Additional Material

- **IP on Board**

Computing Memory Architecture Lab.

Overview

- **How to check HW system**
  - Processing System + BRAM + Connectivity + Custom IP (M*M)

- **How to run HW system with SW**
  - Convolution lowering on my Custom IP (M*M) on FPGA

How to check HW system

In this part, we’ll check your bitstream file is properly generated.

Edit Interface

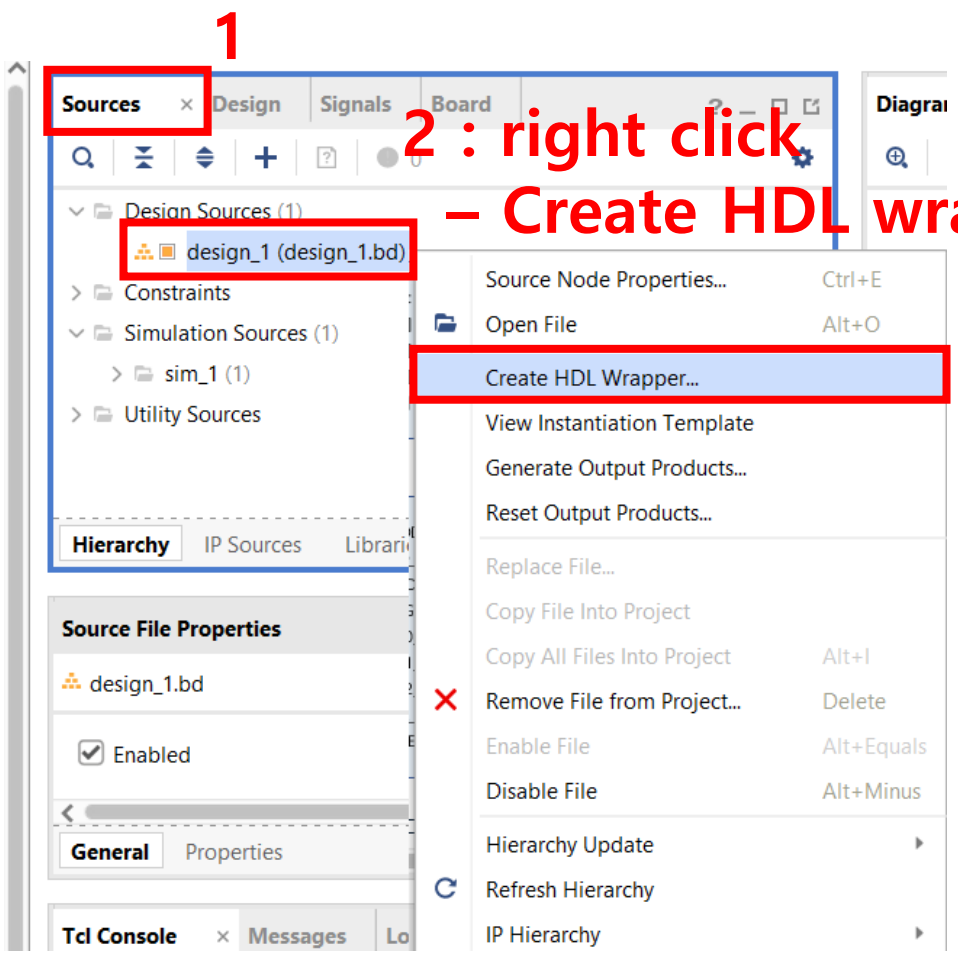
- Edit `lab10_ip_repo/myip_1.0/hdl/myip_v1_0_S00_AXI.v`
  - Refer lab10 slides for more details

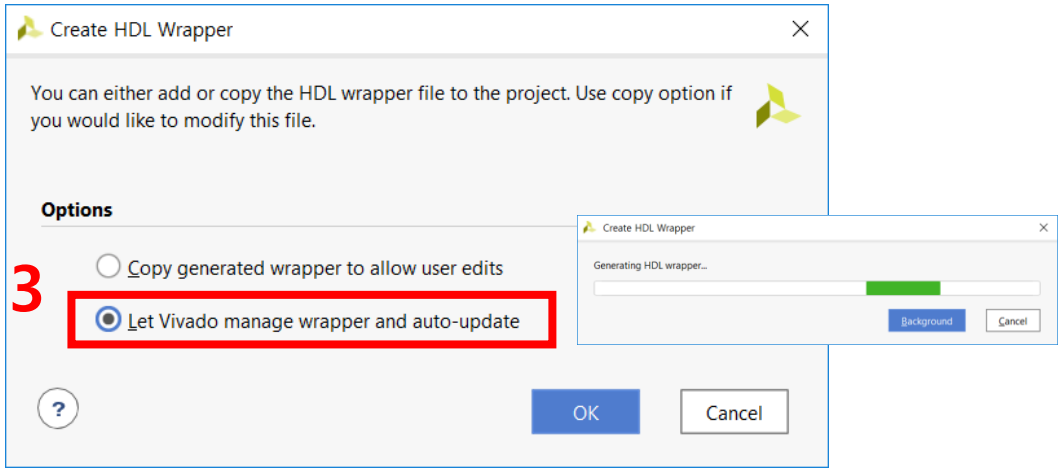
```
401     // Add user logic here
402     clk_wiz_0 u_clk_180 (.clk_out1(BRAM_CLK), .clk_in1(S_AXI_ACLK));
403     assign BRAM_EN = 1'b1;
404     assign BRAM_RST = 1'b0;
405     wire bram_write;
406     assign BRAM_WE = (bram_write)? 4'hF : 4'h0;
407
408     reg [1:0] bram_counter;
409     wire [31:0] bram_wr_addr = {28'd1, bram_counter, 2'd0};
410     wire [31:0] bram_rd_addr = {28'd0, bram_counter, 2'd0};
411     assign BRAM_ADDR = (bram_write)? bram_wr_addr : bram_rd_addr;
```

