[SoC Design – Term Project]

Accelerator for CNN

Chester Sungchung Park

SoC Design Lab, Konkuk University

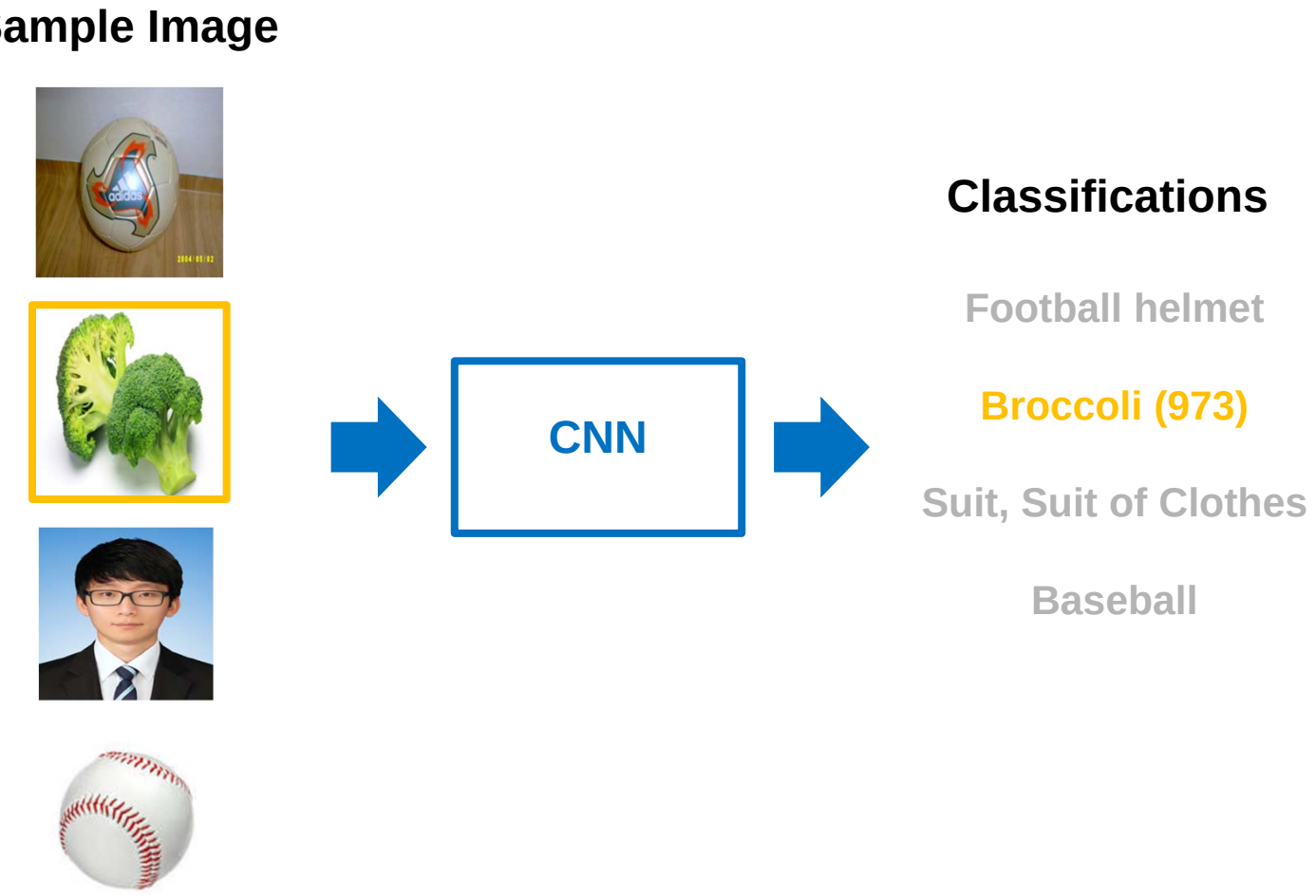
Webpage: http://soclab.konkuk.ac.kr

Outline

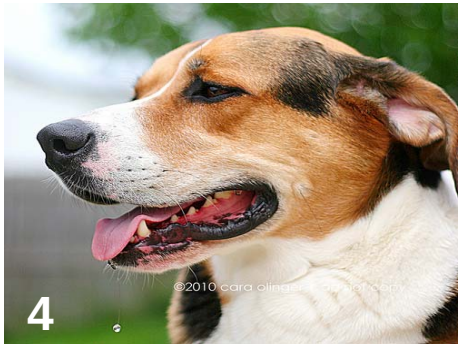
- Hardware accelerator
- Convolution in CNN
  - AlexNet for ImageNet
- Design flow
- Design constraints
- Evaluation
- Submission
- Presentation
- Appendix

Convolutional Neural Networks (CNN)

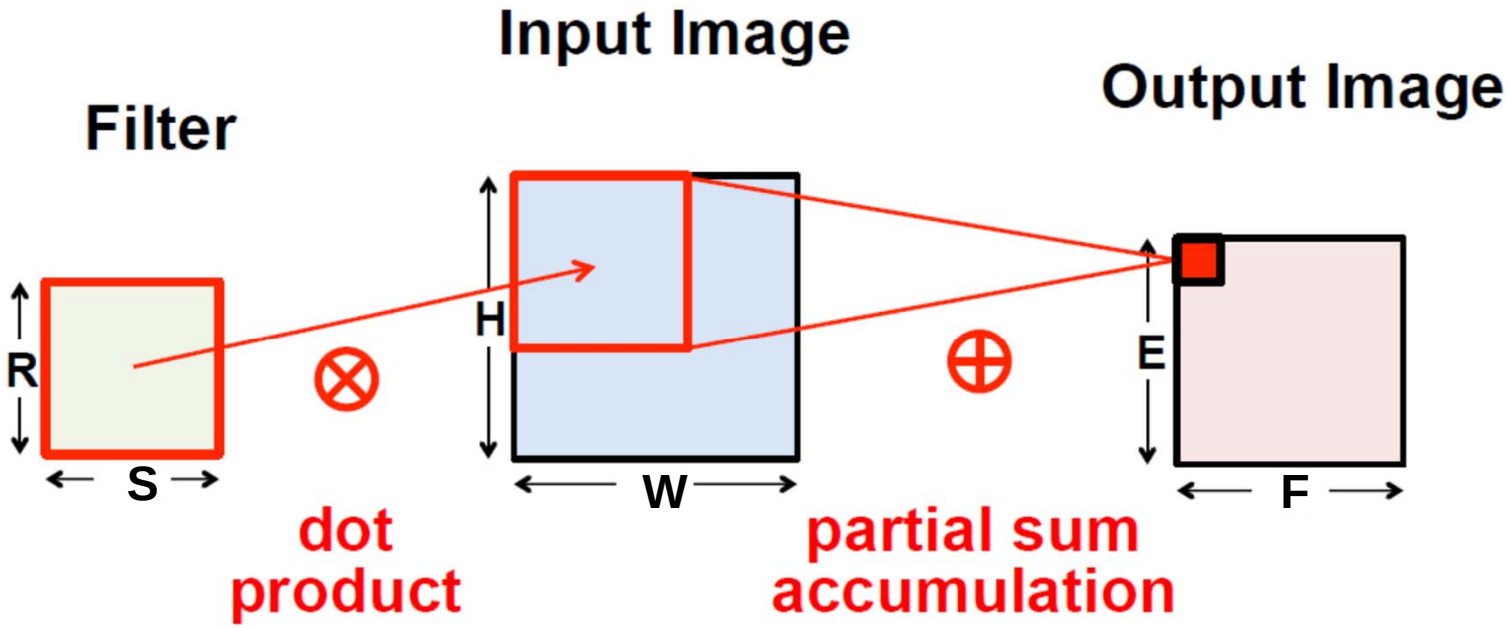
- Example: AlexNet for ImageNet



ImageNet (Input Images)

