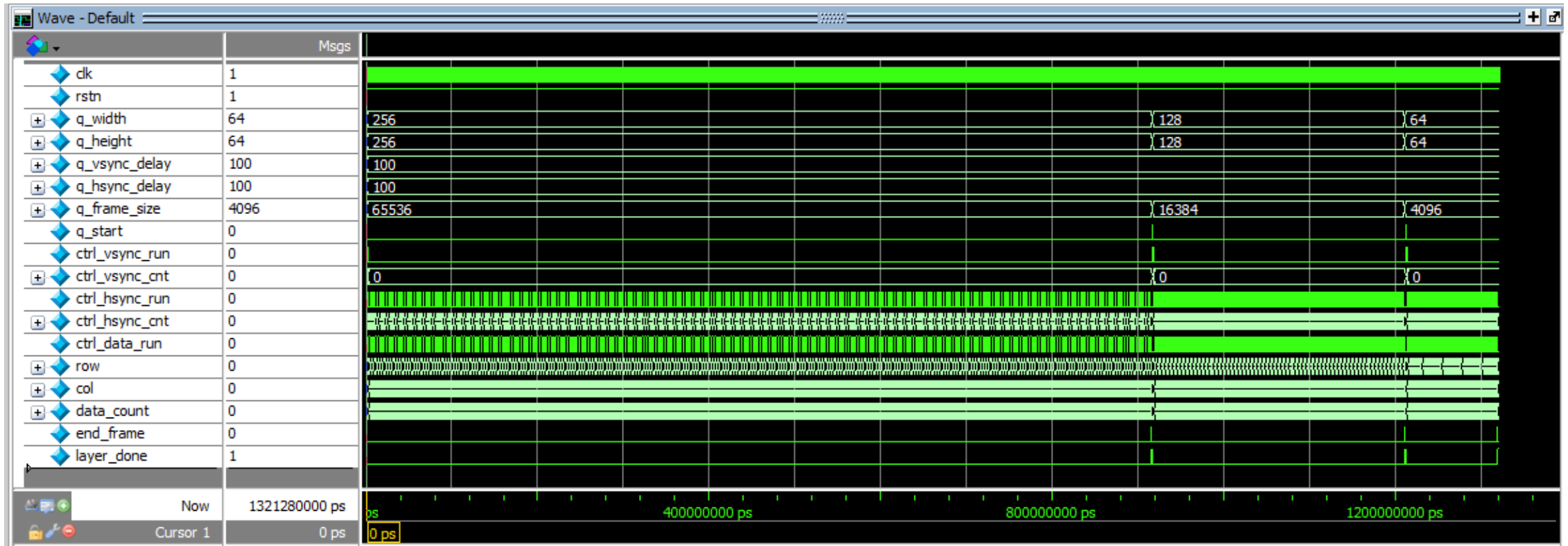
The screenshot displays a Verilog simulation environment with the following components:

- Instance Tree:** Shows the hierarchy starting with `cnn_ctrl_multi_layer_tb`, which contains `u_cnn_ctrl` and several initial values (`#INITIAL#59`, `#INITIAL#67`, `#ALWAYS#132`). A standard library (`std`) and a capacity parameter (`#vsim_capacity#`) are also listed.
- Parameter Table:** A table with columns `Name`, `Value`, and `Kind`. It lists parameters for the `cnn_ctrl_multi_layer_tb` instance:

Name	Value	Kind
W_DELAY	12	Para... Internal
WIDTH	256	Para... Internal
HEIGHT	256	Para... Internal
FRAME_SIZE	65536	Para... Internal
- Waveform List:** A list of signals added for waveform viewing, including `sim:/cnn_ctrl_multi_layer_tb/q_height`, `sim:/cnn_ctrl_multi_layer_tb/q_vsync_delay`, `sim:/cnn_ctrl_multi_layer_tb/q_hsync_delay`, `sim:/cnn_ctrl_multi_layer_tb/q_frame_size`, `sim:/cnn_ctrl_multi_layer_tb/q_start`, `sim:/cnn_ctrl_multi_layer_tb/ctrl_vsync_run`, `sim:/cnn_ctrl_multi_layer_tb/ctrl_vsync_cnt`, `sim:/cnn_ctrl_multi_layer_tb/ctrl_hsync_run`, `sim:/cnn_ctrl_multi_layer_tb/ctrl_hsync_cnt`, `sim:/cnn_ctrl_multi_layer_tb/ctrl_data_run`, `sim:/cnn_ctrl_multi_layer_tb/row`, `sim:/cnn_ctrl_multi_layer_tb/col`, `sim:/cnn_ctrl_multi_layer_tb/data_count`, `sim:/cnn_ctrl_multi_layer_tb/end_frame`, and `sim:/cnn_ctrl_multi_layer_tb/layer_done`.
- Console Output:** The bottom pane shows the execution log:

