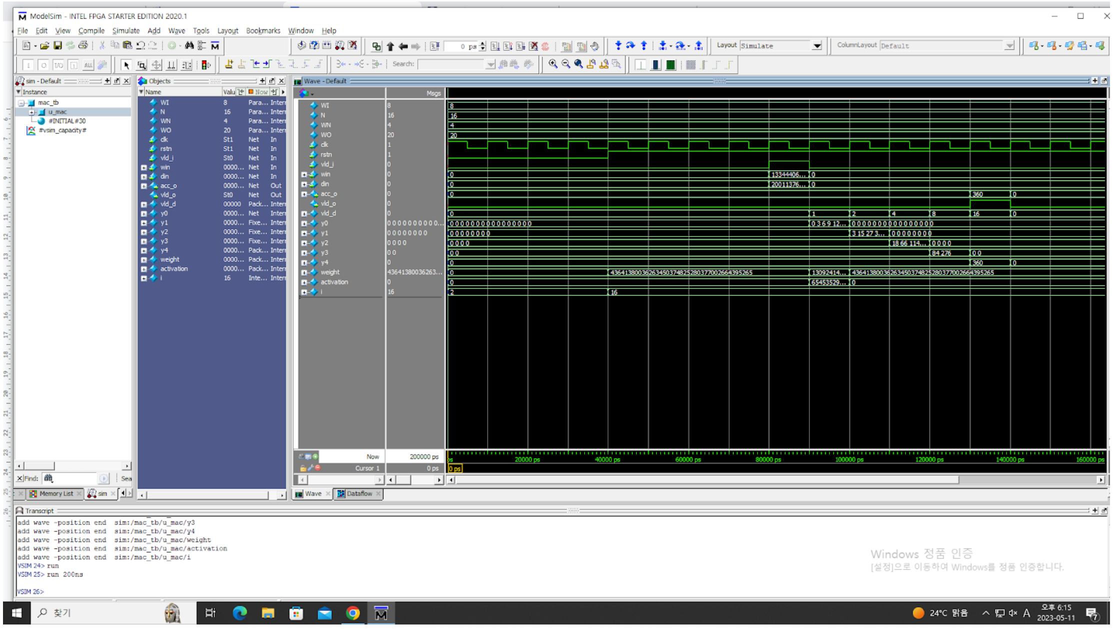
Homework 9: MAC kernel, Convolutional kernels

Name: Satyam (2023-81784)

Problem 1 (10p): MAC (Channel-wise accumulation)

a. Completed the missing codes in mac.v.
b. Do a simulation and capture the waveform.



c. Explain why the final result is 360.

The final result is the summation of weights with activations. Where weights are 3 and activations range from 0 to 15. Therefore,

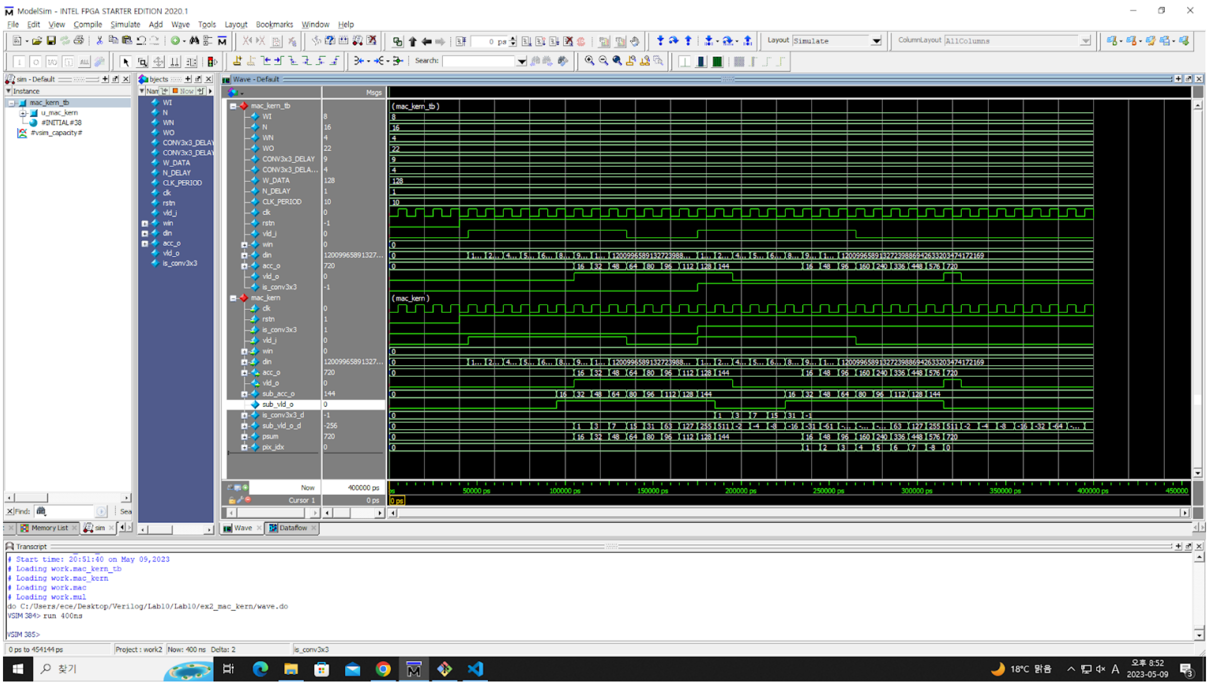
Summation of all (weight*activation) = 3*0 + 3*1 … + 3*14 + 3*15 = 360