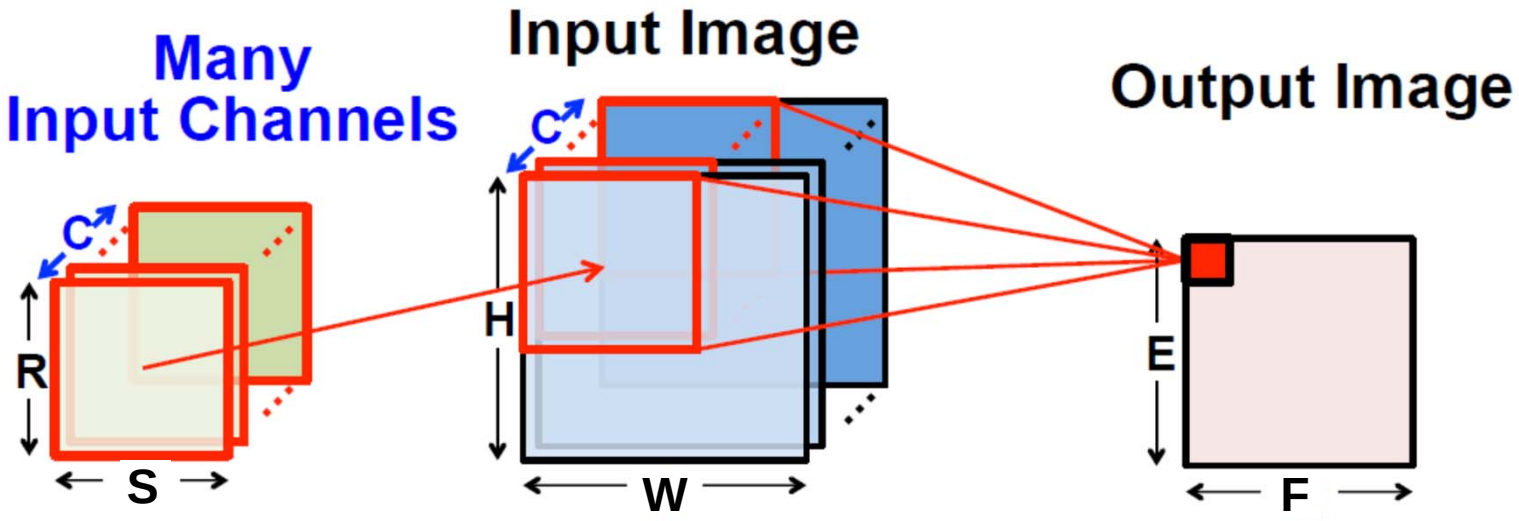


Convolution in CNN



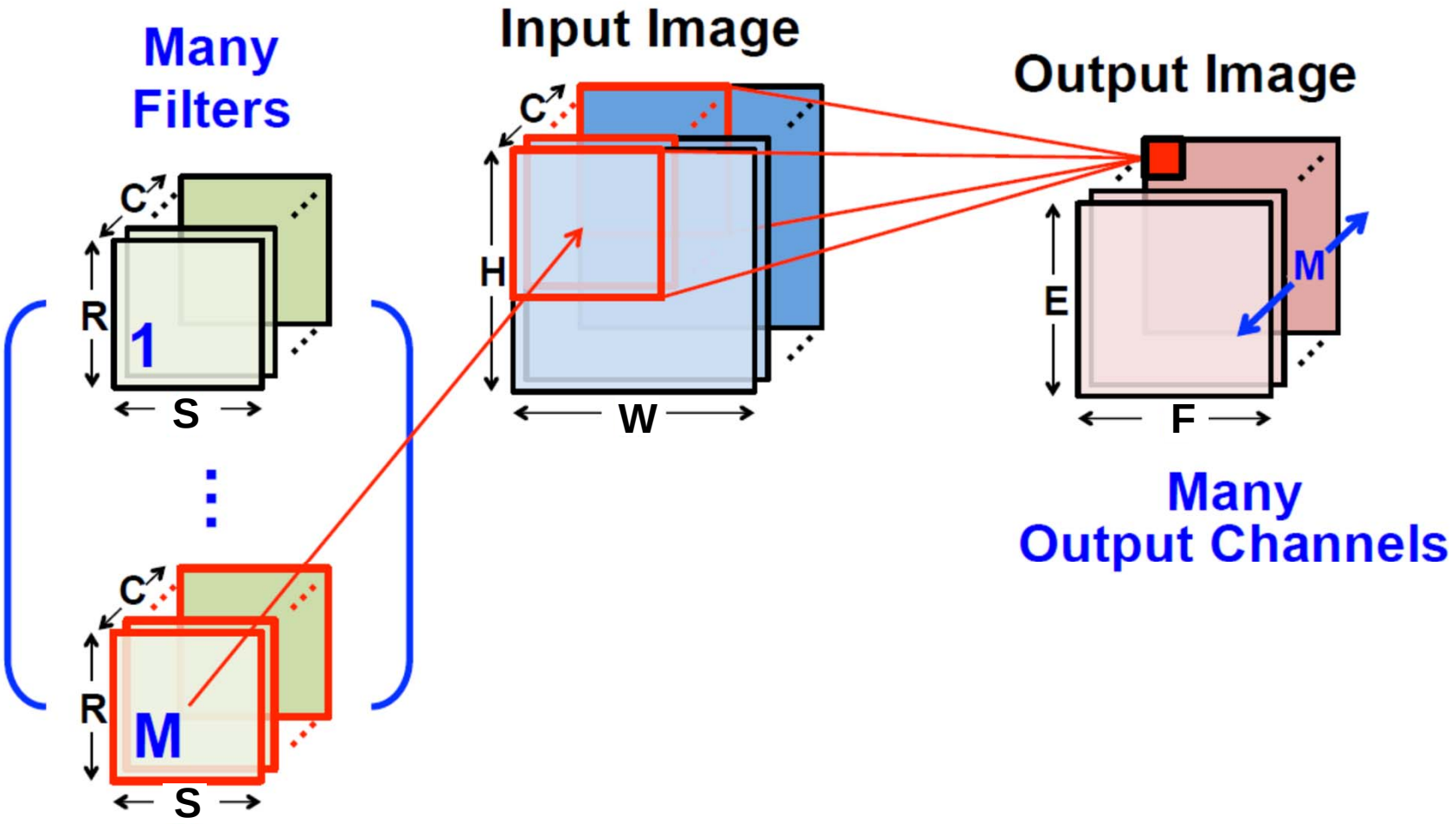
Source: Y-H. Chen

Convolution in CNN



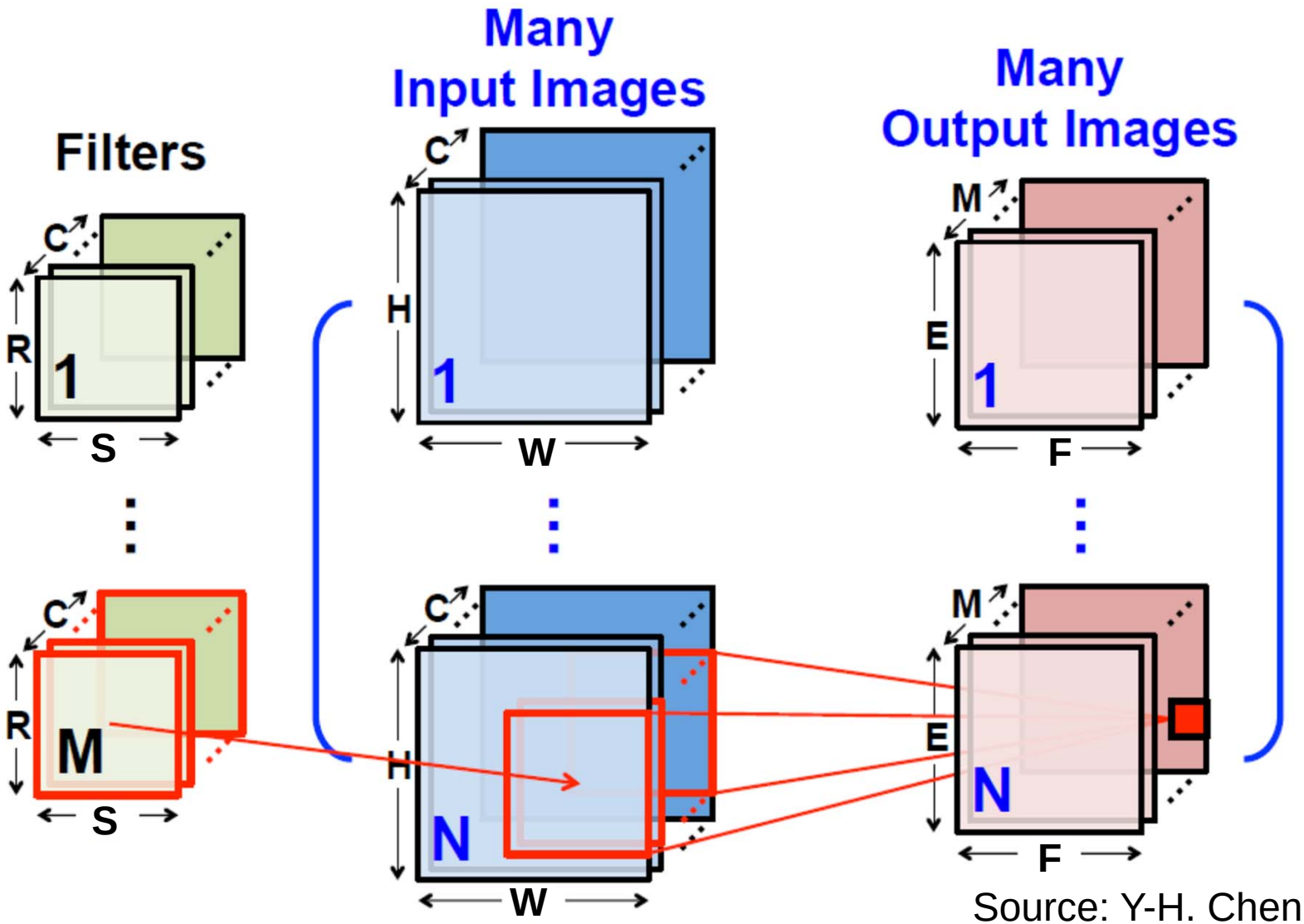
Source: Y-H. Chen

Convolution in CNN



Source: Y-H. Chen

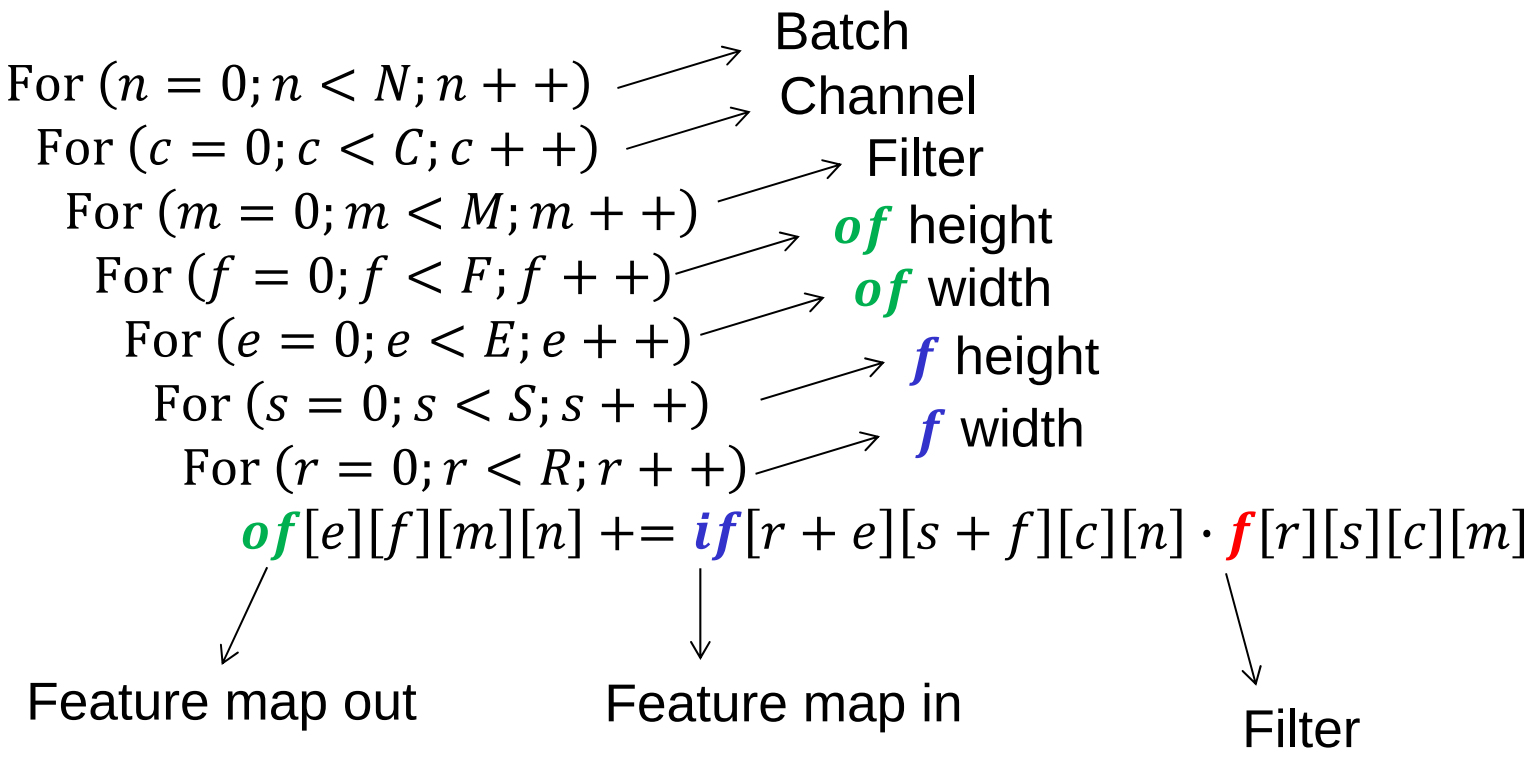
Convolution in CNN



Source: Y-H. Chen

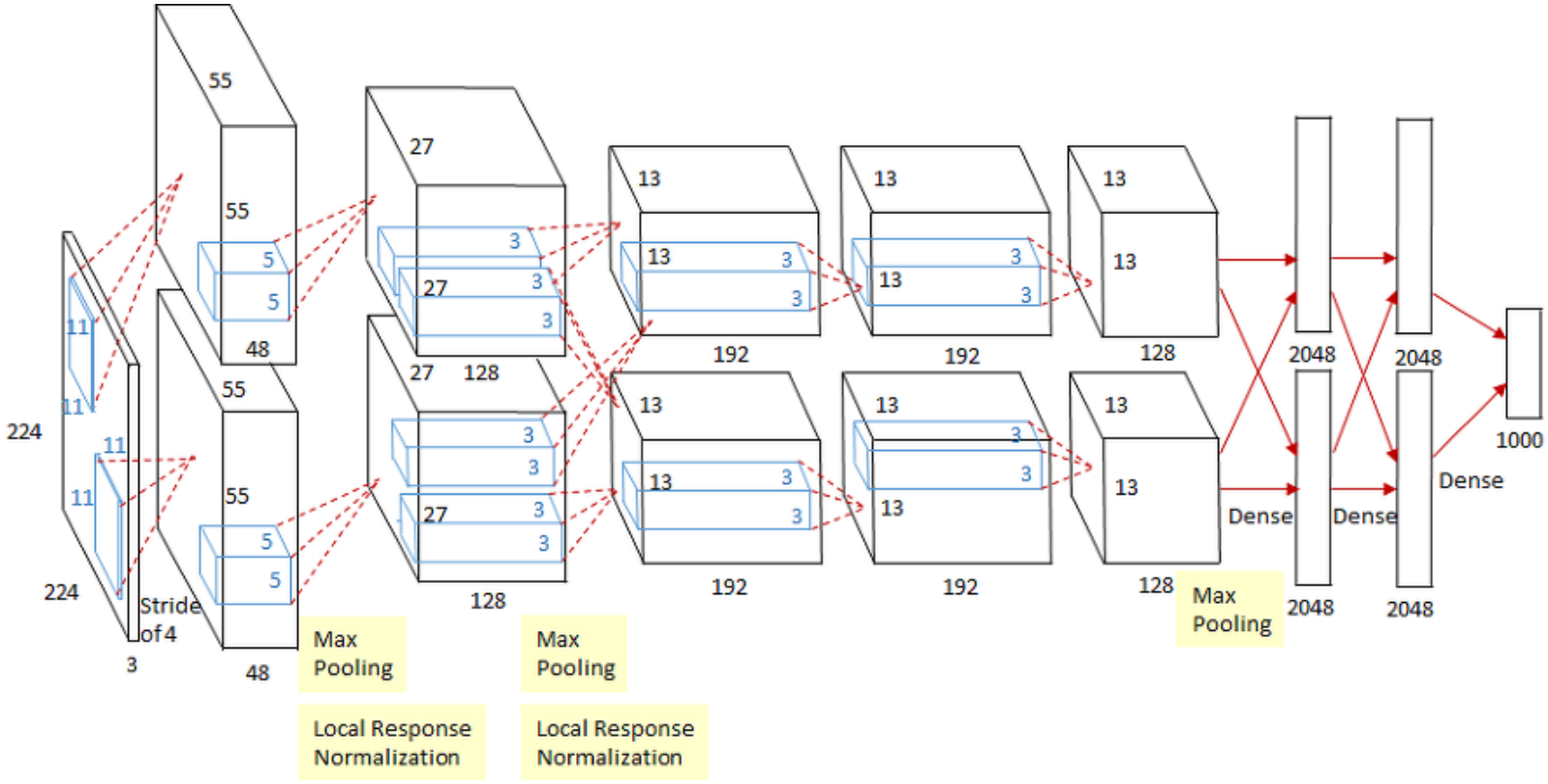
Convolution in CNN

- Pseudo code



Convolution in CNN

- AlexNet for ImageNet



Convolution in CNN

- AlexNet for ImageNet (cont'd)

	Feature map in		Filter		Channel	
	Width (W)	Height (H)	Width (S)	Height (R)	# I-Ch (C)	# O-Ch (M)
Layer 1	227	227	11	11	3	96
Layer 2	31	31	5	5	48	256
Layer 3	15	15	3	3	256	384
Layer 4	15	15	3	3	192	384
Layer 5	15	15	3	3	192	256
Layer 6	6	6	6	6	256	4096
Layer 7	1	1	1	1	4096	4096
Layer 8	1	1	1	1	4096	1000

Reference Implementation

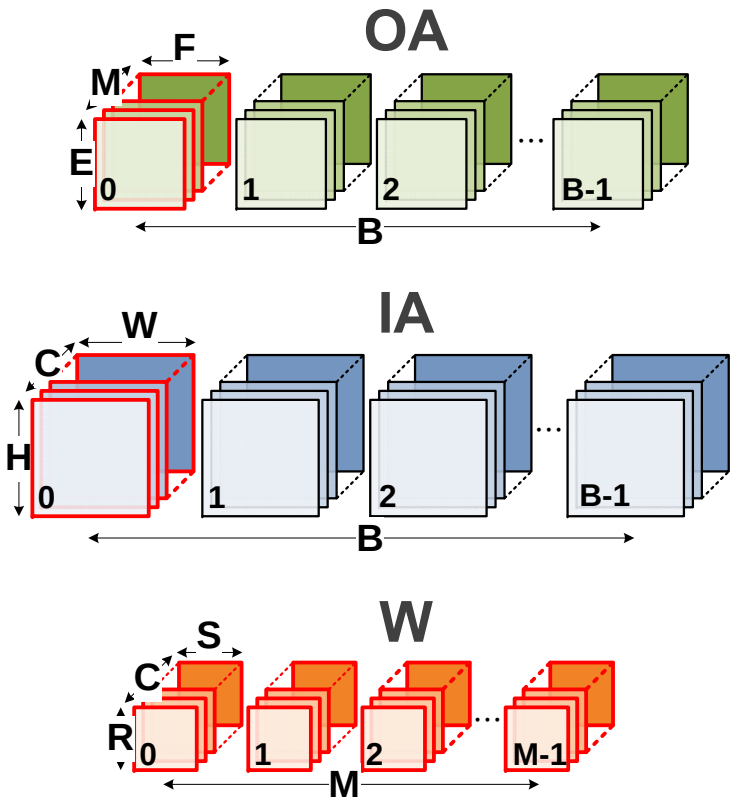
- Run the attached reference implementation as explained in the appendix
  - AlexNet with layer 3 in HW and the others in SW
  - No local reuse (NLR) chosen as dataflow

TP_SD2019_src.zip

- For details of NLR, refer to the following paper
  - "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural", C. Zhang et al., FPGA 2015

No Local Reuse (NLR)

- Original algorithm

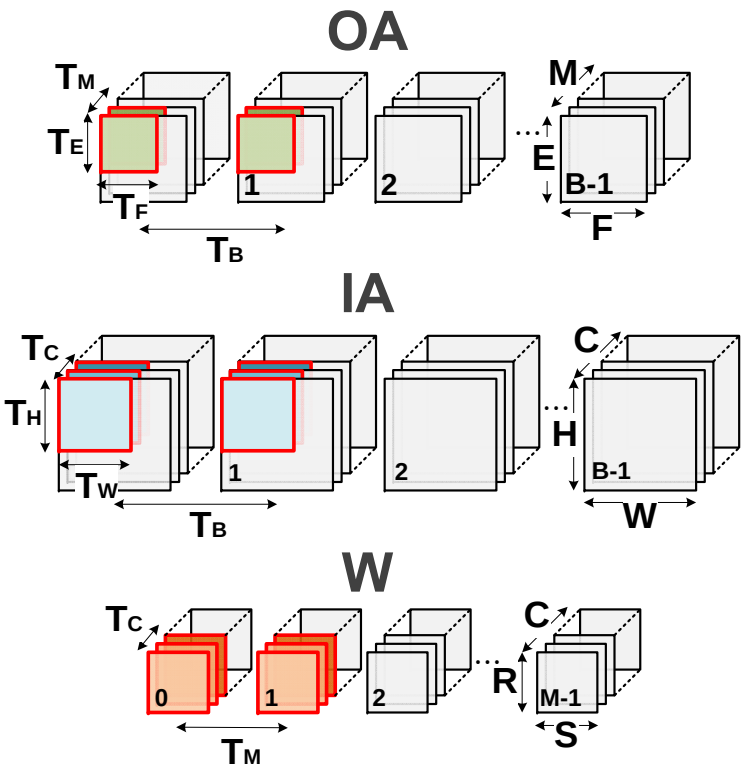



```
for (b = 0; b < B; b++) {
 for (e = 0; e < E; e++) {
  for (f = 0; f < F; f++) {
   for (m = 0; m < M; m++) {
    for (c = 0; c < C; c++) {
     for (r = 0; r < R; r++) {
      for (s = 0; s < S; S++) {
        O[b][m][e][f]+=
        W[m][c][r][s]*I[b][c][e+r][f+s];
} } } } } } } }
```

B: No.of images
E: Output height
F: Output width
M: No.of filters
C: No.of features
R: Filter height
S: Filter width

No Local Reuse (NLR)

- Loop tiling for NLR

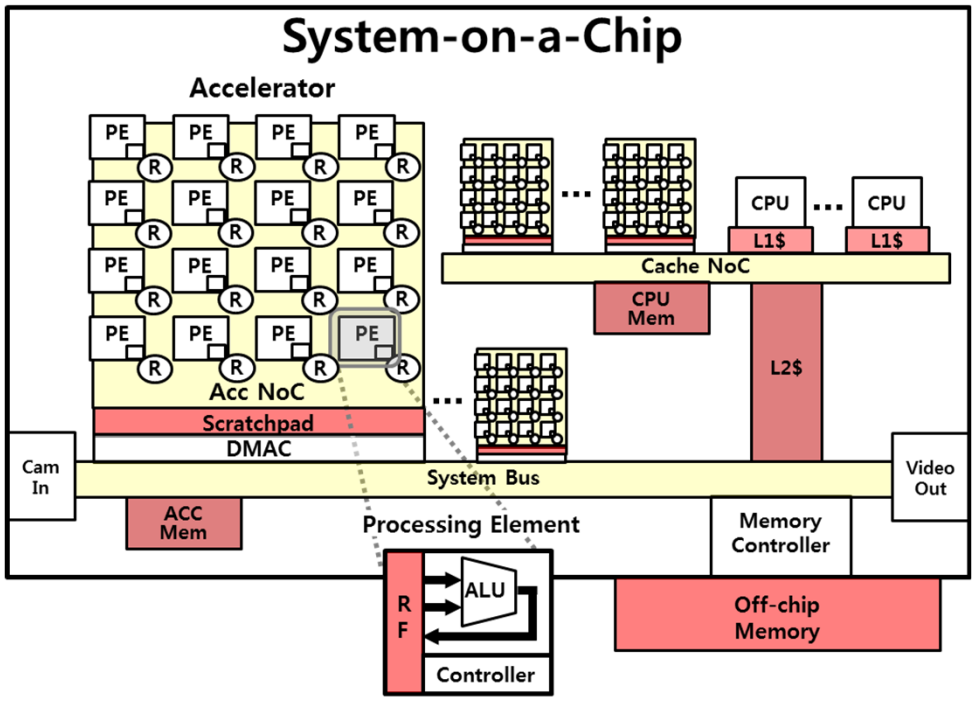


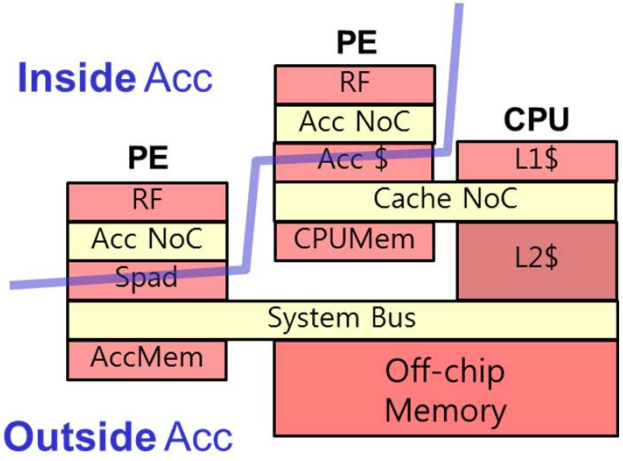

```
for (b = 0; b < B; b += TB) {
 for (e = 0; e < E; e += TE) {
  for (f = 0; f < F; f += TF) {
   for (m = 0; m < M; m += TM) {
    for (c = 0; c < C; c += TC) {
     Ibuf = I[b:b+TB-1][c:c+TC-1][e:e+TE+R-2][f:f+TF+S-2];
     Wbuf = W[m:m+TM-1][c:c+TC-1][0:R-1][0:S-1];
     for (r = 0; r < R; r++) {
      for (s = 0; s < S; s++) {
       for (tb = 0; tb+b < min(B,b+TB); tb++) {
        for (te = 0; te+e < min(E,e+TE); te++) {
         for (tf = 0; tf+f < min(F,f+TF); tf++) {
          for (tm = 0; tm < TM; tm++) {
           for (tc = 0; tc < TC; tc++) {
            wx[tm][tc] = Wbuf[tm][tc][r][s];
            ix[tc] = Ibuf[tb][tc][te+r][tf+s];
            ox[tm] = Obuf[tb][tm][te][tf];
            Obuf[tb][tm][te][tf] = wx[tm][tc]*ix[tc]+ox[tm];
    }}}}}}}}
    O[b:b+TB-1][m:m+TM-1][e:e+TE-1][f:f+TF-1] = Obuf;
}}}}
```

TB: Batch size
TE: Output height /tile
TF: Output width /tile
TM: No.of filters /tile
C: No.of features /tile
R: Filter height /tile
S: Filter width /tile

No Local Reuse (NLR)