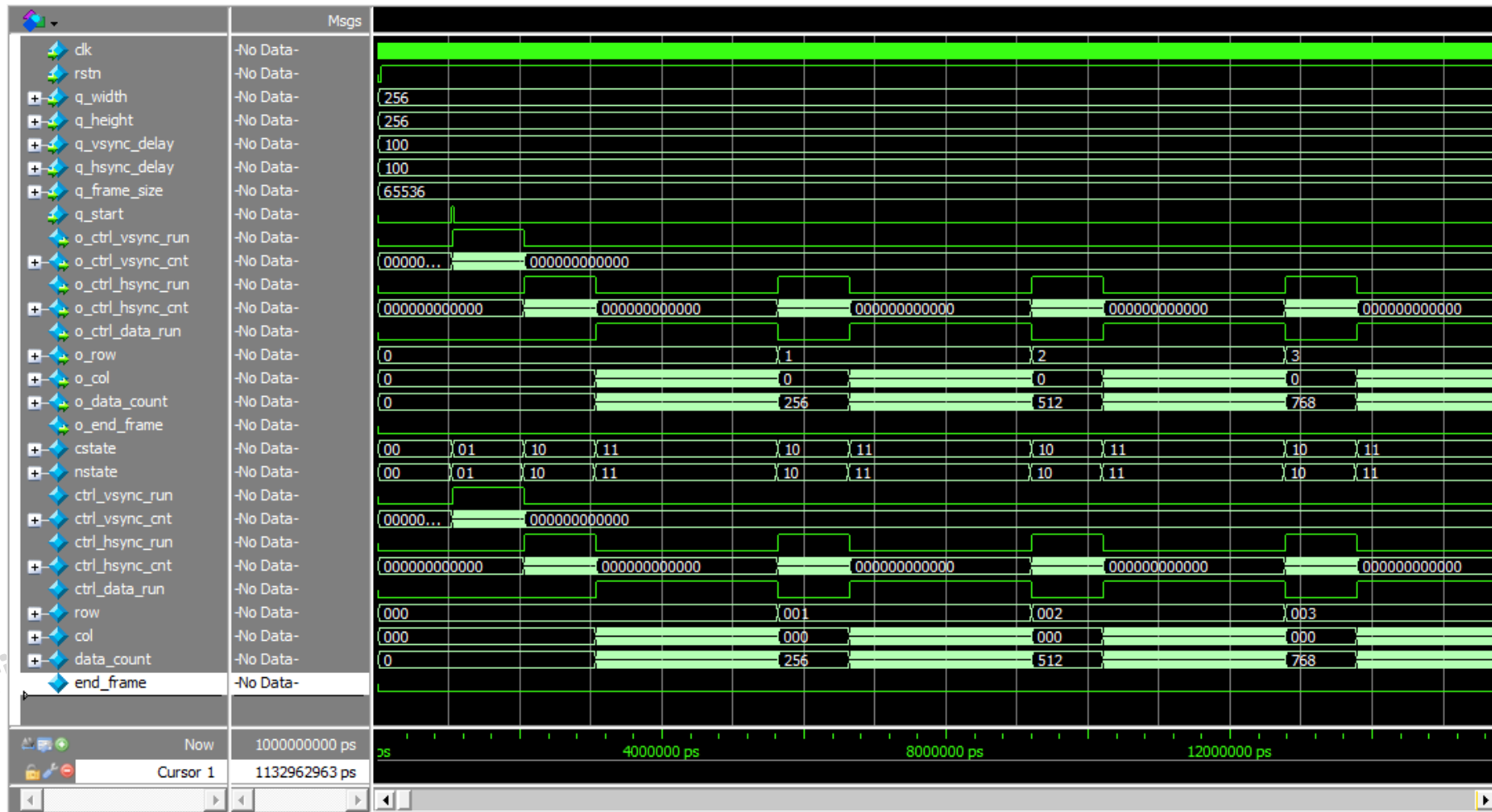
```
VSIM 175> run 2ms
# CONV_00: Done !!!
# CONV_02: Done !!!
# CONV_04: Done !!!
# ** Note: $stop      : C:/yolohw/sim/cnn_ctrl_multi_layer_tb.v(129)
#   Time: 1321280 ns  Iteration: 1  Instance: /cnn_ctrl_multi_layer_tb
# Break in Module cnn_ctrl_multi_layer_tb at C:/yolohw/sim/cnn_ctrl_multi_layer_tb.v line 129
```