d. Explain why the output valid signal (vld_o) delays 5 cycles after the input valid signal (vld_i).

vld_i is equal to 5 cycles of vld_o because first validation is multiplication of weights and activations (y0) then 4 loops are for additions (y1, y2, y3, y4). 4 loops are for additions because pairing and adding16 elements requires 4 loops (log2(16)).

Problem 2 (10p): MAC kernel (Filter-wise accumulation)

a. Reused the implemented mac.v in Problem 1.
b. Completed the missing codes in mac_kern.v.
c. Waveform.



d. Explain the values of the output port acc_o.

“acc_o” is the output of multiplication and accumulation of summation of weights and activations. Actually, “acc_o” depends on the conv1x1 or conv3x3. If conv1x1 then “acc_o” is summation from 0 to 15 but if conv3x3 then “acc_o” is summation from 0 to 143.

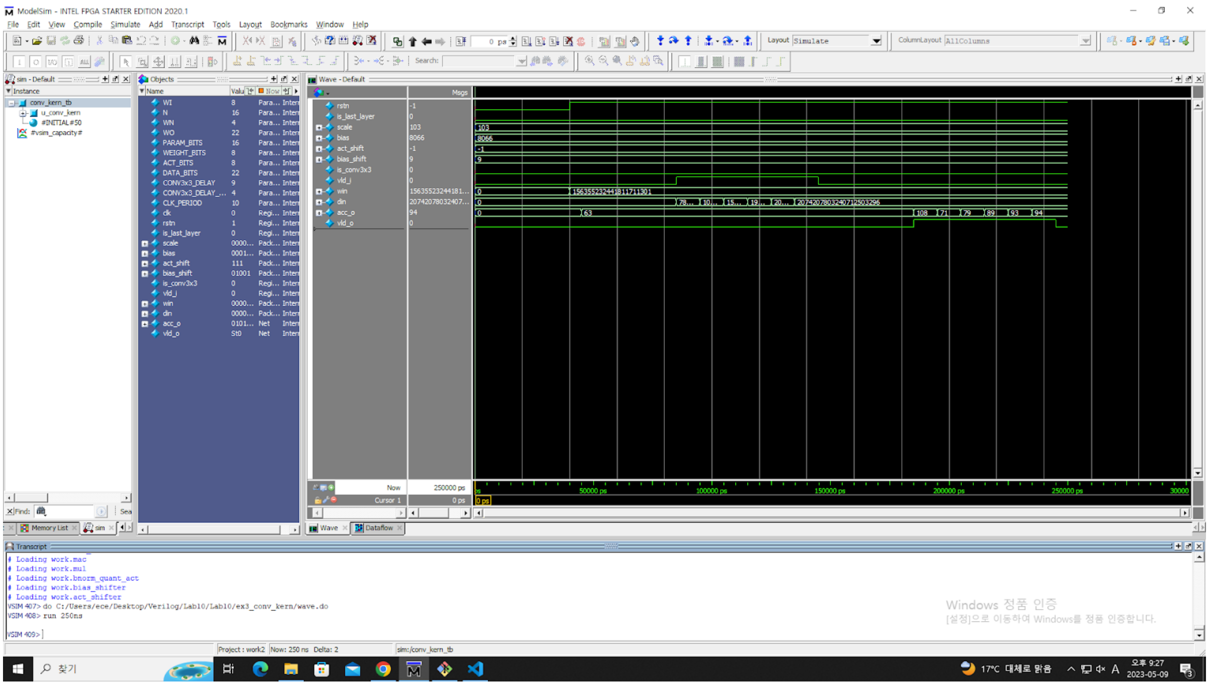
e. According to the waveform, the time intervals between vld_i and vld_o of mac_kern are 6 and 14 cycles for conv1x1 and conv3x3, respectively. Explain those numbers.