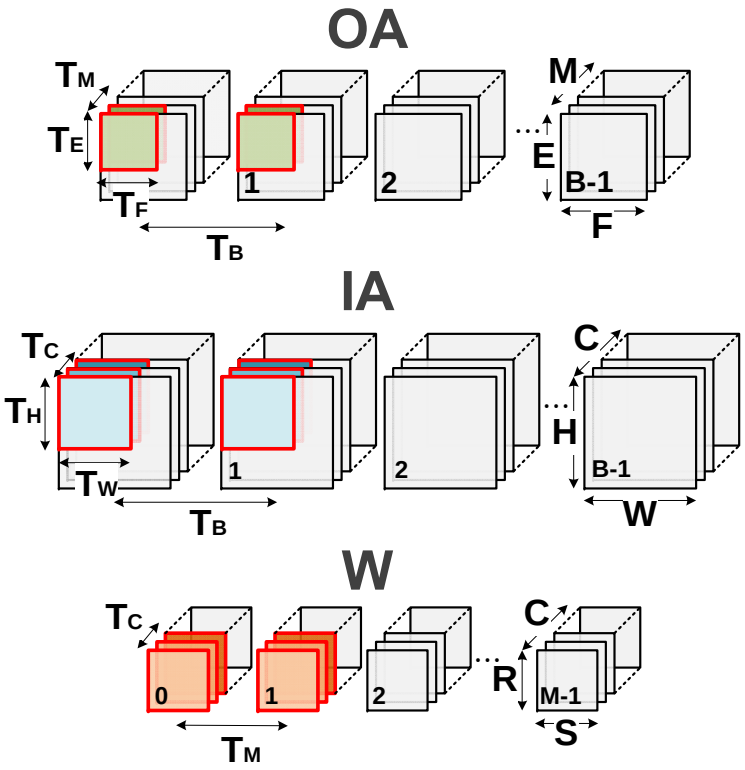
- Memory hierarchy

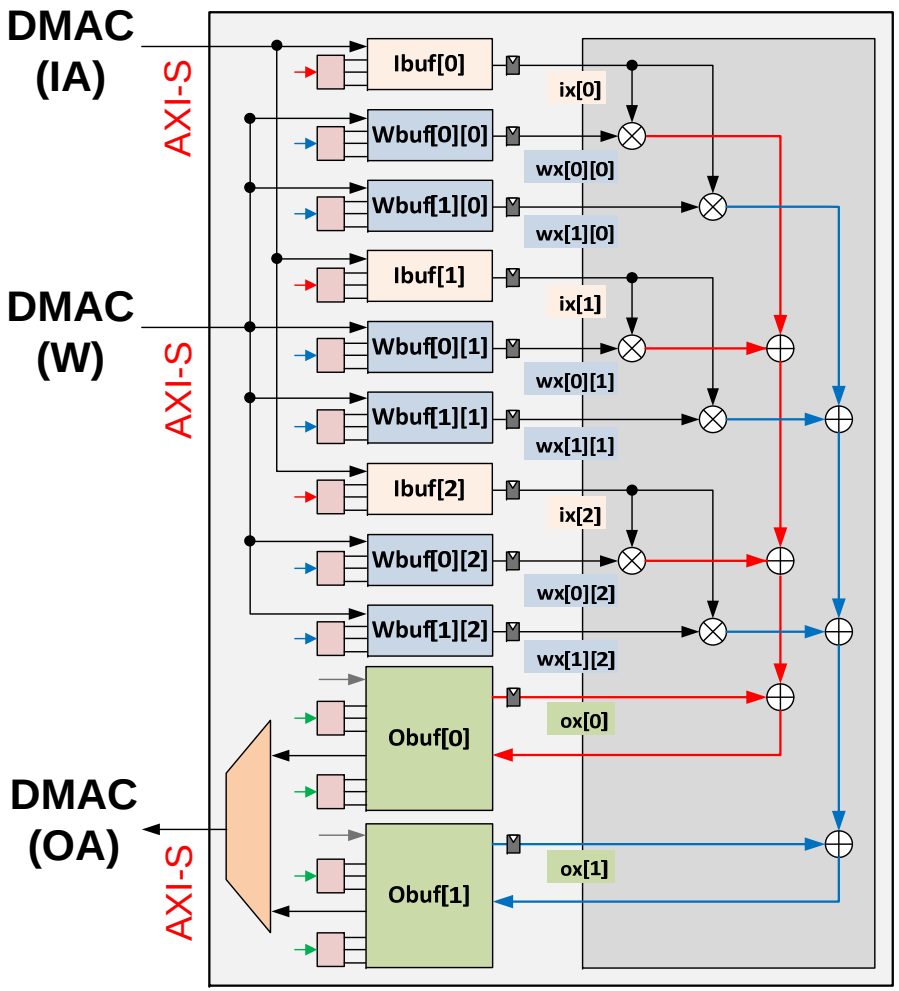




No Local Reuse (NLR)

- Accelerator hardware



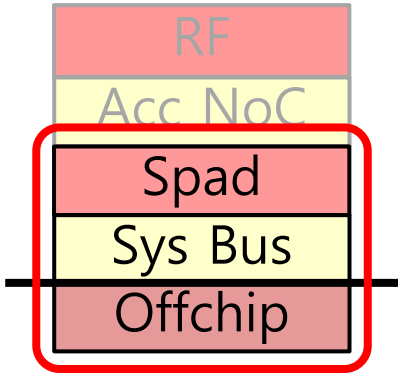


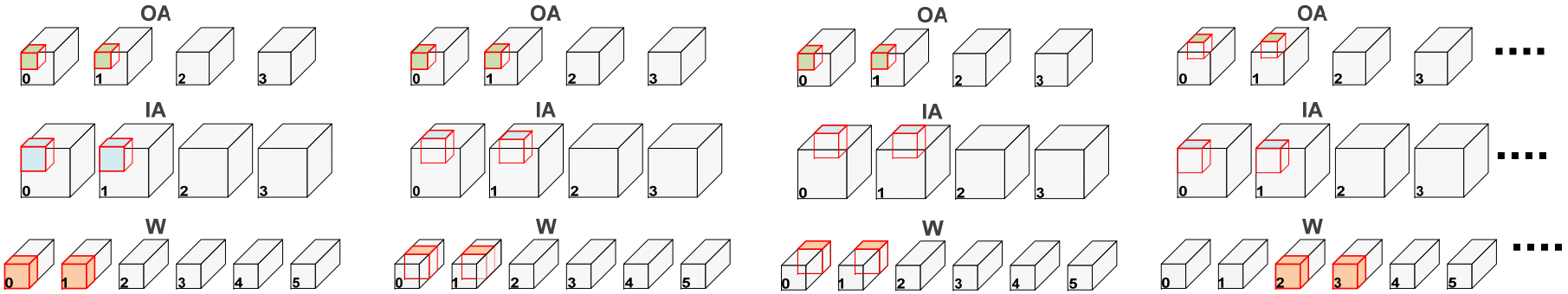
* $T_C = 2$, $T_M = 3$

No Local Reuse (NLR)

```
for (b = 0; b < B; b += TB) {
 for (e = 0; e < E; e += TE) {
  for (f = 0; f < F; f += TF) {
   for (m = 0; m < M; m += TM) {
    for (c = 0; c < C; c += TC) {
     Ibuf = I[b:b+TB-1][c:c+TC-1][e:e+TE+R-2][f:f+TF+S-2];
     Wbuf = W[m:m+TM-1][c:c+TC-1][0:R-1][0:S-1];
     ..............................................

    }}}}}}}
    O[b:b+TB-1][m:m+TM-1][e:e+TE-1][f:f+TF-1] = Obuf;
}}}}
```

Outer-Pipeline
(**Inter**-Pass)



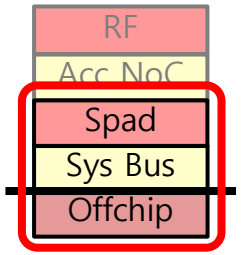


* C = 6, M = 6, T_C = 2, T_M = 3

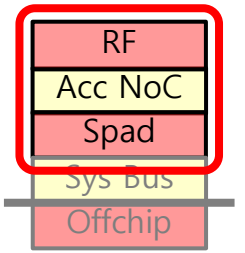
No Local Reuse (NLR)

```
for (b = 0; b < B; b += TB) {
 for (e = 0; e < E; e += TE) {
  for (f = 0; f < F; f += TF) {
   for (m = 0; m < M; m += TM) {
    for (c = 0; c < C; c += TC) {
     Ibuf = I[b:b+TB-1][c:c+TC-1][e:e+TE+R-2][f:f+TF+S-2];
     Wbuf = W[m:m+TM-1][c:c+TC-1][0:R-1][0:S-1];
     for (r = 0; r < R; r++) {
      for (s = 0; s < S; s++) {
       for (tb = 0; tb+b < min(B,b+TB); tb++) {
        for (te = 0; te+e < min(E,e+TE); te++) {
         for (tf = 0; tf+f < min(F,f+TF); tf++) {
          for (tm = 0; tm < TM; tm++) {
           for (tc = 0; tc < TC; tc++) {
            wx[tm][tc] = Wbuf[tm][tc][r][s];
            ix[tc] = Ibuf[tb][tc][te+r][tf+s];
            ox[tm] = Obuf[tb][tm][te][tf];
            Obuf[tb][tm][te][tf] = wx[tm][tc]*ix[tc]+ox[tm];
    }}}}}}}
    O[b:b+TB-1][m:m+TM-1][e:e+TE-1][f:f+TF-1] = Obuf;
}}}}
```

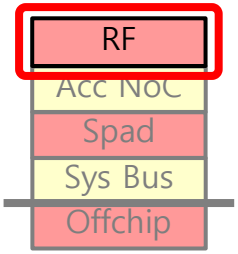
Outer-Pipeline (**Inter**-Pass)



Inner-Pipeline (**Intra**-Pass)

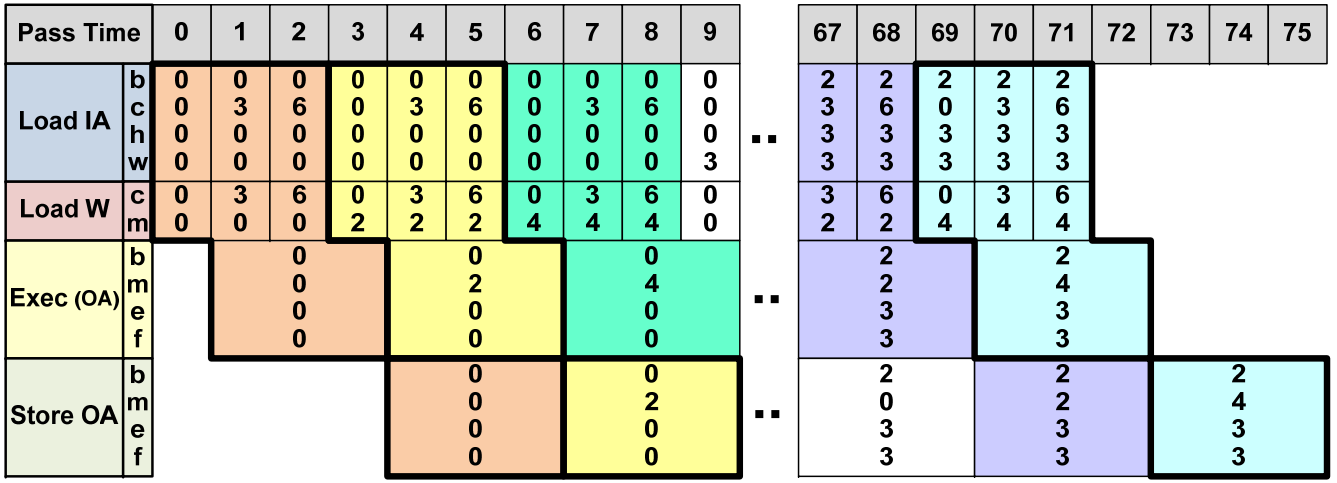


Unroll



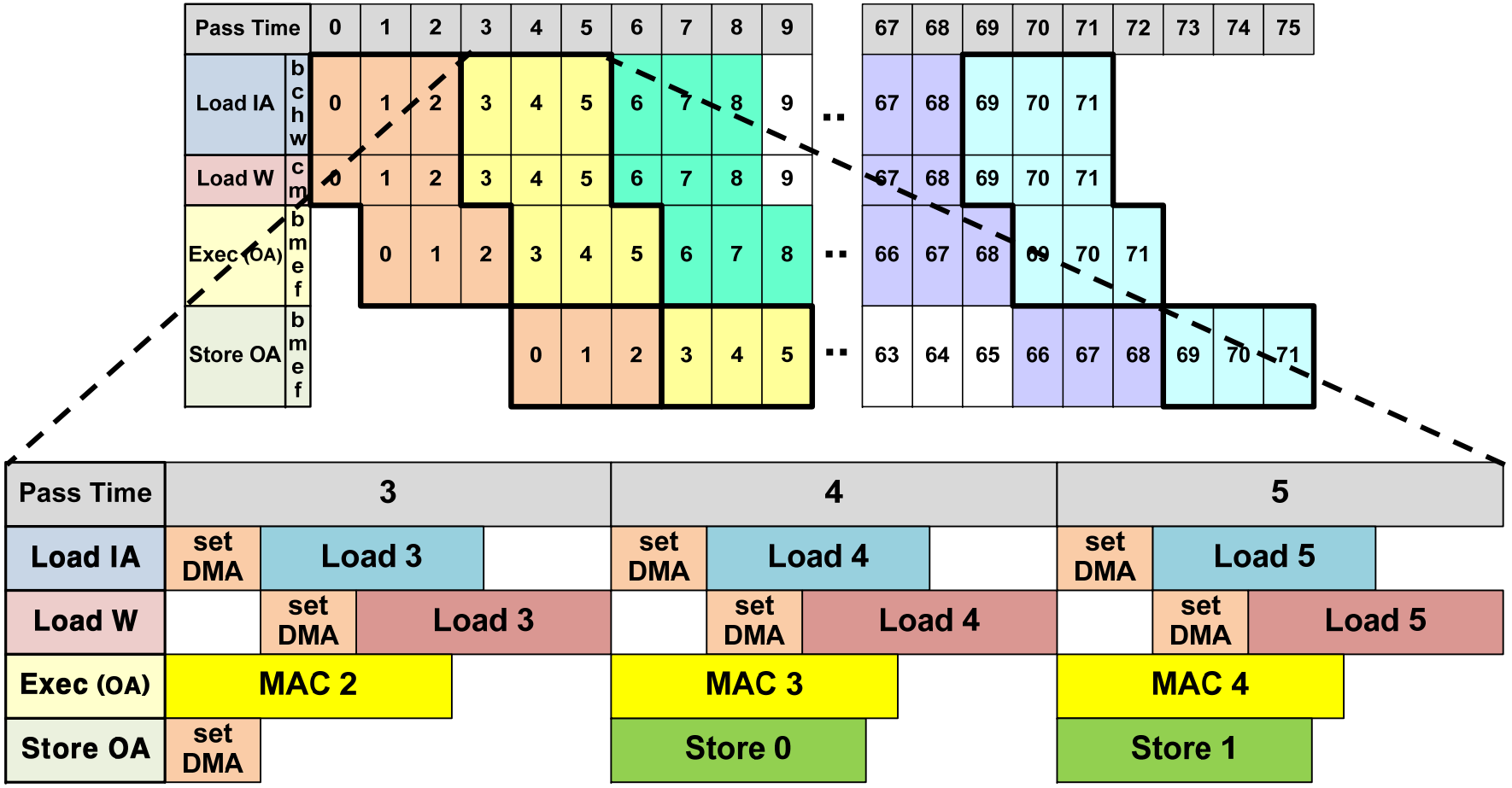
No Local Reuse (NLR)

- Outer pipeline (inter-pass)



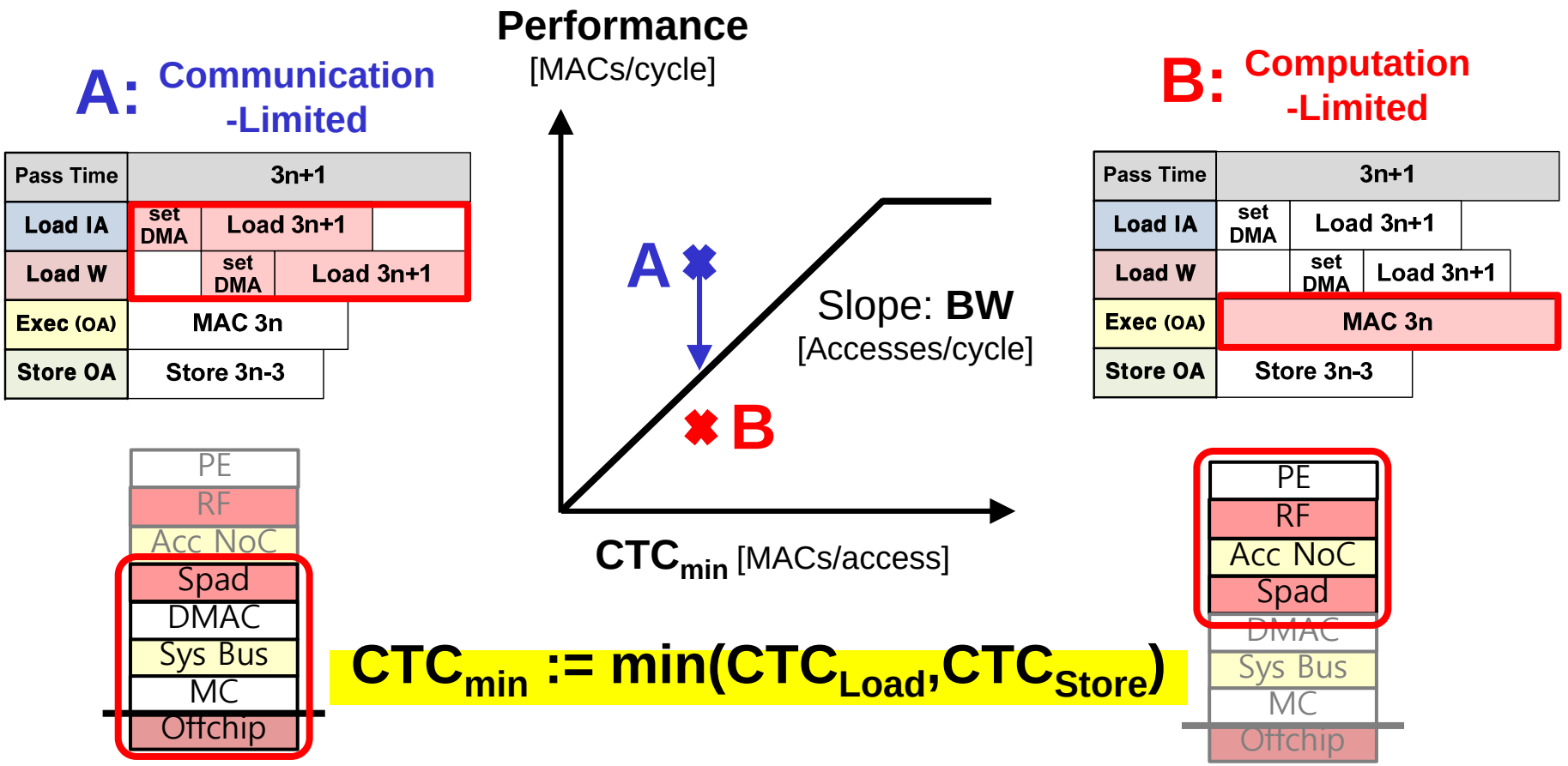
No Local Reuse (NLR)

- Outer pipeline (inter-pass) (cont'd)