Simulation Results: Layers 0, 2, 4

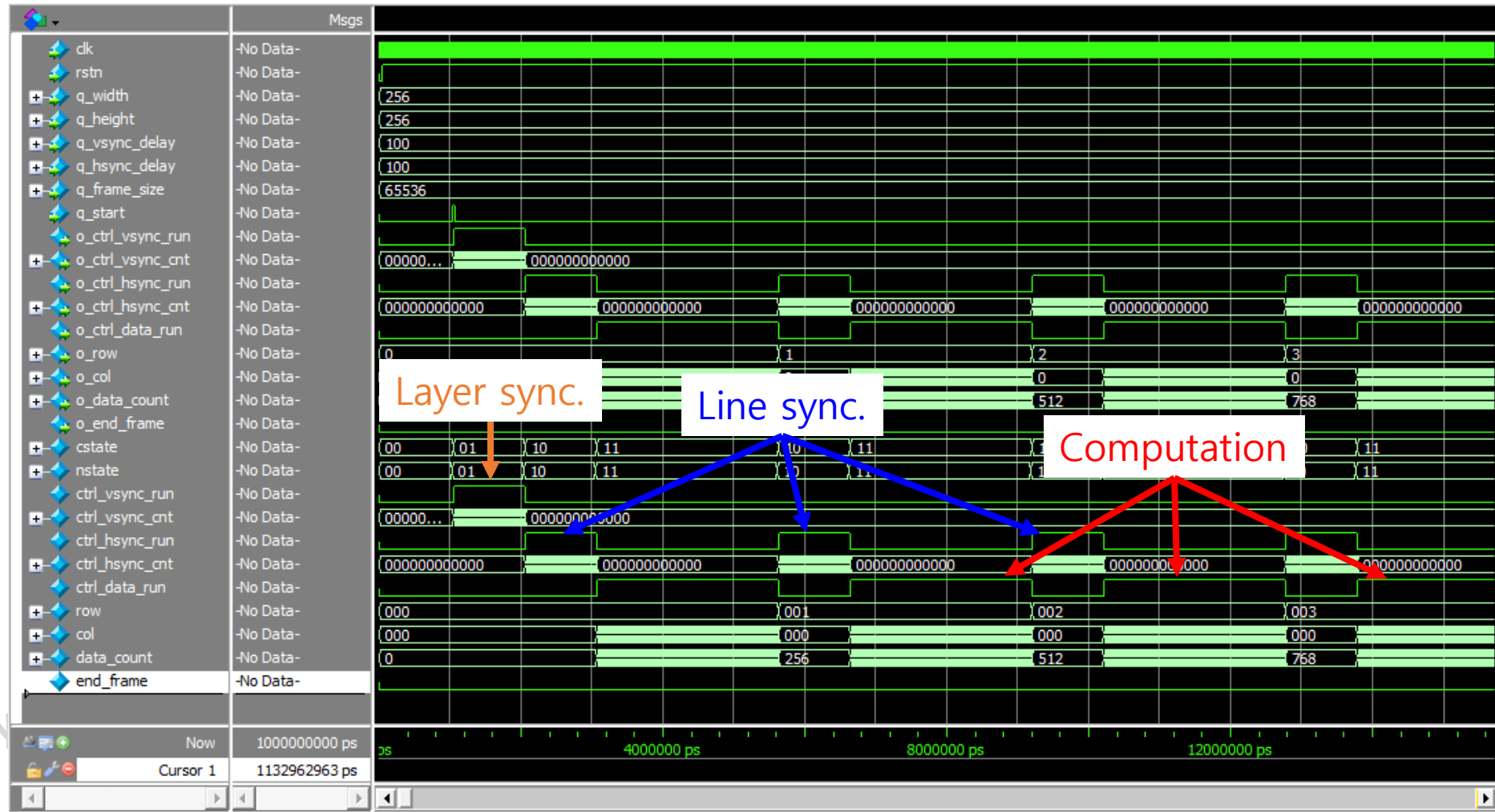


Waveform (cnn_ctrl_tb.v)

- 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

Copyright 2024. (차세대반도체 혁신공유대학 사업단) all rights reserved.