For conv1x1, vld_i outputs to sub_vld_o (sub-validation) for 5 cycles (1 for multiplication and 4 for addition) and then 1 more cycle for outputting sub_vld_o to vld_o, so 6 cycles total.

For conv3x3, same 6 cycles as above (for first pixel) but additional 8 cycles for summing rest of the 8 pixels. So, 6+8 = 14 cycles in total.

Problem 3 (10p): Convolutional Kernel

a. Reused the implemented mac.v and mac_kern.v in Problems 1 and 2.
b. Completed the missing codes in bias_shifter.v, act_shifter.v and conv_kern_tb.v.
c. Waveform.



Problem 4: (Optional) Sliding window (2p)

Let H and W be the input image's height and weight, respectively. In our case, H = 128 and W = 128. Assume an input image is stored in an array in_img[H][W] whose element has 8 bits. We aim to compute an output feature map stored in an array out_img[H][W]. Like Problem 3, let's assume that weights (win) are pre-defined. Your task is to complete the following pseudocode:

```
for h = 0 to H-1
    for w = 0 to W-1
        // Generate din from in_img
        //{{{
        //corner of the images
        if (h == 0 and w ==0){din[0] = 0; din[1] = 0; din[2] = 0; din[3] = 0; din[4] = in_img[h][w]; din[5] = in_img[h][w+1]; din[6] = 0; din[7] =  in_img[h+1][w]; din[8] = in_img[h+1][w+1];}
        elif (h == 0 and w == W-1){din[0] = 0; din[1] = 0; din[2] = 0; din[3] = in_img[h][w-1]; din[4] = in_img[h][w]; din[5] = 0; din[6] = in_img[h+1][w-1]; din[7] = in_img[h+1][w]; din[8] = 0;}
        elif (h == H-1 and w == 0){din[0] = 0; din[1] = in_img[h-1][w]; din[2] = in_img[h-1][w+1]; din[3] = 0; din[4] = in_img[h][w]; din[5] = in_img[h][w+1]; din[6] = 0; din[7] = 0; din[8] = 0;}
        elif (h == H-1 and w ==W-1){din[0] = in_img[h-1][w-1]; din[1] = in_img[h-1][w]; din[2] = 0; din[3] = in_img[h][w-1]; din[4] = in_img[h][w]; din[5] = 0; din[6] = 0; din[7] = 0; din[8] = 0;}
        //borders
        elif (h == 0 and 0<w <W-1){din[0] = 0; din[1] = 0; din[2] = 0; din[3] = in_img[h][w-1]; din[4] = in_img[h][w]; din[5] = in_img[h][w+1]; din[6] = in_img[h+1][w-1]; din[7] = in_img[h+1][w]; din[8] = in_img[h+1][w+1];}
        elif (0<h <H-1 and w == 0){din[0] = 0; din[1] = in_img[h-1][w]; din[2] = in_img[h-1][w+1]; din[3] = 0; din[4] = in_img[h][w]; din[5] = in_img[h][w+1]; din[6] = 0; din[7] = in_img[h+1][w]; din[8] = in_img[h+1][w+1];}
        elif (h==W-1 and 0<w <W-1){din[0] = in_img[h-1][w-1]; din[1] = in_img[h-1][w]; din[2] = in_img[h-1][w+1]; din[3] = in_img[h][w-1]; din[4] = in_img[h][w]; din[5] = in_img[h][w+1]; din[6] = 0; din[7] = 0; din[8] = 0;}
        elif (0<h <H-1 and w==W-1){din[0] = in_img[h-1][w-1]; din[1] = in_img[h-1][w]; din[2] = 0; din[3] = in_img[h][w-1]; din[4] = in_img[h][w]; din[5] = 0; din[6] = in_img[h+1][w-1]; din[7] = in_img[h+1][w]; din[8] = 0;}
        //other pixels
        elif (0<h<H-1 and 0<w<W-1){din[0] = in_img[h-1][w-1]; din[1] = in_img[h-1][w]; din[2] = in_img[h-1][w+1]; din[3] = in_img[h][w-1]; din[4] = in_img[h][w]; din[5] = in_img[h][w+1]; din[6] = in_img[h+1][w-1]; din[7] = in_img[h+1][w]; din[8] = in_img[h+1][w+1];}
        //}}}
        // Compute an output pixel which corresponds to acc_o of conv_kern in Problem 3
        out_img[h][w] = conv_kern(win,din)
    end for
end for
```