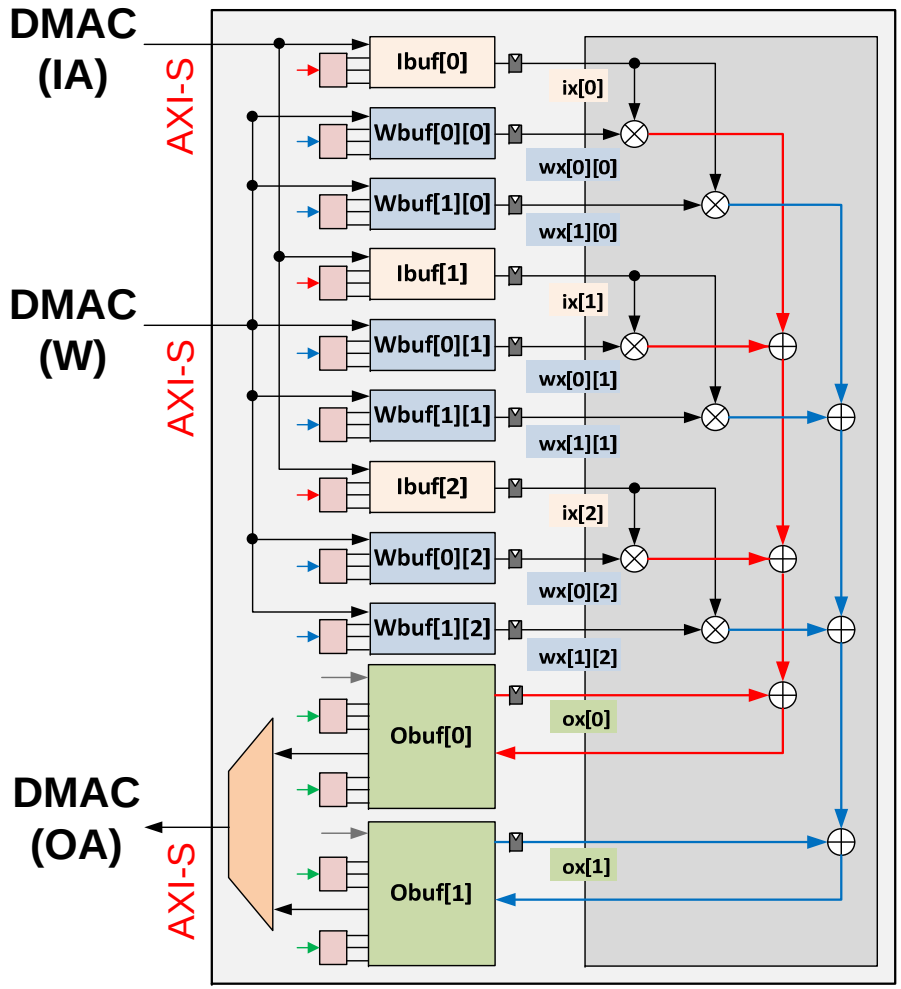


No Local Reuse (NLR)

- Roofline model (revisited)



Available Hardware Blocks



Denoted as
“HLS2x3_i” for layer i

* $T_C = 2$, $T_M = 3$

Available Hardware Blocks

Rev1

You're allowed to use whatever hardware blocks (even to use all the blocks) to improve the performance

Tc x Tm	3 x 4	2 x 8	8 x 2	2 x 4	4 x 2
L1	HLS3x4_1	NA	NA	NA	NA
L2	NA	HLS2x8_2	HLS8x2_2	HLS2x4_2	HLS4x2_2
L3	NA	HLS2x8_3	HLS8x2_3	HLS2x4_3	HLS4x2_3
L4	NA	HLS2x8_4	HLS8x2_4	HLS2x4_4	HLS4x2_4
L5	NA	HLS2x8_5	HLS8x2_5	HLS2x4_5	HLS4x2_5

HLS2x8.zip

HLS8x2.zip

HLS2x4.zip

HLS4x2.zip

Group 4 should run **two** layers in HW **simultaneously** (refer to the paper in p. 31)

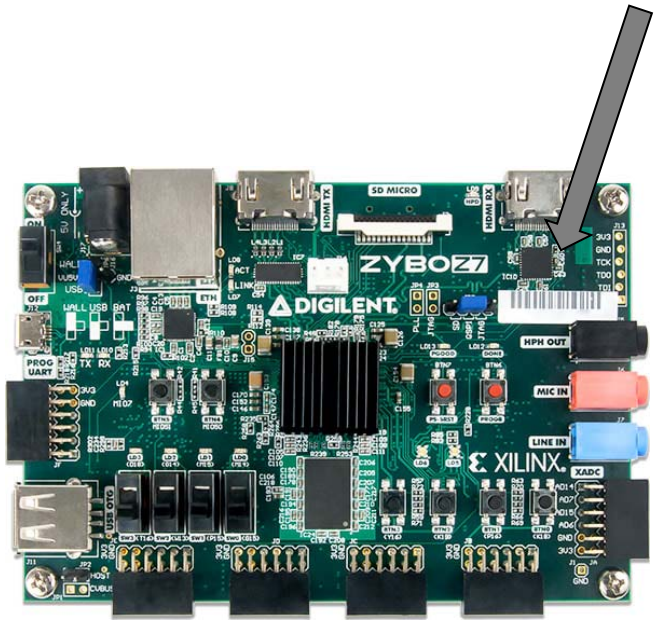
Design Flow

Rev2

Vivado

SDK

ZYNQ/Zybo



Design Constraints

- You are allowed to modify only the following function where all the multiply-accumulate (MAC) operations are implemented in HW
  - **conv_hw**
- You are not allowed to change anything outside the above function

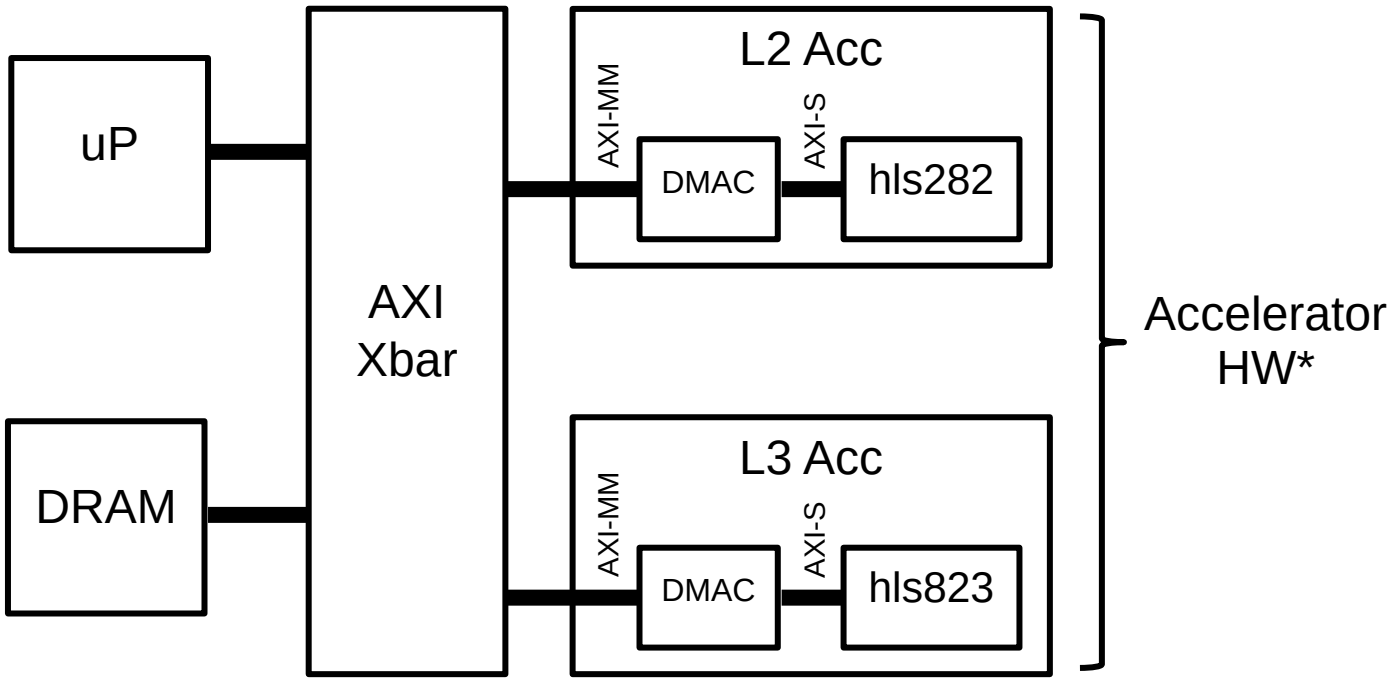
```
void conv_ref(){
    // Layer #1
    convolution_S(ofmap1, ifmap1, fmap1, M_C1, C_C1, F_C1, E_C1, R_C1, S_C1, H_C1, W_C1, U_C1, P_C1);
    bias(ofmap1, bias1, M_C1, E_C1, F_C1);
    relu(ofmap1, M_C1, E_C1, F_C1);
    LRN(ofmap11, ofmap1, 5, 0.0001, 0.75, 1.0, M_C1, E_C1, F_C1);
    pooling(ofmap1p, ofmap11, E_C1, F_C1, M_C1, E_P1, F_P1, K_P1, S_P1); //Pool #1
    paddata(ofmap1pp, ofmap1p, M_C1, E_P1, F_P1, P_C2); //zero padding added for standard convolution

    // Layer #2
    convolution_G(ofmap2, ofmap1pp, fmap2, M_C2, C_C2, F_C2, E_C2, R_C2, S_C2, H_C2, W_C2, 0, G_C2);
    bias(ofmap2, bias2, M_C2, E_C2, F_C2);
    relu(ofmap2, M_C2, E_C2, F_C2);
    LRN(ofmap21, ofmap2, 5, 0.0001, 0.75, 1.0, M_C2, E_C2, F_C2);
    pooling(ofmap2p, ofmap21, E_C2, F_C2, M_C2, E_P2, F_P2, K_P2, S_P2); //Pool #2
    paddata(ofmap2pp, ofmap2p, M_C2, E_P2, F_P2, P_C3); //zero padding added for standard convolution

    // Layer #3
    convolution_B(ofmap3, ofmap2pp, fmap3, M_C3, C_C3, F_C3, E_C3, R_C3, S_C3, H_C3, W_C3, 0);
    bias(ofmap3, bias3, M_C3, E_C3, F_C3);
    relu(ofmap3, M_C3, E_C3, F_C3);
    paddata(ofmap3p, ofmap3, M_C3, E_P3, E_P3, P_C4); //zero padding added for standard convolution

    // Layer #4
    convolution_G(ofmap4, ofmap3p, fmap4, M_C4, C_C4, F_C4, E_C4, R_C4, S_C4, H_C4, W_C4, 0, G_C4);
    bias(ofmap4, bias4, M_C4, E_C4, F_C4);
    relu(ofmap4, M_C4, E_C4, F_C4);
    paddata(ofmap4p, ofmap4, M_C3, E_P3, E_P3, P_C4); //zero padding added for standard convolution

    // Layer #5
    convolution_G(ofmap5, ofmap4p, fmap5, M_C5, C_C5, F_C5, E_C5, R_C5, S_C5, H_C5, W_C5, 0, G_C5);
    bias(ofmap5, bias5, M_C5, E_C5, F_C5);
    relu(ofmap5, M_C5, E_C5, F_C5);
    pooling(ofmap5p, ofmap5, E_C5, F_C5, M_C5, H_C6, W_C6, K_P3, S_P3); //Pool #3
}
```

```
void conv_hw(){
    // Layer #1
    convolution_S(ofmap1, ifmap1, fmap1, M_C1, C_C1, F_C1, E_C1, R_C1, S_C1, H_C1, W_C1, U_C1, P_C1);
    bias(ofmap1, bias1, M_C1, E_C1, F_C1);
    relu(ofmap1, M_C1, E_C1, F_C1);
    LRN(ofmap11, ofmap1, 5, 0.0001, 0.75, 1.0, M_C1, E_C1, F_C1);
    pooling(ofmap1p, ofmap11, E_C1, F_C1, M_C1, E_P1, F_P1, K_P1, S_P1); //Pool #1
    paddata(ofmap1pp, ofmap1p, M_C1, E_P1, F_P1, P_C2); //zero padding added for standard convolution

    // Layer #2
    convolution_G_hw(XPAR_AXI_DMA_0_DEVICE_ID, XPAR_AXI_DMA_1_DEVICE_ID, XPAR_AXI_DMA_2_DEVICE_ID,
                     ofmap2, ofmap1pp, fmap2, M_C2, C_C2, F_C2, E_C2, R_C2, S_C2, H_C2, W_C2, G_C2, 8, 2, 27, 27);
    bias(ofmap2, bias2, M_C2, E_C2, F_C2);
    relu(ofmap2, M_C2, E_C2, F_C2);
    LRN(ofmap21, ofmap2, 5, 0.0001, 0.75, 1.0, M_C2, E_C2, F_C2);
    pooling(ofmap2p, ofmap21, E_C2, F_C2, M_C2, E_P2, F_P2, K_P2, S_P2); //Pool #2
    paddata(ofmap2pp, ofmap2p, M_C2, E_P2, F_P2, P_C3); //zero padding added for standard convolution

    // Layer #3
    convolution_B(ofmap3, ofmap2pp, fmap3, M_C3, C_C3, F_C3, E_C3, R_C3, S_C3, H_C3, W_C3, 0);
    bias(ofmap3, bias3, M_C3, E_C3, F_C3);
    relu(ofmap3, M_C3, E_C3, F_C3);
    paddata(ofmap3p, ofmap3, M_C3, E_P3, E_P3, P_C4); //zero padding added for standard convolution

    // Layer #4
    convolution_G(ofmap4, ofmap3p, fmap4, M_C4, C_C4, F_C4, E_C4, R_C4, S_C4, H_C4, W_C4, 0, G_C4);
    bias(ofmap4, bias4, M_C4, E_C4, F_C4);
    relu(ofmap4, M_C4, E_C4, F_C4);
    paddata(ofmap4p, ofmap4, M_C3, E_P3, E_P3, P_C4); //zero padding added for standard convolution

    // Layer #5
    convolution_G(ofmap5, ofmap4p, fmap5, M_C5, C_C5, F_C5, E_C5, R_C5, S_C5, H_C5, W_C5, 0, G_C5);
    bias(ofmap5, bias5, M_C5, E_C5, F_C5);
    relu(ofmap5, M_C5, E_C5, F_C5);
    pooling(ofmap5p, ofmap5, E_C5, F_C5, M_C5, H_C6, W_C6, K_P3, S_P3); //Pool #3
}
```

You're allowed to modify only this part

Design Constraints

- Example of accelerator
  - Layers 2 & 3 in HW and the others in SW



* The FPGA considered here (ZC7020) is assumed to accommodate no more than **220 MAC hardware units** and **570KBs BRAM**

Evaluation

- Submission completeness
  - Reproducibility (10pt)
- Accuracy
  - Classification error (20pt)
  - Quantization error (10pt)
- Performance
  - Execution time (30pt)
- Technical solidness
  - Any good ideas or contributions (30pt)

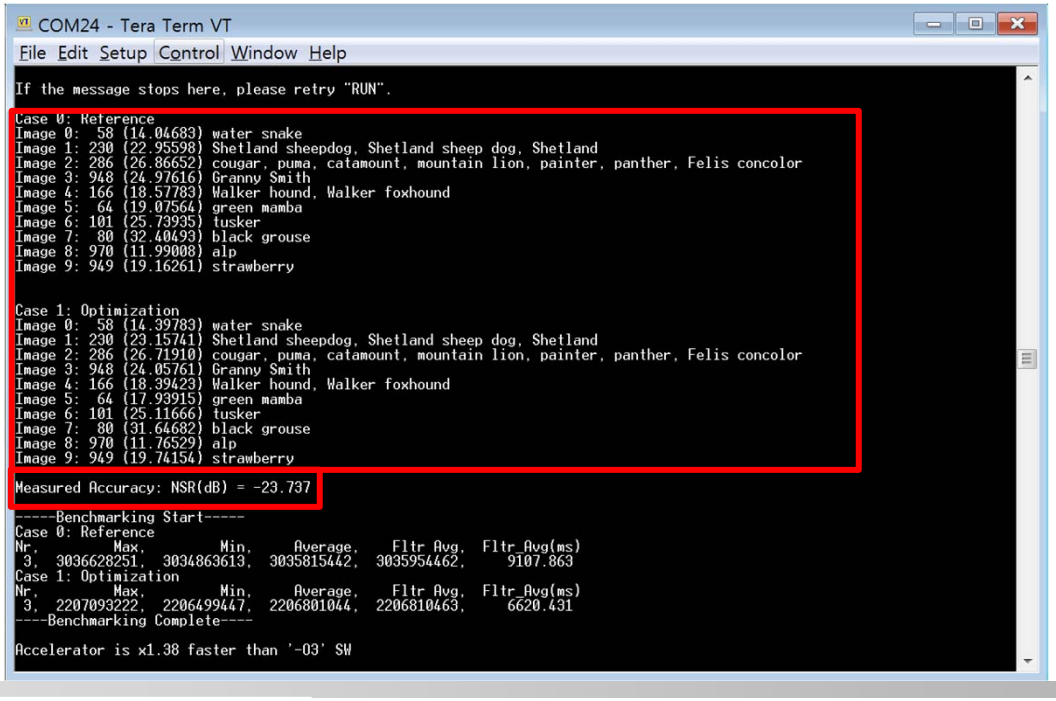
Reproducibility

- Make sure that your submission is reproducible
  - In other words, it is possible to reproduce your design together with the claimed accuracy and performance using only the files that you submitted

Accuracy

Rev2

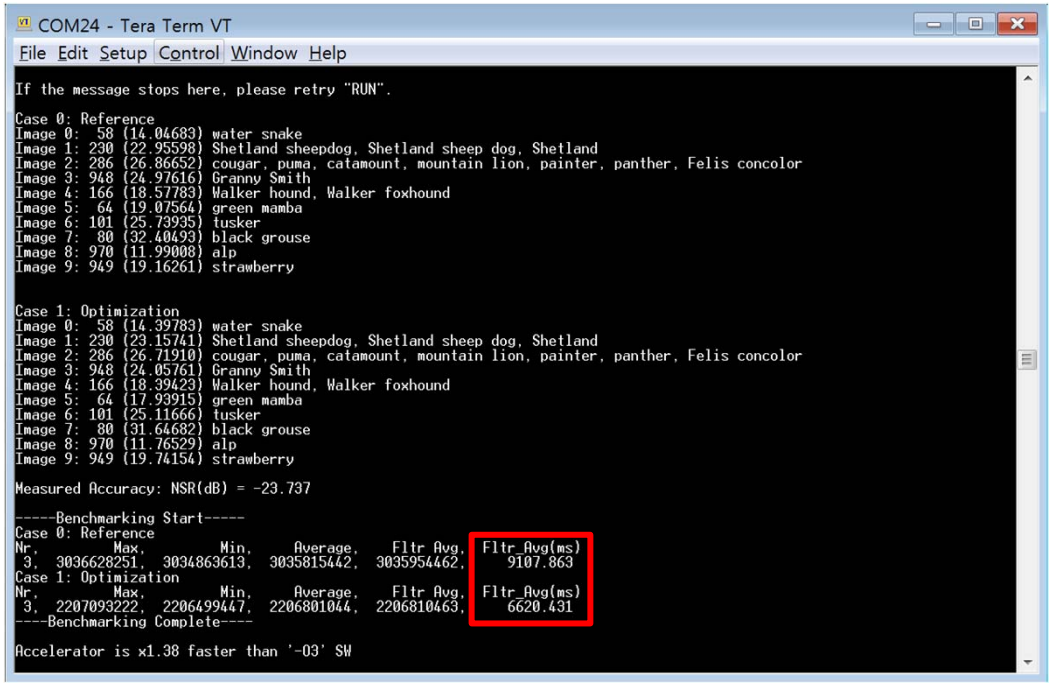
- Run the program to check the accuracy of **conv_hw**
  - Classification error
    - ✓A new set of images will be given onsite
  - Noise-to-signal ratio (NSR)
    - ✓Difference from **conv_ref**