Top module: yolo_engine.v

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

```
9 module yolo_engine #(
10     parameter AXI_WIDTH_AD = 32,
11     parameter AXI_WIDTH_ID = 4,
12     parameter AXI_WIDTH_DA = 32,
13     parameter AXI_WIDTH_DS = AXI_WIDTH_DA/8,
14     parameter OUT_BITS_TRANS = 18,
15     parameter WBUF_AW = 9,
16     parameter WBUF_DW = 8*3*3*16,
17     parameter WBUF_DS = WBUF_DW/8,
18     parameter MEM_BASE_ADDR = 'h8000_0000,
19     parameter MEM_DATA_BASE_ADDR = 4096
20 )
21
22     input                clk
23     , input              rstn
24
25     , input [31:0] i_ctrl_reg0 // network_start, // {debug_big(1)
26     , input [31:0] i_ctrl_reg1 // Read address base
27     , input [31:0] i_ctrl_reg2 // Write address base
28     , input [31:0] i_ctrl_reg3 // Write address base
29
30     , output          M_ARVALID
31     , input           M_ARREADY
32     , output [AXI_WIDTH_AD-1:0] M_ARADDR
33     , output [AXI_WIDTH_ID-1:0] M_ARID
34     , output [7:0] M_ARLEN
35     , output [2:0] M_ARSIZE
36     , output [1:0] M_ARBURST
37     , output [1:0] M_ARLOCK
38     , output [3:0] M_ARCACHE
39     , output [2:0] M_ARPROT
40     , output [3:0] M_ARQOS
41     , output [3:0] M_ARREGION
42     , output [3:0] M_ARUSER
43     , input          M_RVALID
44     , output         M_RREADY
45     , input [AXI_WIDTH_DA-1:0] M_RDATA
46     , input          M_RLAST
47     , input [AXI_WIDTH_ID-1:0] M_RID
48     , input [3:0] M_RUSER
49     , input [1:0] M_RRESP
```

```
50
51     , output          M_AWVALID
52     , input           M_AWREADY
53     , output [AXI_WIDTH_AD-1:0] M_AWADDR
54     , output [AXI_WIDTH_ID-1:0] M_AWID
55     , output [7:0] M_AWLEN
56     , output [2:0] M_AWSIZE
57     , output [1:0] M_AWBURST
58     , output [1:0] M_AWLOCK
59     , output [3:0] M_AWCACHE
60     , output [2:0] M_AWPROT
61     , output [3:0] M_AWQOS
62     , output [3:0] M_AWREGION
63     , output [3:0] M_AWUSER
64
65     , output          M_WVALID
66     , input           M_WREADY
67     , output [AXI_WIDTH_DA-1:0] M_WDATA
68     , output [AXI_WIDTH_DS-1:0] M_WSTRB
69     , output          M_WLAST
70     , output [AXI_WIDTH_ID-1:0] M_WID
71     , output [3:0] M_WUSER
72
73     , input           M_BVALID
74     , output         M_BREADY
75     , input [1:0] M_BRESP
76     , input [AXI_WIDTH_ID-1:0] M_BID
77     , input          M_BUSER
78
79     , output network_done
80     , output network_done_led
81 );
82 `include "define.v"
83
```

Top module (yolo_engine.v): Control signals

```
118 //-----
119 // Control signals
120 //-----
121 always @ (*) begin
122     ap_done    = ctrl_write_done;
123     ap_ready   = 1;
124 end
125 assign network_done    = interrupt;
126 assign network_done_led = interrupt;
127
128
129 always @ (posedge clk, negedge rstn) begin
130     if(~rstn) begin
131         ap_start <= 0;
132     end
133     else begin
134         if(!ap_start && i_ctrl_reg0[0])
135             ap_start <= 1;
136         else if (ap_done)
137             ap_start <= 0;
138     end
139 end
140
141 always @(posedge clk, negedge rstn) begin
142     if(~rstn) begin
143         interrupt <= 0;
144     end
145     else begin
146         if(i_ctrl_reg0[0])
147             interrupt <= 0;
148         else if (ap_done)
149             interrupt <= 1;
150     end
151 end
```

Top module (yolo_engine.v):