Performance

Rev2

- Run the program to check the performance of **conv_hw**
  - Execution time (-O3)



Technical Solidness

- Any good ideas or contributions that help to improve the performance such as
  - Optimum allocation of accelerator resources (MAC units & BRAMs) across layers
  - Maximum utilization of accelerator resources
- For the state-of-the-art hardware accelerator for CNN, refer to the following paper (and those citing it – available at http://ieeexplore.ieee.org):
  - “Maximizing CNN accelerator efficiency through resource partitioning”, Y. Shen et al., ISCA 2017

Important Notes

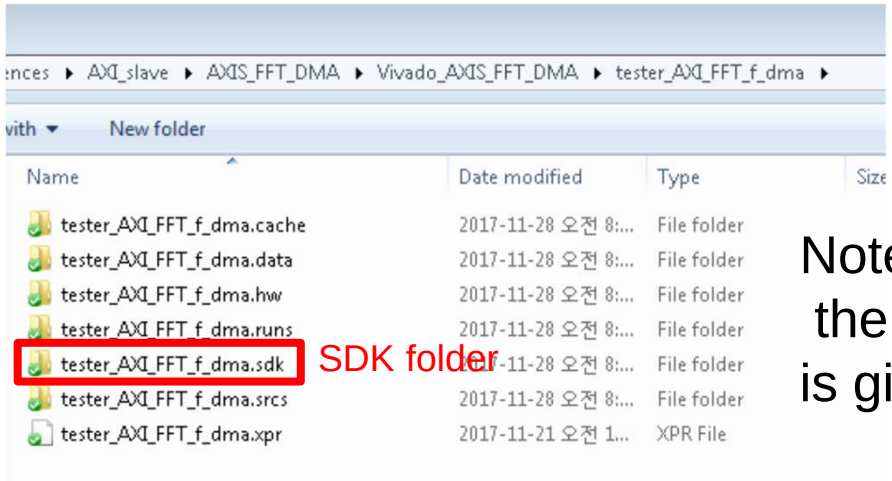
- You should keep the reference implementation (**`conv_ref`**) unchanged
- The compiler optimization level should be set to -O3 when the performance is measured
- You will have some bonus points if you implement some of the paper ideas cited in the previous slide
- You may be able to provide two versions of the convolution function, e.g.,
  - 1st version with the best performance
  - 2nd version implementing paper idea(s)

Submission

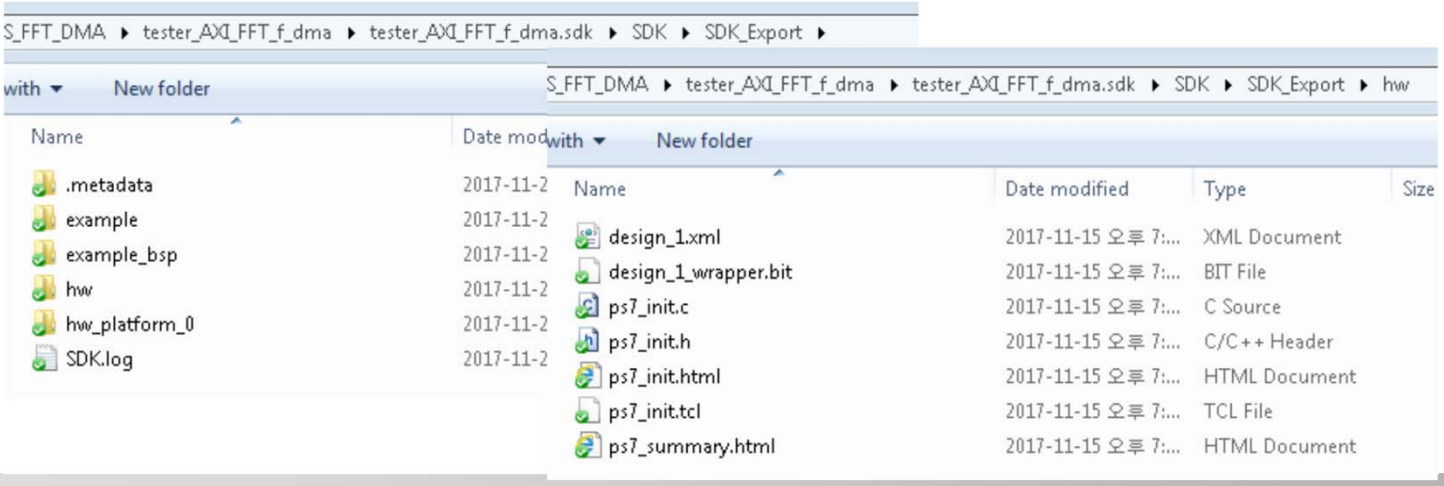
- Due date: **Dec. 15 (Sun), 2019, 15:00:00** GMT+9
- Zip the following files into a single file and send it to chesterku2013@gmail.com
  - **C/Verilog source/header files** and any other files that are needed to *reproduce* your design
  - Copy of the entire **SDK folder** in your project (including the bitstream)
  - **Slideset file** (PPT) including the results in the Console window
- You can post a question in the Q&A of the course page (https://www.sites.google.com/site/kusocdesignlab/q-a-2)
- No delay will be acceptable!

Submission

- SDK folder in your project



Note that the name of
the SDK folder generally
is given as “[project name].sdk"



Presentation & Demo

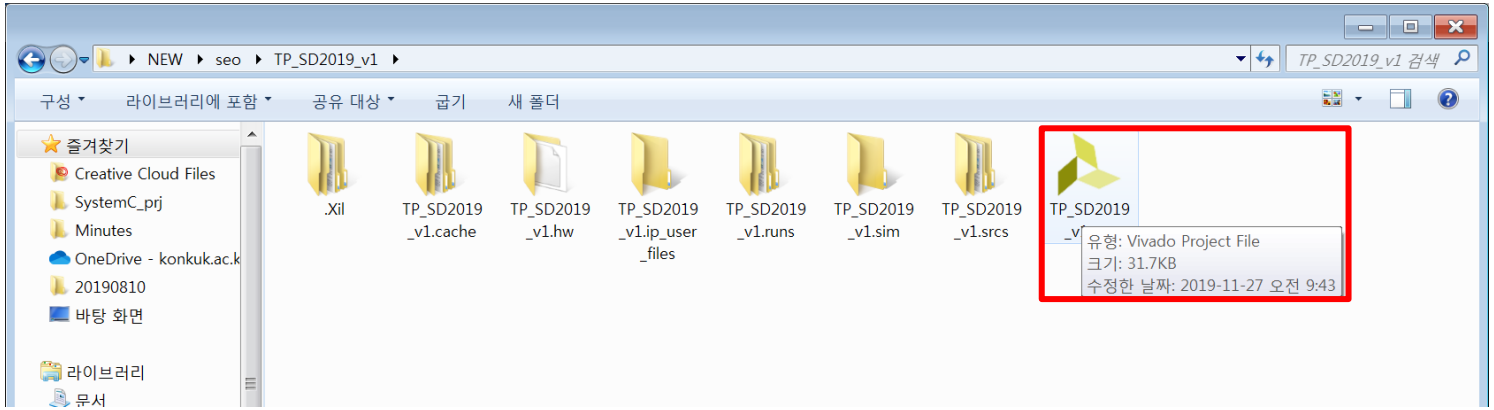
- **Dec. 16 (Mon) 10:30~13:30, New Eng. Bldg. #1113**
  - Team-presentation with exactly the same **slideset files** as submitted on **Dec. 15**
  - Team-demo with exactly the same **SDK folder** as submitted on **Dec. 15**

1. *Bring a storage device (e.g., HDD) having the entire **project folder** just in case (e.g., when the SDK folder does not work)*
2. *Note that any progress made later than **Dec. 15** can**not** be counted for evaluation*

Appendix

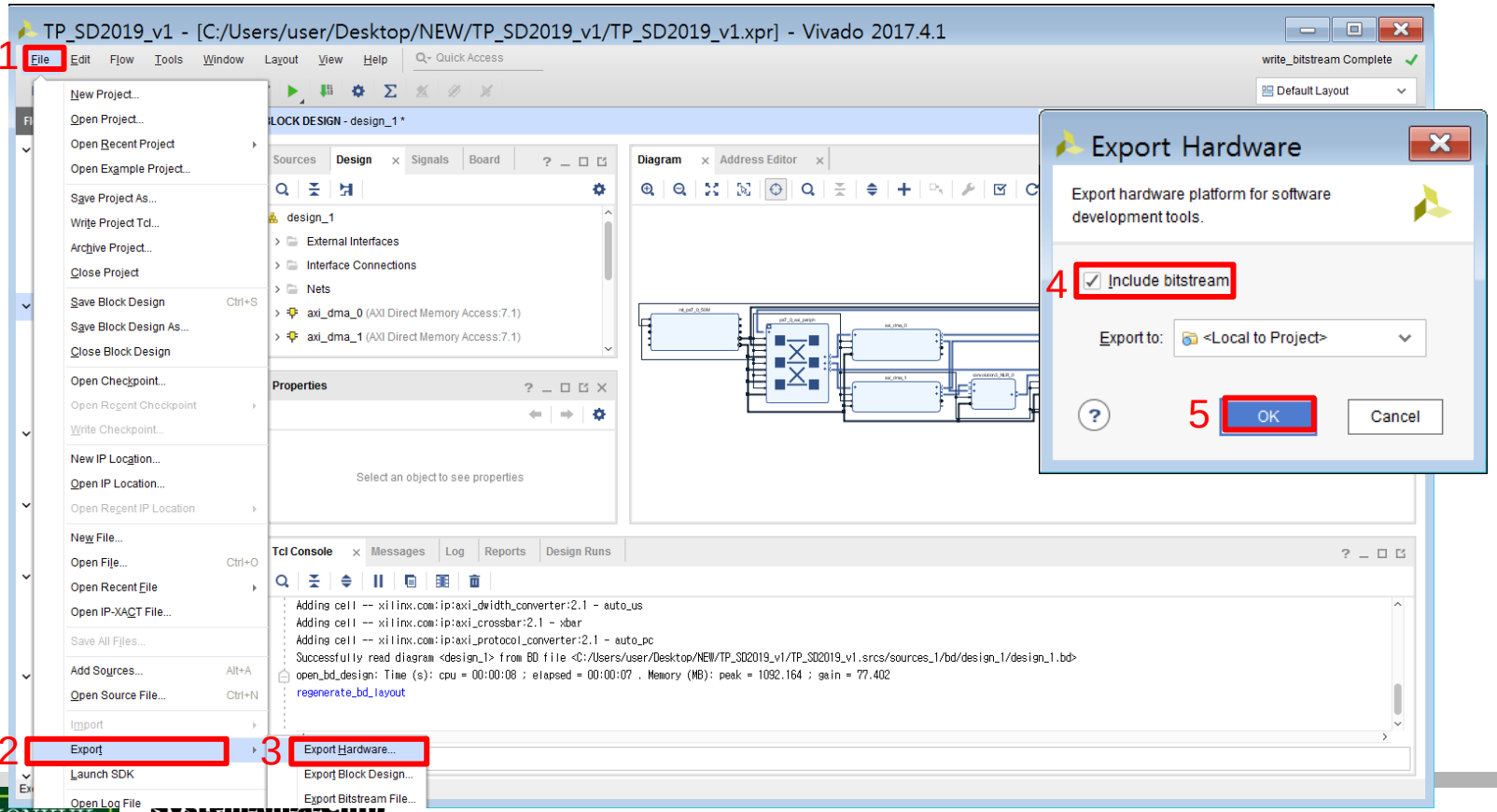
Open Vivado projects

- Open Vivado projects
  - Unzip and open '***TP_SD2019_prj_v2***' folder and run file **'TP_SD2019_v2.xpr':** https://drive.google.com/open?id=11uuEABQbbdKVmTsBIWDNwLgnz41QEQCz



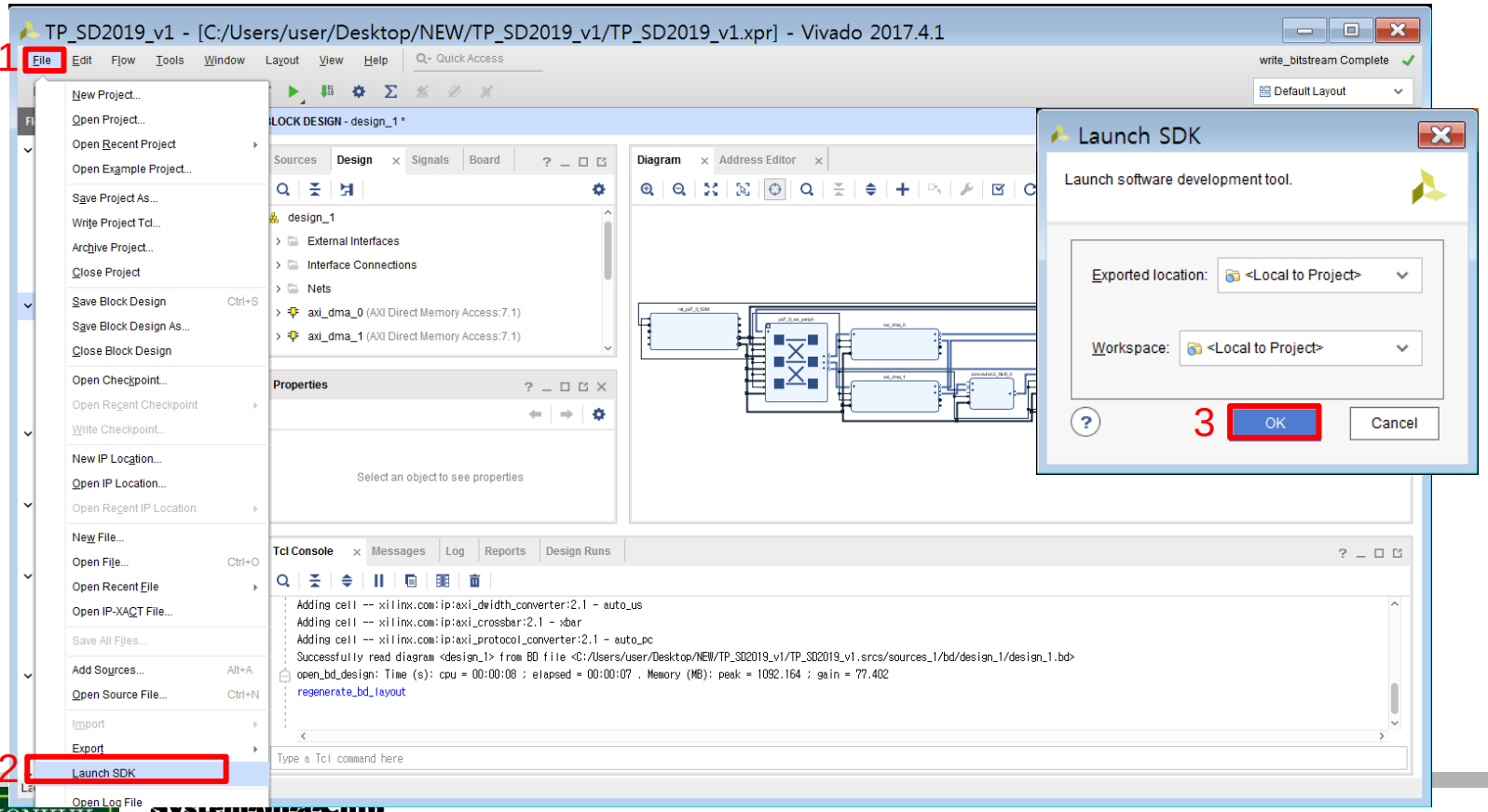
Open Vivado projects

- Export Hardware
  - Click ***'Export > Export Hardware'*** in ***'File'*** menu
  - Check ***'Include Bitstream'*** and then click ***'OK'***



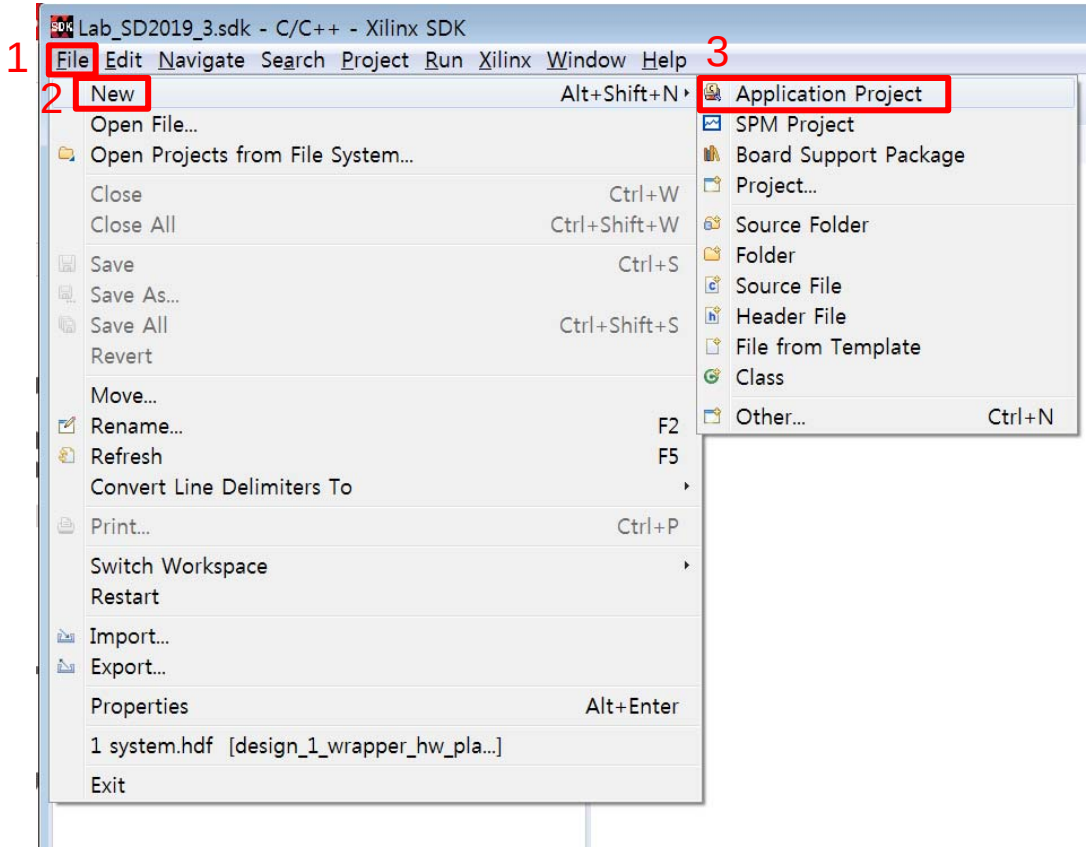
Open Vivado projects

- Launch SDK
  - Click ***'Launch SDK'*** in ***'File'*** menu
  - Click ***'OK'***



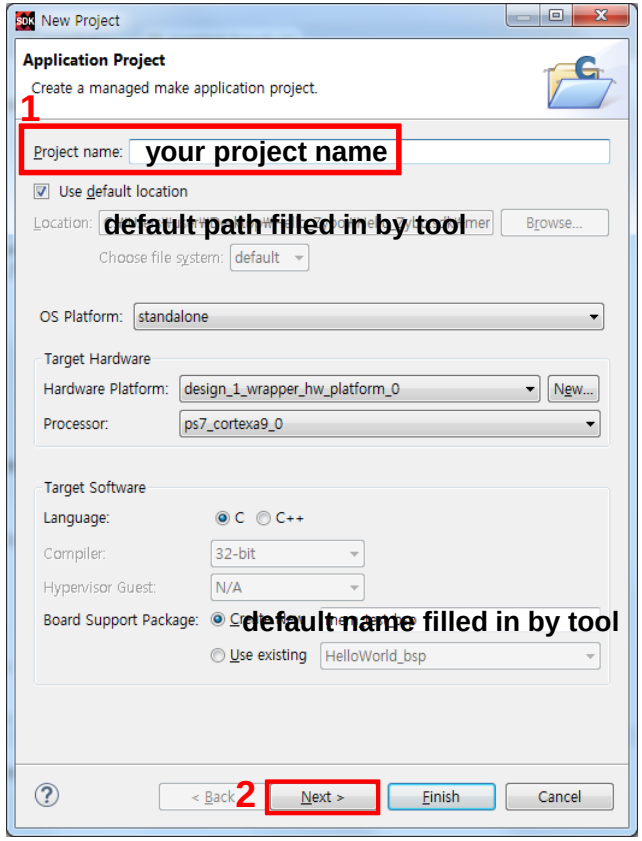
Running C Applications

- Create a C application project
  - Click ***'File' > 'New' > 'Application Project'***



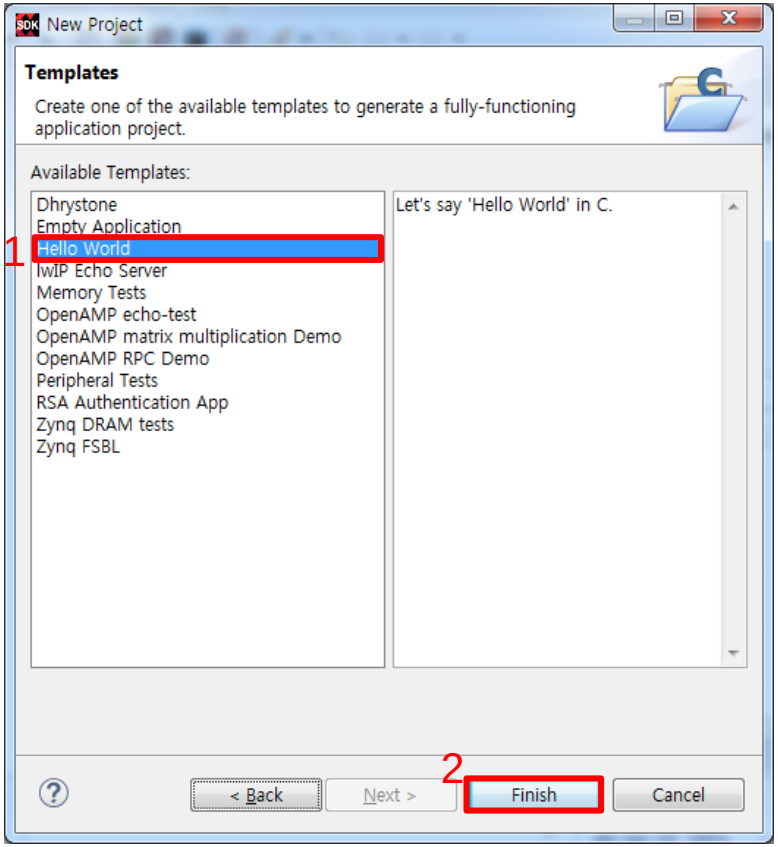
Running C Applications

- Create a C application project (cont'd)
  - Type the project name
  - Click ***'Next'***



Running C Applications

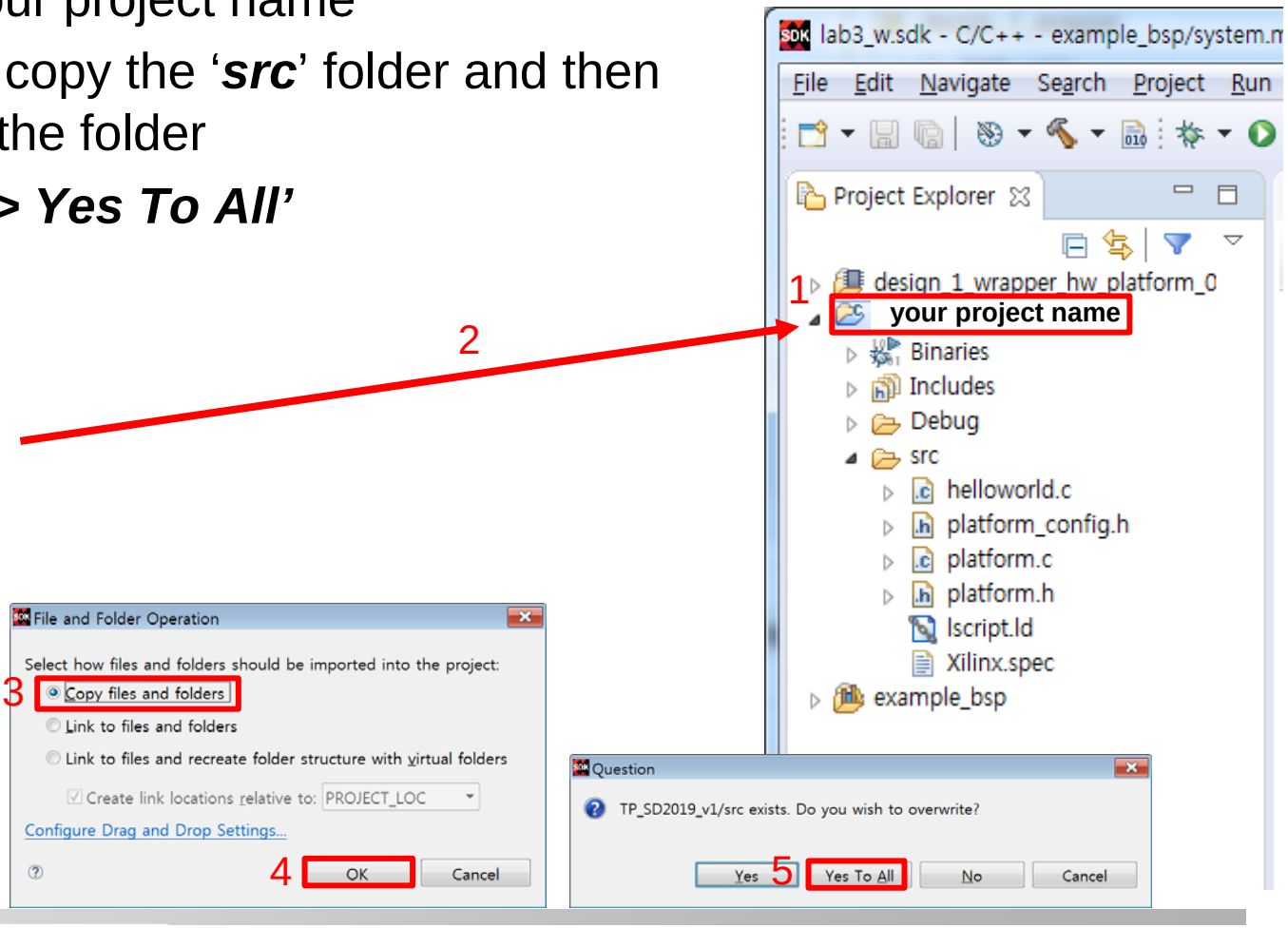
- Create a C application project (cont'd)
  - Choose ***'Hello World'*** and then click ***'Finish'***



Running C Applications

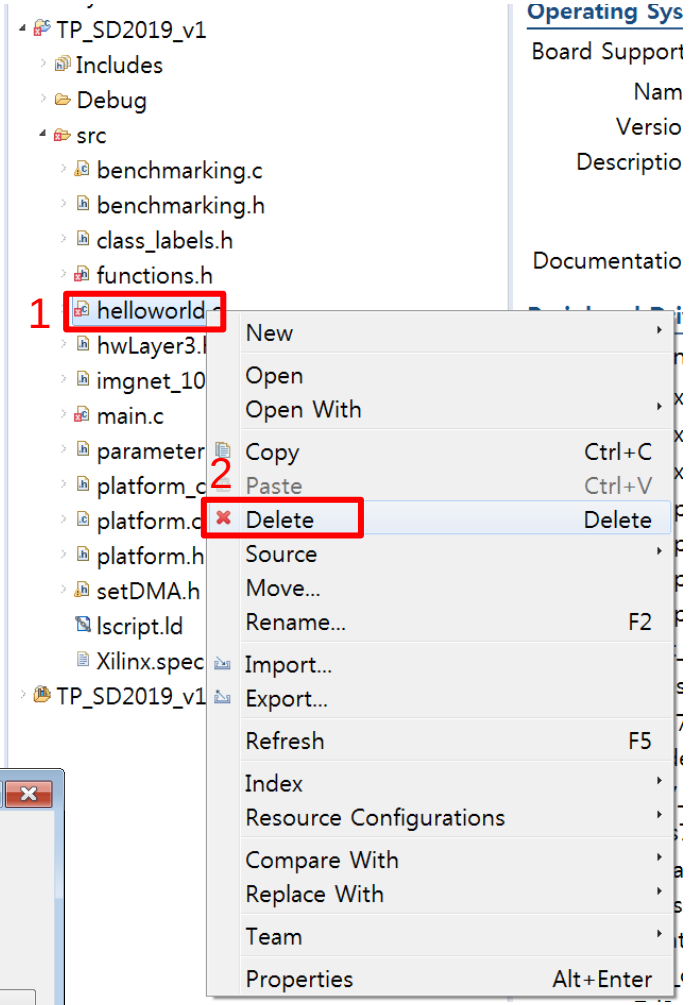
- Add source files
  - Choose your project name
  - Unzip and copy the '***src***' folder and then paste into the folder
  - Click ***'OK > Yes To All'***

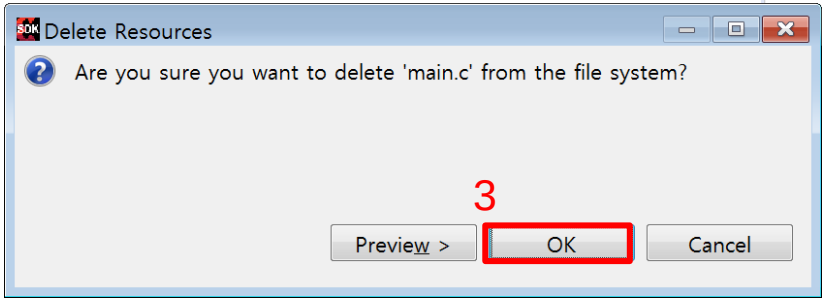
TP_SD2019_src.zip



Running C Applications

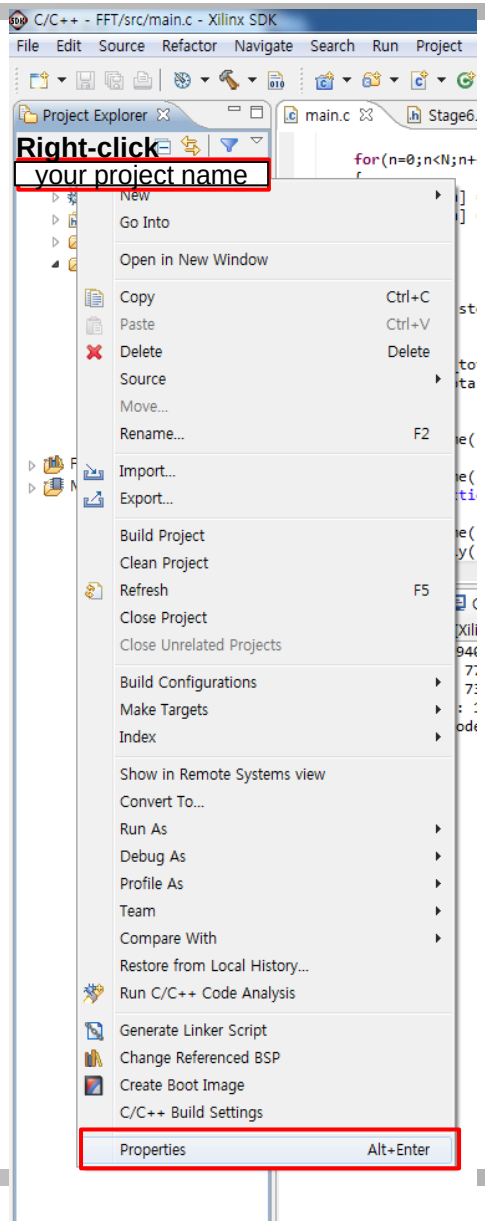
- Add source files (cont'd)
  - Right-click '***hellowrld.c***' in the '***src***' folder
  - Click ***'Delete > OK'***





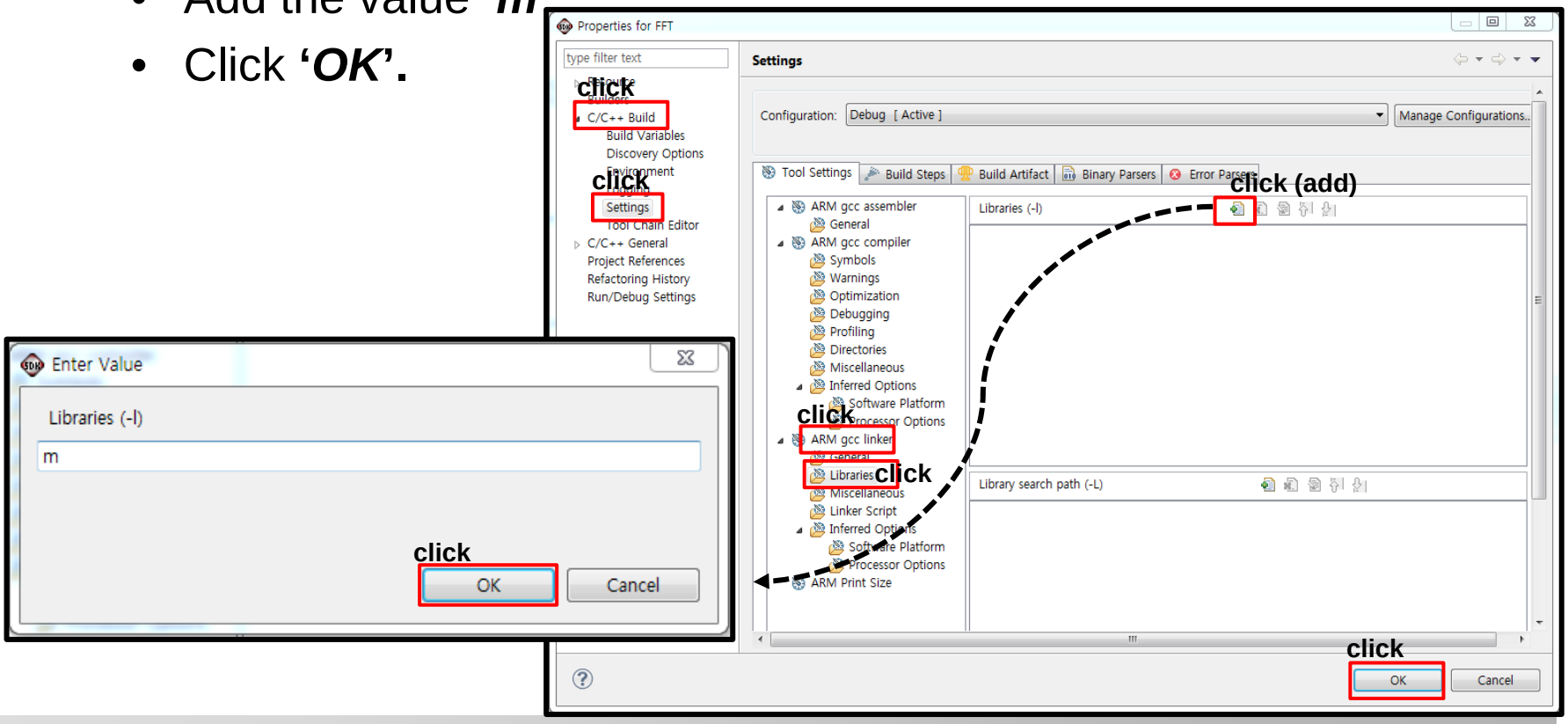
Running C Applications

- Adding Math Library
  - Project must have ***'–lm library'*** to use ***'math.h'*** header file.
  - Right-click ***'your project name'*** and then click ***'Properties'***