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

```
153 // Parse the control registers
154 always @ (posedge clk, negedge rstn) begin
155     if(~rstn) begin
156         dram_base_addr_rd <= 0;
157         dram_base_addr_wr <= 0;
158         reserved_register <= 0; // unused
159     end
160     else begin
161         if(!ap_start && i_ctrl_reg0[0]) begin
162             dram_base_addr_rd <= i_ctrl_reg1; // Base Address for READ (Input image, Model parameters)
163             dram_base_addr_wr <= i_ctrl_reg2; // Base Address for WRITE (Intermediate feature maps, Outputs)
164             reserved_register <= i_ctrl_reg3; // reserved
165         end
166         else if (ap_done) begin
167             dram_base_addr_rd <= 0;
168             dram_base_addr_wr <= 0;
169             reserved_register <= 0;
170         end
171     end
172 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)

```
173 //-----
174 // DUTs
175 //-----
176 // DMA Controller
177 axi_dma_ctrl #(.BIT_TRANS(BIT_TRANS))
178 u_dma_ctrl(
179     .clk          (clk          )
180     ,.rstn        (rstn         )
181     ,.i_start     (i_ctrl_reg0[0] )
182     ,.i_base_address_rd(dram_base_addr_rd)
183     ,.i_base_address_wr(dram_base_addr_wr)
184     ,.i_num_trans (num_trans    )
185     ,.i_max_req_blk_idx(max_req_blk_idx )
186     // DMA Read
187     ,.i_read_done  (read_done   )
188     ,.o_ctrl_read  (ctrl_read   )
189     ,.o_read_addr  (read_addr   )
190     // DMA Write
191     ,.i_indata_req_wr (indata_req_wr )
192     ,.i_write_done  (write_done  )
193     ,.o_ctrl_write  (ctrl_write  )
194     ,.o_write_addr  (write_addr  )
195     ,.o_write_data_cnt (write_data_cnt )
196     ,.o_ctrl_write_done(ctrl_write_done )
197 );
```

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)

```
210 u_dma_read(  
211     //AXI Master Interface  
212     //Read address channel  
213     .M_ARVALID (M_ARVALID), // address/control valid handshake  
214     .M_ARREADY (M_ARREADY), // Read addr ready  
215     .M_ARADDR (M_ARADDR), // Address Read  
216     .M_ARID (M_ARID), // Read addr ID  
217     .M_ARLEN (M_ARLEN), // Transfer length  
218     .M_ARSIZE (M_ARSIZE), // Transfer width  
219     .M_ARBURST (M_ARBURST), // Burst type  
220     .M_ARLOCK (M_ARLOCK), // Atomic access information  
221     .M_ARCACHE (M_ARCACHE), // Cachable/bufferable infor  
222     .M_ARPROT (M_ARPROT), // Protection info  
223     .M_ARQOS (M_ARQOS), // Quality of Service  
224     .M_ARREGION (M_ARREGION), // Region signaling  
225     .M_ARUSER (M_ARUSER), // User defined signal  
226  
227     //Read data channel  
228     .M_RVALID (M_RVALID), // Read data valid  
229     .M_RREADY (M_RREADY), // Read data ready (to Slave)  
230     .M_RDATA (M_RDATA), // Read data bus  
231     .M_RLAST (M_RLAST), // Last beat of a burst transfer  
232     .M_RID (M_RID), // Read ID  
233     .M_RUSER (M_RUSER), // User defined signal  
234     .M_RRESP (M_RRESP), // Read response  
235  
236     //Functional Ports  
237     .start_dma (ctrl_read),  
238     .num_trans (num_trans), //Number of 128-bit words transferred  
239     .start_addr (read_addr), //iteration_num * 4 * 16 + read_address_d  
240     .data_o (read_data),  
241     .data_vld_o (read_data_vld),  
242     .data_cnt_o (read_data_cnt),  
243     .done_o (read_done),  
244  
245     //Global signals  
246     .clk (clk),  
247     .rstn (rstn)  
248 );
```


Top module: Data buffer

- A buffer saves the data coming from u_dma_read

```
251 // dpram_256x32
252 v dpram_wrapper #(
253     .DEPTH (BUFF_DEPTH    ),
254     .AW     (BUFF_ADDR_W   ),
255     .DW     (AXI_WIDTH_DA  )
256 v u_data_buffer(
257     .clk     (clk           ),
258     .ena     (1'd1         ),
259     .addra   (read_data_cnt ),
260     .wea     (read_data_vld ),
261     .dia     (read_data     ),
262     .enb     (1'd1         ), // Always Read
263     .addrb   (write_data_cnt),
264     .dob     (write_data    )
265 );
266
```

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)