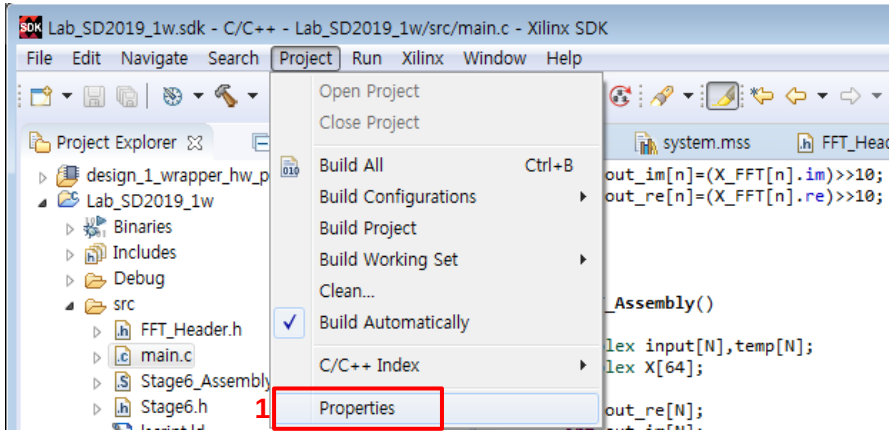
Running C Applications

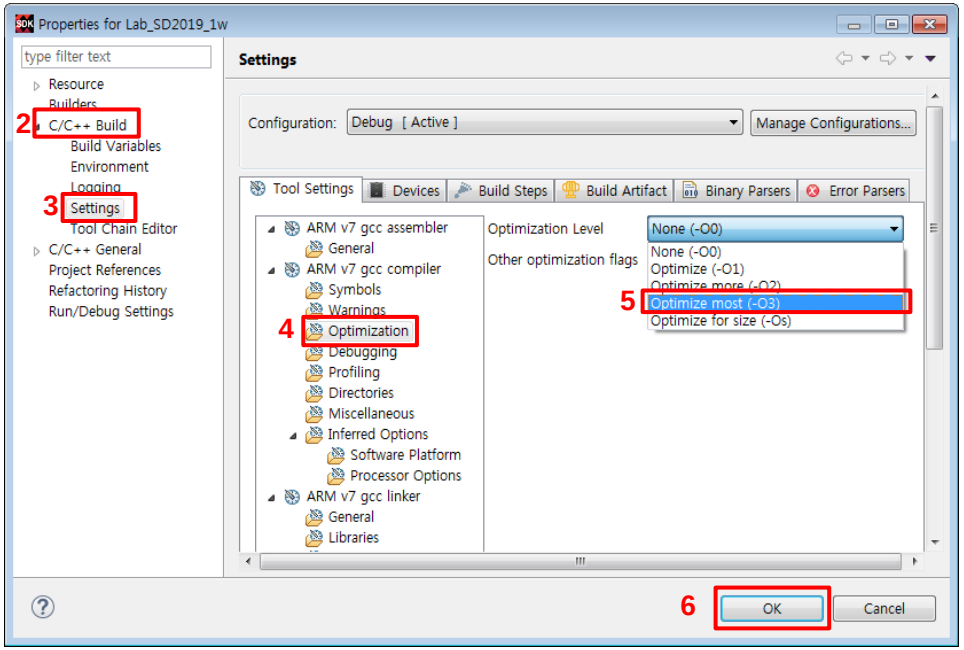
- Adding Math Library (cont'd)
  - Click ***'C/C++ Build*** > ***Settings*** > ***ARM gcc linker*** > ***libraries*** > ***add'***
  - Add the value ***'m'***
  - Click '***OK***'.



Running C Applications

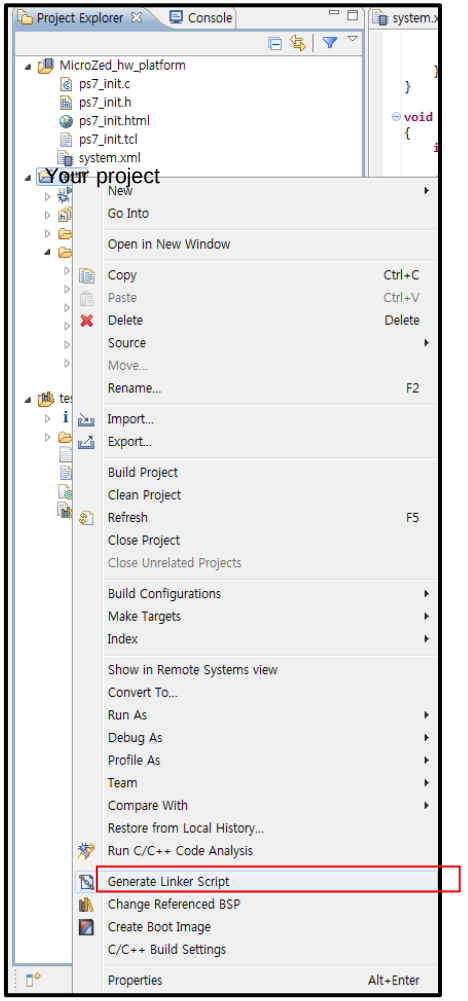
- Set the compiler optimization level
  - Select ***'Project'*** menu and click ***'Properties'***
  - Select ***'Settings'*** tap and click ***'ARM v7 gcc compilier > Optimization'***
  - Select ***'Optimization most (-O3)'*** in the dropdown menu of ***'Optimization Level'***
  - Click ***'OK'***





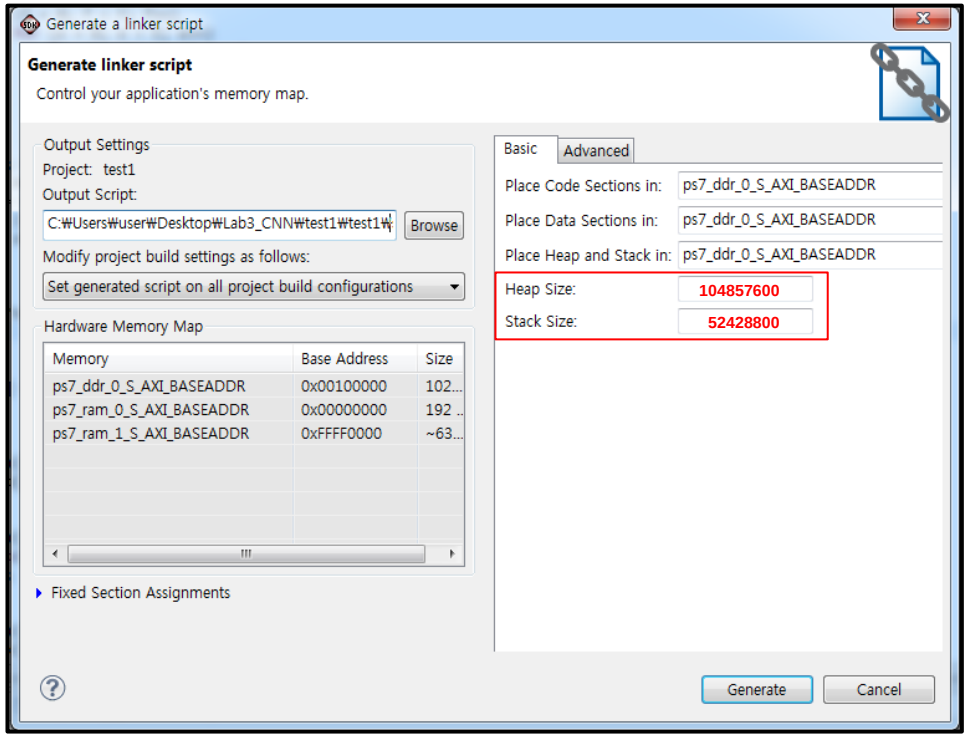
Running C Applications

- Setting Stack & Heap Size
  - Select the application project in the ***Project Explorer*** or ***C/C++ Projects*** view
  - Right-click ***'your project name'*** and then click '***Generate Linker Script'*** or click '**Xilinx Tools > Generate Linker script'**



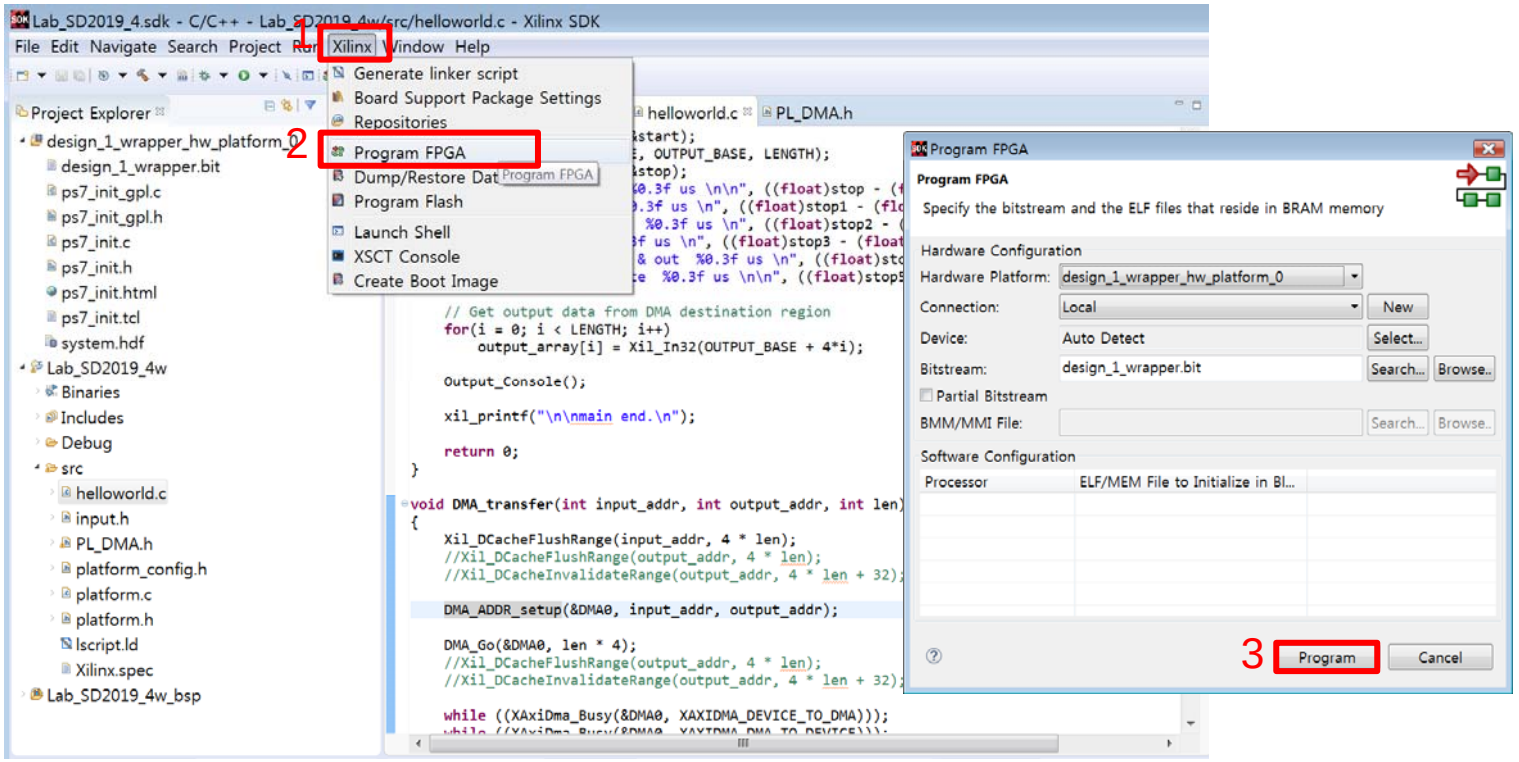
Running C Applications

- Setting Stack & Heap Size (Cont'd)
  - Set both the heap and stack sizes in the ***Basic*** tab to ***104857600*** (100 MB) and ***52428800*** (50 MB) respectively, as shown below



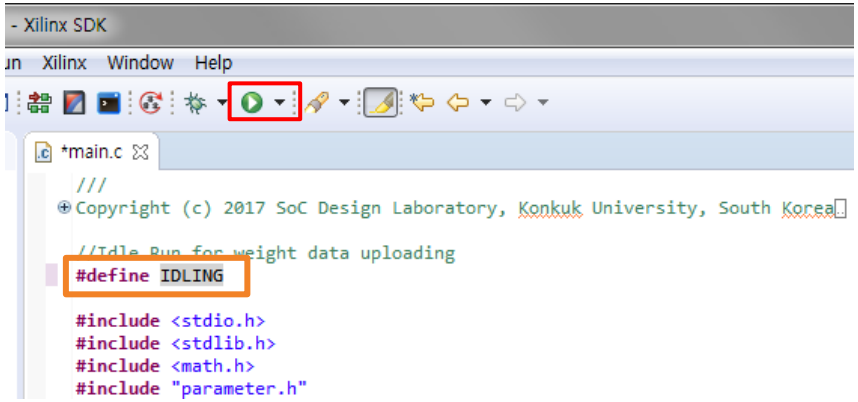
Running C Applications

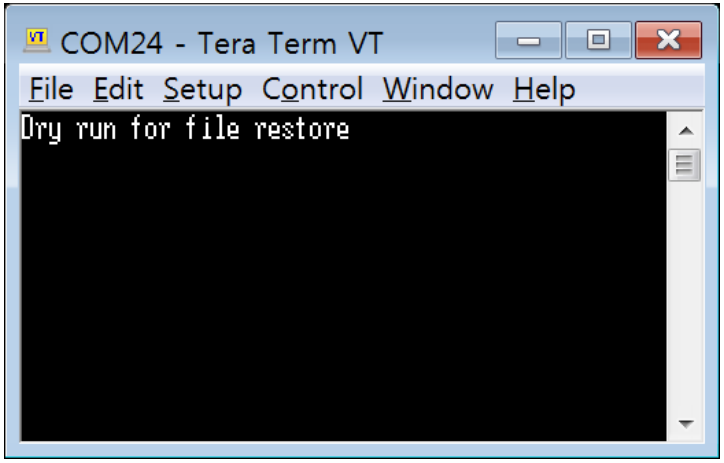
- Program FPGA
  - Choose the ***'Xilinx'*** menu and then click ***'Program FPGA'***
  - Click ***'Program'***



Running C Applications

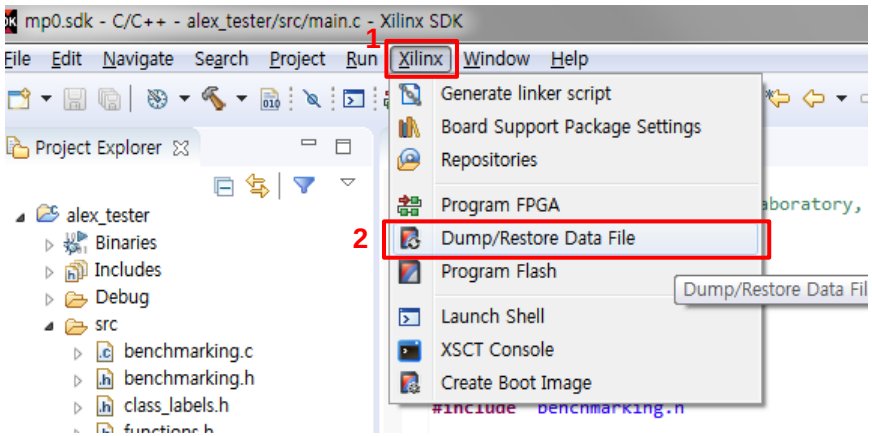
- Check the source code: **main.c**
  - Make sure that **'#define IDLING'** is written at the top of the source code
- Run the application
  - This application is simply for uploading the convolution weights into the DDR





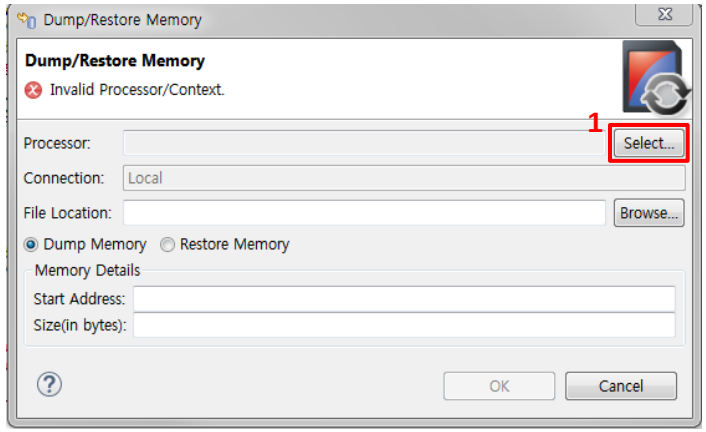
Running C Applications

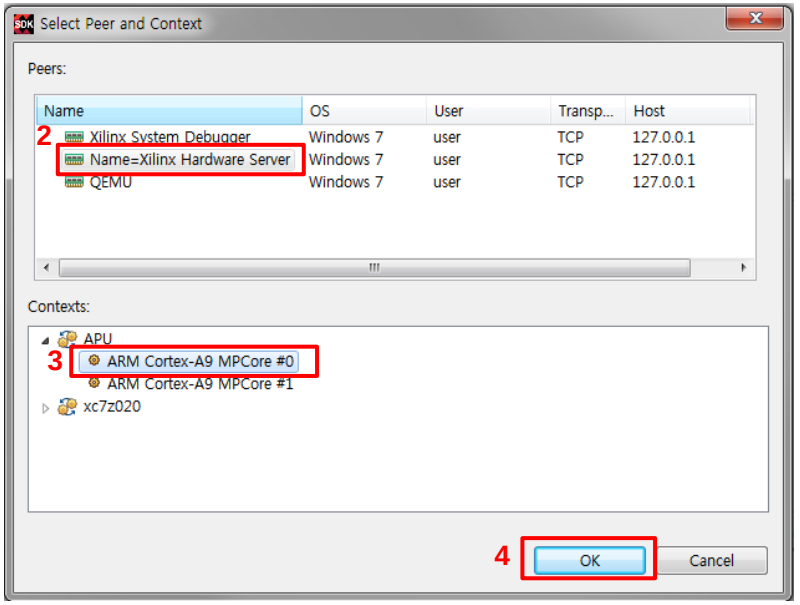
- Restore Memory
  - Click '***Xilinx > Dump/Restore Data File***'



Running C Applications

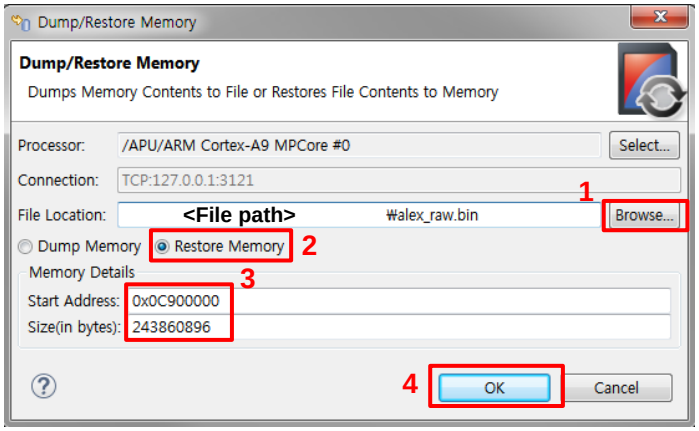
- Restore Memory (cont'd)
  - Click '***Select***'
  - Select '***Name=Xilinx Hardware Server***'
  - Select '***ARM Cortex-A9 MPCore #0***'
  - Click '***OK***'





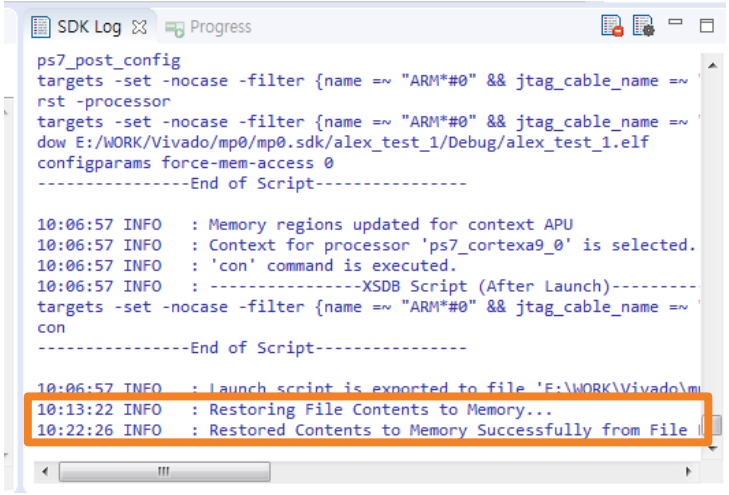
Running C Applications

- Restore Memory (cont'd)
  - Click '***Browse***' and select the file named **alex_raw.bin**: https://drive.google.com/open?id=122wv9t3FVrXlQYgc9bk7eK_L5-Q8ansA
  - Select '***Restore Memory***'
  - Type **0x0C900000** on '**Start Address**' and **243860896** on '**Size(in bytes)**'



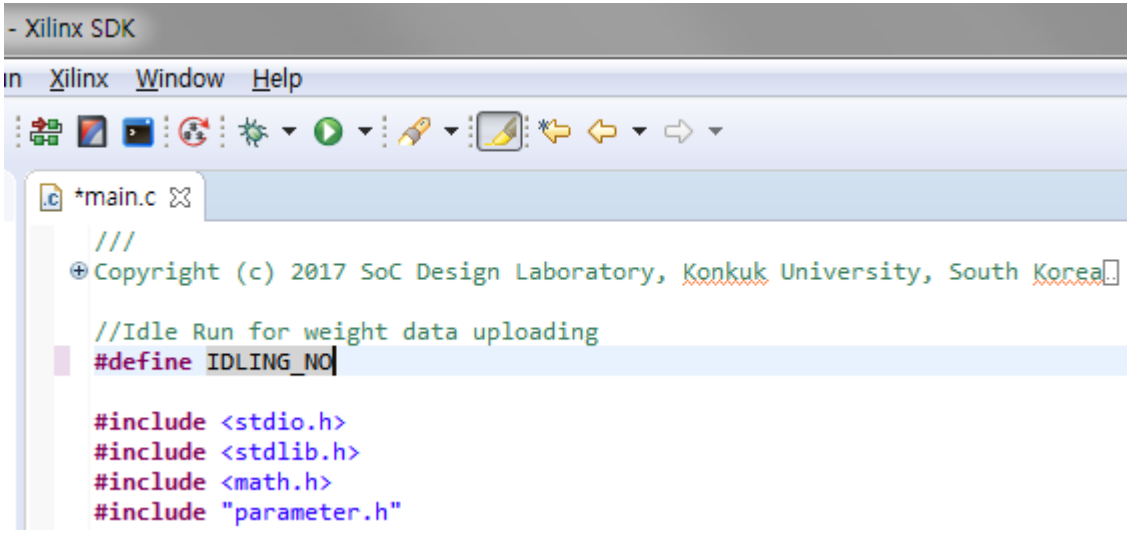
Running C Applications

- Restore Memory (Cont'd)
  - Check the SDK log window (which may take about 9 minutes) to see the message '***Restored Contents to Memory Successfully from File ~***'



Running C Applications

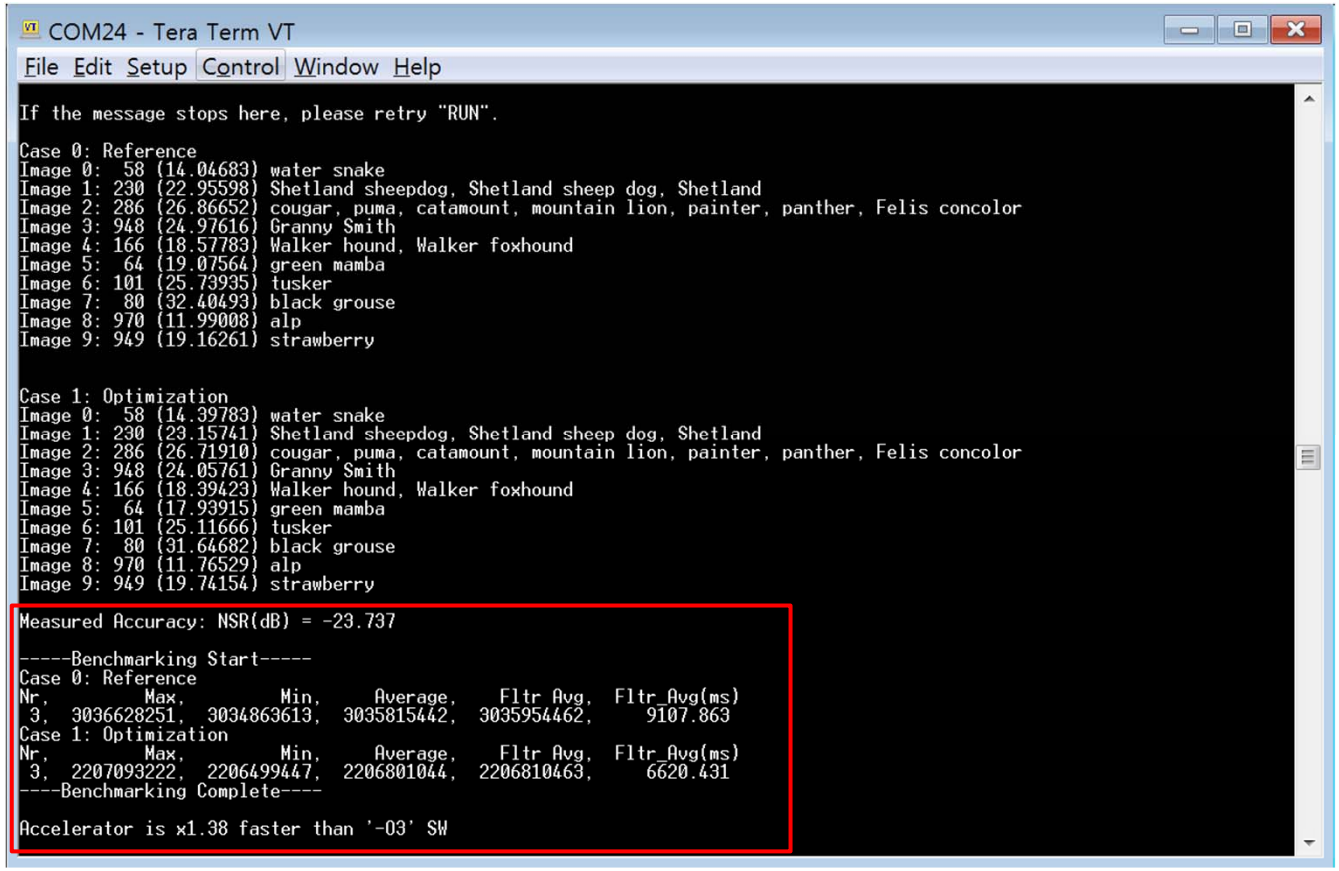
- Modify the source code: ***main.c***
  - Change **'#define IDLING'** to **'#define IDLING_NO'** (or anything different from **'IDLING'**)
- Run the application again
  - This application is for actually running the CNN algorithm



Running C Applications

Rev2

- Run the application again (Cont'd)
  - Check the output of the application on ***'Tera Term'***



Creating IP Projects

- Repeat the previous steps
  - Follow pp. 4~17 of the following lab workbook: **Lab_SD2019_3w.pdf**

Creating Block Designs

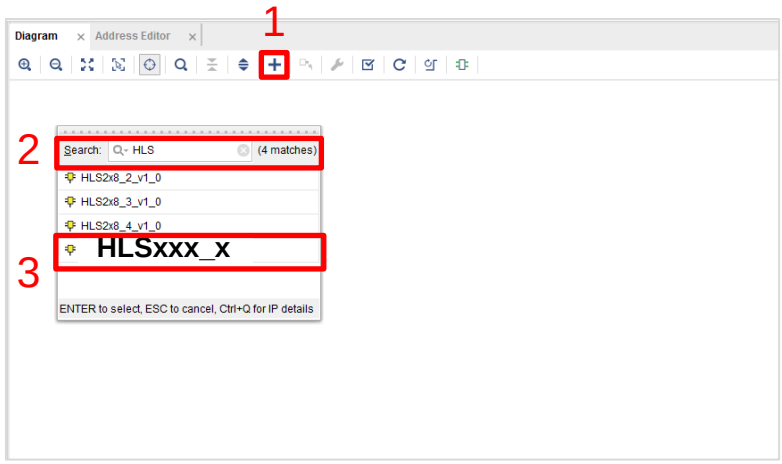
Rev2

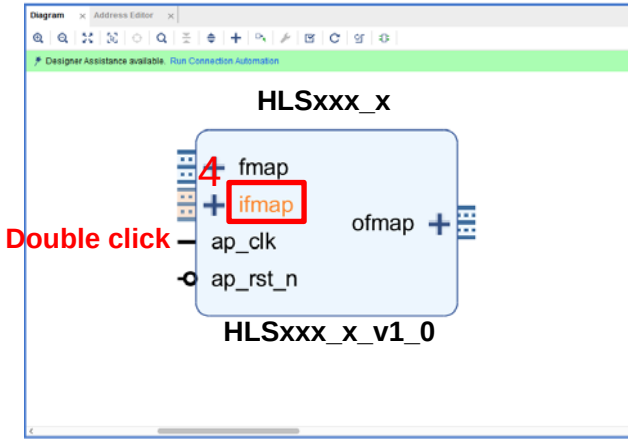
- Repeat the previous steps for '**Add IP repository**'
  - Follow pp. 19~23 of the following lab workbook: **Lab_SD2019_3w.pdf**

Creating Block Designs

Rev2

- Add HLS Hardware Block
  - Click the ***'Add IP'*** icon and type "***HLSxxx_x"*** in the Search field
  - Double-click ***'HLSxxx_x'***
  - Double-click the ***'ifmap'*** pin in ***'HLSxxx_x'*** block

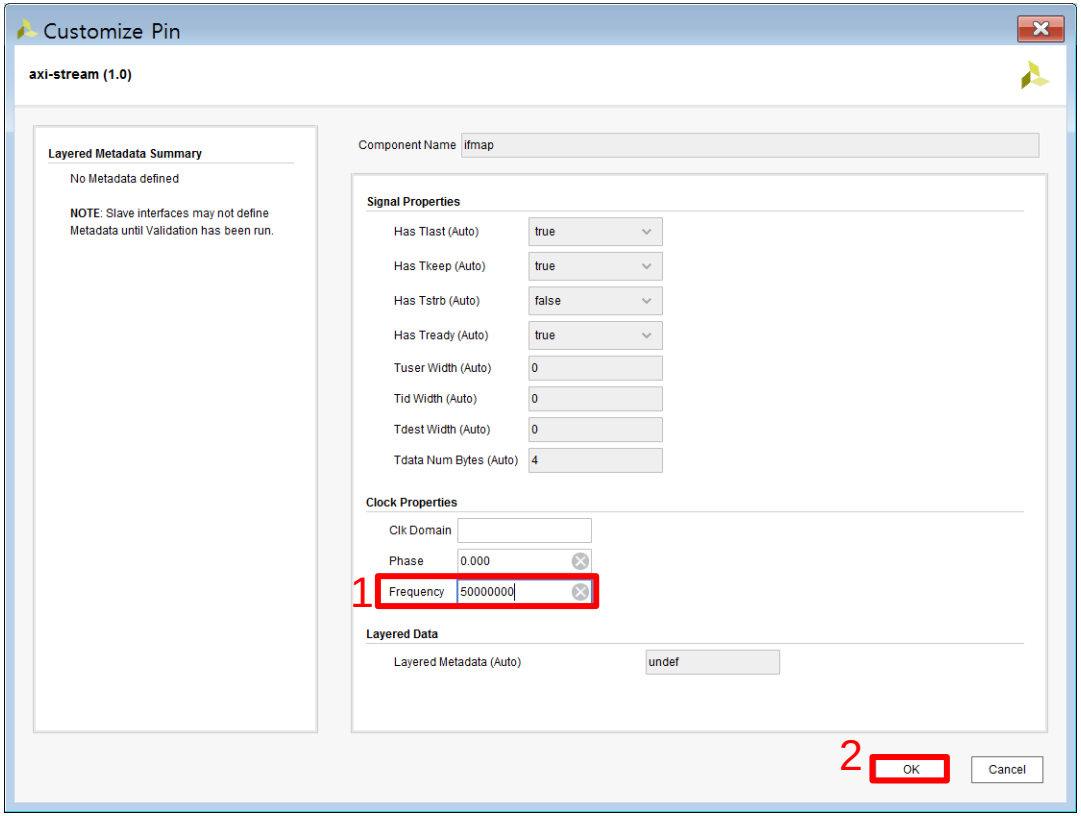




Creating Block Designs

Rev2

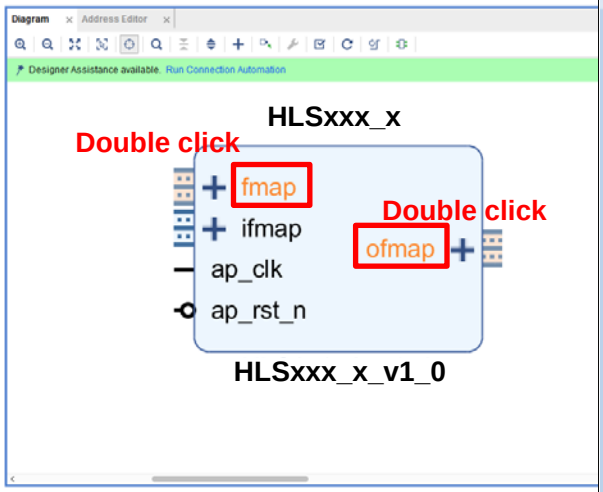
- Customize Pin
  - Type ***'50000000'*** on ***'Frequency'*** tap and then click ***'OK'***

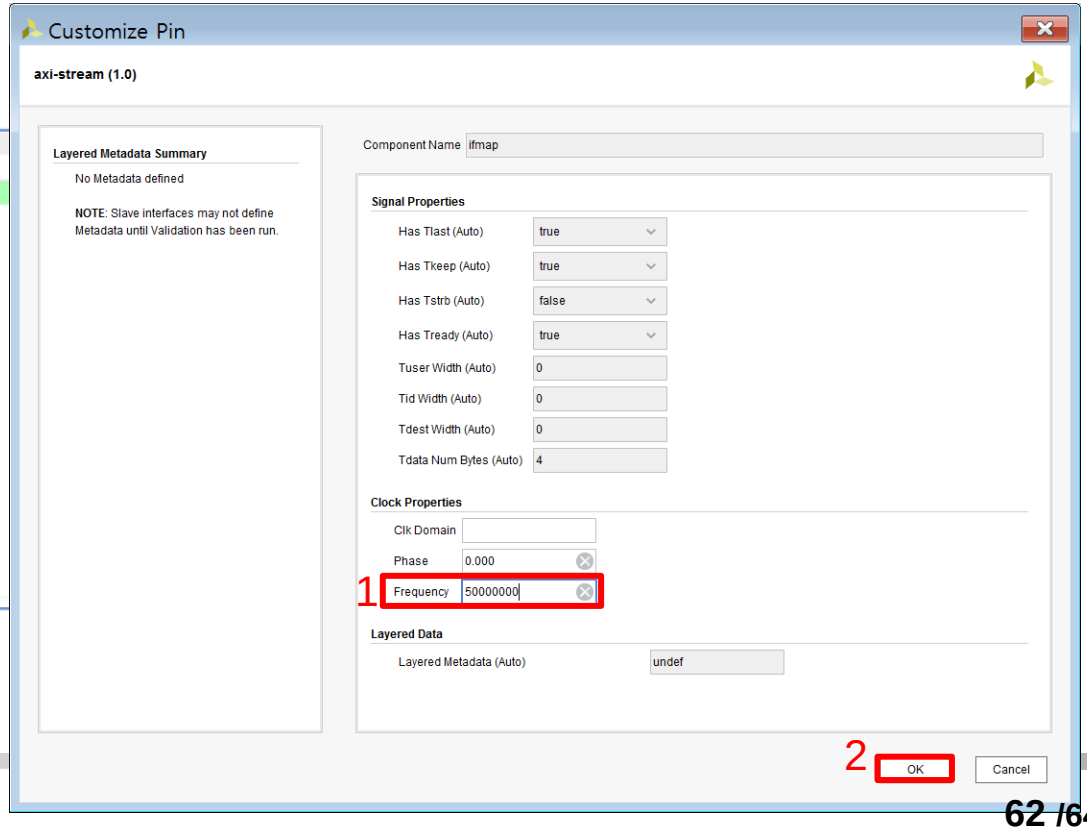


Creating Block Designs

Rev2

- Customize Pin (cont'd)
  - Double-click the ***'fmap'*** and ***'ofmap'*** pins in the ***'HLSxxx_x'*** block
  - Type ***'50000000'*** in the ***'Frequency'*** field and then click ***'OK'***





Creating Block Designs

Rev2

- Repeat the previous steps for '**Add DMA controller**'
  - Follow pp. 13~14 of the following lab workbook: **Lab_SD2019_4w.pdf**

Creating Block Designs

Rev2

- Re-customize IP
  - Type ***'23'*** in the ***'Width of Buffer Length Register'*** field
  - Click ***'OK'***