```
277 u_dma_write(  
278     .M_AWID      (M_AWID      ), // Address ID  
279     .M_AWADDR    (M_AWADDR    ), // Address Write  
280     .M_AWLEN     (M_AWLEN     ), // Transfer length  
281     .M_AWSIZE    (M_AWSIZE    ), // Transfer width  
282     .M_AWBURST   (M_AWBURST   ), // Burst type  
283     .M_AWLOCK    (M_AWLOCK    ), // Atomic access information  
284     .M_AWCACHE   (M_AWCACHE   ), // Cachable/bufferable infor  
285     .M_AWPROT    (M_AWPROT    ), // Protection info  
286     .M_AWREGION  (M_AWREGION  ),  
287     .M_AWQOS     (M_AWQOS     ),  
288     .M_AWVALID   (M_AWVALID   ), // address/control valid handshake  
289     .M_AWREADY   (M_AWREADY   ),  
290     .M_AUSER     (            ),  
291     //Write data channel  
292     .M_WID       (M_WID       ), // Write ID  
293     .M_WDATA     (M_WDATA     ), // Write Data bus  
294     .M_WSTRB     (M_WSTRB     ), // Write Data byte lane strobes  
295     .M_WLAST     (M_WLAST     ), // Last beat of a burst transfer  
296     .M_WVALID    (M_WVALID    ), // Write data valid  
297     .M_WREADY    (M_WREADY    ), // Write data ready  
298     .M_WUSER     (            ),  
299     .M_BUSER     (            ),  
300     //Write response channel  
301     .M_BID       (M_BID       ), // buffered response ID  
302     .M_BRESP     (M_BRESP     ), // Buffered write response  
303     .M_BVALID    (M_BVALID    ), // Response info valid  
304     .M_BREADY    (M_BREADY    ), // Response info ready (to slave)  
305     //Read address channel  
306     //User interface  
307     .start_dma   (ctrl_write   ),  
308     .num_trans   (num_trans    ), //Number of words transferred  
309     .start_addr  (write_addr   ),  
310     .indata      (write_data   ),  
311     .indata_req_o(indata_req_wr),  
312     .done_o      (write_done   ), //Blk transfer done  
313     .fail_check  (            ),  
314     //User signals  
315     .clk         (clk          ),  
316     .rstn        (rstn         ),  
317 );
```

Test bench (sim/yolo_engine_tb.v)

An external memory model and a memory controller

```
71 //-----
72 v axi_sram_if # ( //New
73     .MEM_ADDRW(MEM_ADDRW), .MEM_DW(MEM_DW),
74     .A(A), .I(I), .L(L), .D(D), .M(M))
75 v u_axi_ext_mem_if_input(
76     .ACLK(clk), .ARESETn(rstn),
77
78     //AXI Slave IF
79     .AWID (M_AWID ),          // Address ID
80     .AWADDR (M_AWADDR ),      // Address Write
81     .AWLEN (M_AWLEN ),        // Transfer length
82     .AWSIZE (M_AWSIZE ),      // Transfer width
83     .AWBURST (M_AWBURST ),    // Burst type
84     .AWLOCK (M_AWLOCK ),      // Atomic access information
85     .AWCACHE (M_AWCACHE ),    // Cachable/bufferable infor
86     .AWPROT (M_AWPROT ),      // Protection info
87     .AWVALID (M_AWVALID ),    // address/control valid handshake
88     .AWREADY (M_AWREADY ),
89     //Write data channel
90     .WID (M_WID ),            // Write ID
91     .WDATA (M_WDATA ),        // Write Data bus
92     .WSTRB (M_WSTRB ),        // Write Data byte lane strobes
93     .WLAST (M_WLAST ),        // Last beat of a burst transfer
94     .WVALID (M_WVALID ),      // Write data valid
95 v .WREADY (M_WREADY ),        // Write data ready
96     //Write response channel
97     .BID (M_BID ),            // buffered response ID
98     .BRESP (M_BRESP ),        // Buffered write response
99     .BVALID (M_BVALID ),      // Response info valid
100     .BREADY (M_BREADY ),      // Response info ready (from Master)
```

```
105 .ARID (M_ARID ),             // Read addr ID
106 .ARADDR (M_ARADDR ),         // Address Read
107 .ARLEN (M_ARLEN ),           // Transfer length
108 .ARSIZE (M_ARSIZE ),         // Transfer width
109 .ARBURST (M_ARBURST ),       // Burst type
110 .ARLOCK (M_ARLOCK ),         // Atomic access information
111 .ARCACHE (M_ARCACHE ),       // Cachable/bufferable infor
112 .ARPROT (M_ARPROT ),         // Protection info
113 .ARVALID (M_ARVALID ),       // address/control valid handshake
114 .ARREADY (M_ARREADY ),
115 .RID (M_RID ),               // Read ID
116 .RDATA (M_RDATA ),           // Read data bus
117 .RRESP (M_RRESP ),           // Read response
118 .RLAST (M_RLAST ),           // Last beat of a burst transfer
119 .RVALID (M_RVALID ),         // Read data valid
120 .RREADY (M_RREADY ),         // Read data ready (to Slave)
121
122 //Interface to SRAM
123 .mem_addr(mem_addr ),
124 .mem_we (mem_we ),
125 .mem_di (mem_di ),
126 .mem_do (mem_do );
127
128 // Input
129 // IMEM for SIM
130 // Inputs
131 v sram #(
132     .FILE_NAME(IFM_FILE),
133     .SIZE(2**MEM_ADDRW),
134     .WL_ADDR(MEM_ADDRW),
135     .WL_DATA(MEM_DW))
136 v u_ext_mem_input (
137     .clk (clk ),
138     .rst (rstn ),
139     .addr (mem_addr ),
140     .wdata (mem_di ),
141     .rdata (mem_do ),
142     .ena (1'b0 )           // Read only
143 );
```

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

```
144 //-----
145 // CNN Accelerator
146 //-----
147 reg [31:0] i_0;
148 reg [31:0] i_1;
149 reg [31:0] i_2;
150
151 yolo_engine #(
152     .AXI_WIDTH_AD(A),
153     .AXI_WIDTH_ID(4),
154     .AXI_WIDTH_DA(D),
155     .AXI_WIDTH_DS(M),
156     .MEM_BASE_ADDR(2048),
157     .MEM_DATA_BASE_ADDR(2048)
158 )
159 u_yolo_engine
160 (
161     .clk(clk),
162     .rstn(rstn),
163
164     .i_ctrl_reg0(i_0), // network_start // {debug_big(1),
165     .i_ctrl_reg1(i_1), // Read_address (INPUT)
166     .i_ctrl_reg2(i_2), // Write_address
167     .i_ctrl_reg3(32'd0), // Reserved
```

```
190     .M_AWVALID (M_AWVALID),
191     .M_AWREADY (M_AWREADY),
192     .M_AWADDR (M_AWADDR ),
193     .M_AWID (M_AWID ),
194     .M_AWLEN (M_AWLEN ),
195     .M_AWSIZE (M_AWSIZE ),
196     .M_AWBURST (M_AWBURST),
197     .M_AWLOCK (M_AWLOCK ),
198     .M_AWCACHE (M_AWCACHE),
199     .M_AWPROT (M_AWPROT ),
200     .M_AWQOS ( ),
201     .M_AWREGION( ),
202     .M_AWUSER ( ),
203
204     .M_WVALID (M_WVALID ),
205     .M_WREADY (M_WREADY ),
206     .M_WDATA (M_WDATA ),
207     .M_WSTRB (M_WSTRB ),
208     .M_WLAST (M_WLAST ),
209     .M_WID (M_WID ),
210     .M_WUSER ( ),
211
212     .M_BVALID (M_BVALID ),
213     .M_BREADY (M_BREADY ),
214     .M_BRESP (M_BRESP ),
215     .M_BID (M_BID ),
216     .M_BUSER ( ),
217
218     .network_done(network_done),
219     .network_done_led(network_done_led)
220 );
```

Test bench (sim/yolo_engine_tb.v)

- We can control and monitor the CNN accelerator

- ✓ Start: $i_0 \leftarrow 1$

- ✓ Send other configurations as you want

- ✓ i_1, i_2

- ✓ Preload:

- ✓ You may test a particular layer

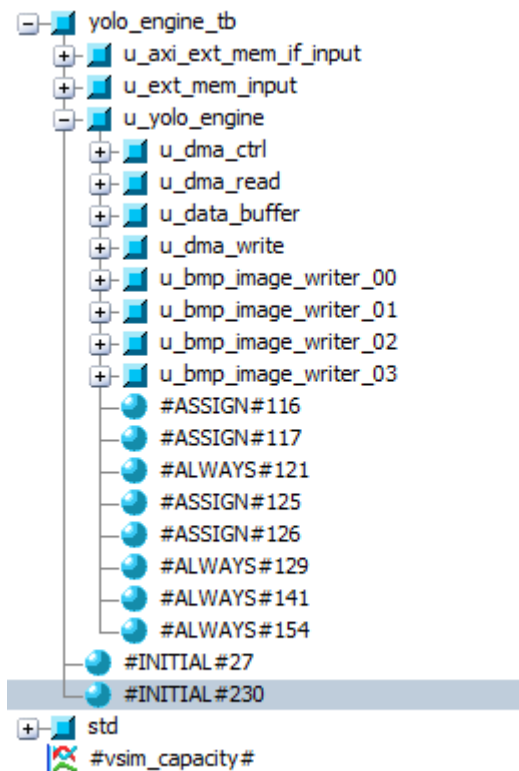
- ✓ For example, compare the results from HW with that from the software.

- ✓ 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
```

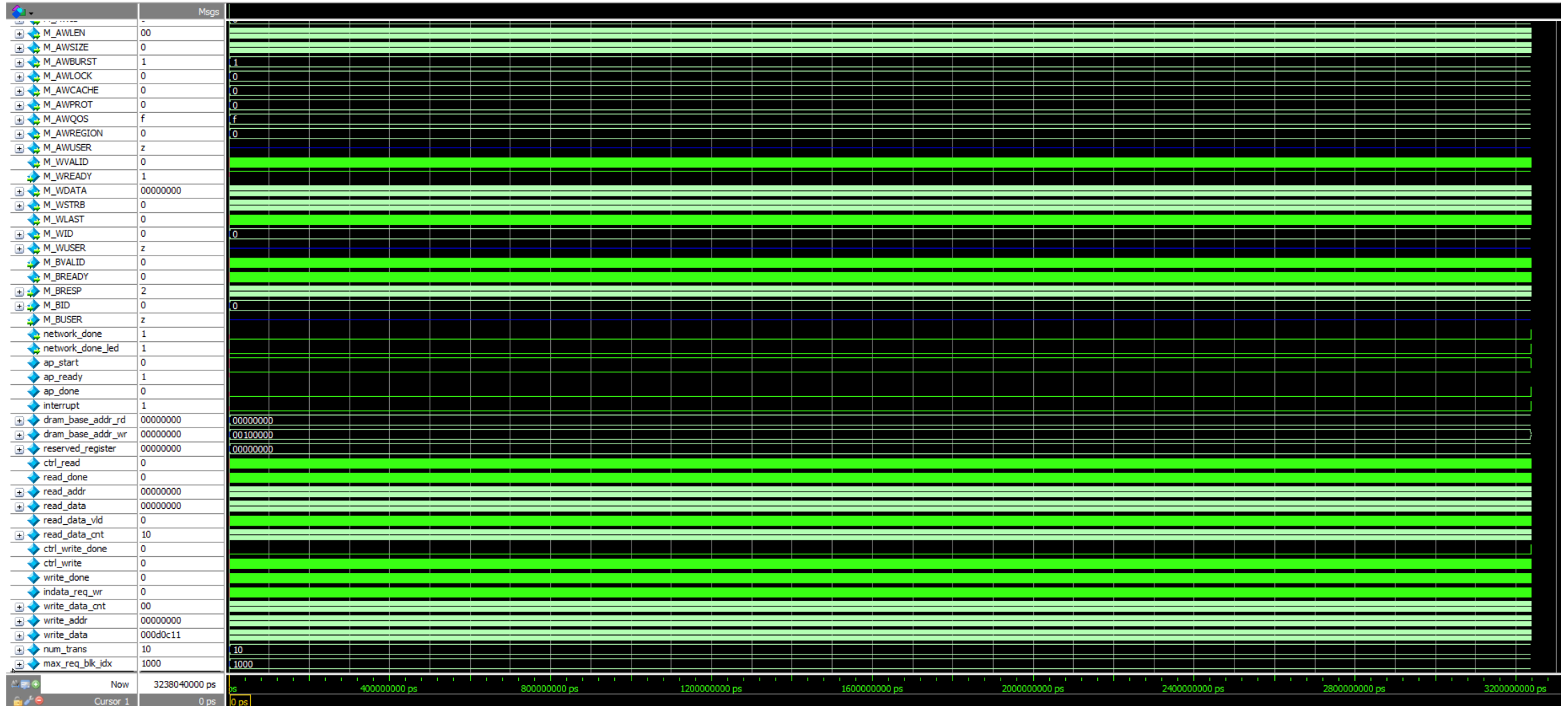
Test bench (sim/yolo_engine_tb.v)

- 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)