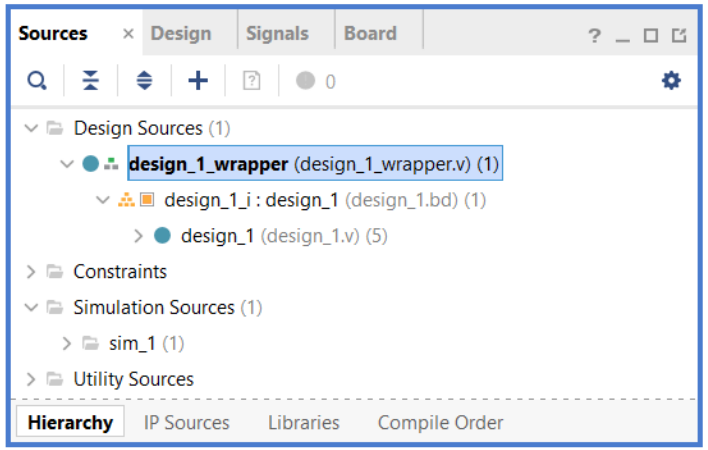
Editing custom IP

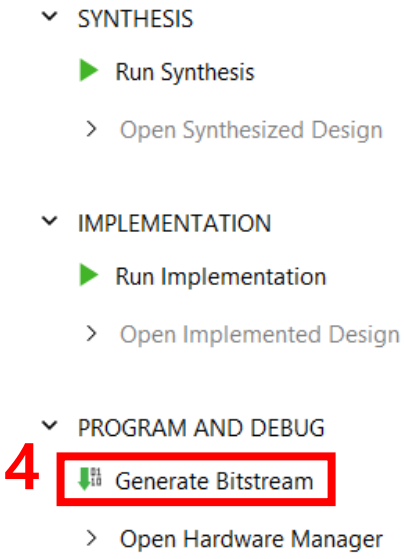
- Attach your custom IP (M*M) by editing custom IP
  - Processing System + BRAM + Connectivity + (Shifter => Custom IP (M*M))
  - Refer appendix on Lab10

Generate bitstream of your IP - MM



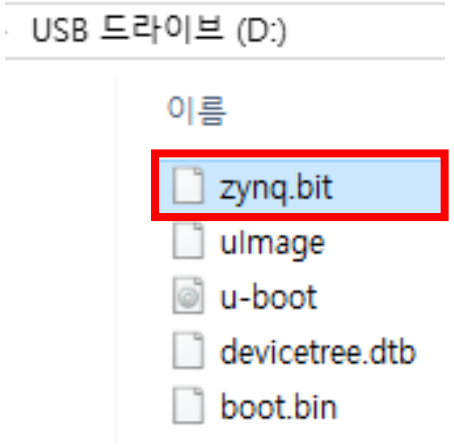






Editing custom IP

- Prepare your bitstream file
  - Change zynq.bit in sd card with generated bit file (.bit)
  - Insert sd card to board



Verify your zynq.bit

- Verification codes
  - BOARD$ git clone https://github.com/tahsd/hsd21_add01_checkhw
- Implement test MM multiplication function (=largeMM)
  - Implement hsd21_add01_checkhw /src/fpga_api.cpp
  - Refer lab09 codes you've implemented

```
void FPGA::largeMM(const float* weight_mat, const float* input_mat,
                   float* output, int num_input, int num_output, int num_matrix2)
{
  float* m1 = this->matrix_M1();
  float* m2 = this->matrix_M2();

  // 0) Initialize output vector
  for(int i = 0; i < num_output*num_matrix2; ++i)
    output[i] = 0;

  for(int i = 0; i < num_output; i += v_size_)
  {
    for(int j = 0; j < num_input; j += v_size_)
    {
      for(int k = 0; k < num_matrix2; k += v_size_)
      {
        // 0) Initialize input vector
        // IMPLEMENT THIS

        // 1) Assign a m1
        // IMPLEMENT THIS

        // 2) Assign a m2
        // IMPLEMENT THIS

        // 3) Call a function `blockMM() to execute Matrix matrix multiplication
        const float* ret = this->blockMM();

        // 4) Accumulate intermediate results
        // IMPLEMENT THIS

      }
    }
  }
}
```

Verify your zynq.bit

- Run
  - Boot with your zynq.bit
  - BOARD$ sudo g++ -I ./include verify.cpp ./src/fpga_api.cpp -o run.out && sudo ./run.out

- Compare the outputs from CPU and FPGA(MM)
  - Output: some MM results of our IP
  - Both should be same
    - Slight differences are acceptable

```
program start
index       CPU         FPGA
0           1.836571    1.836571    %
1           2.588725    2.588725    %
2           2.984560    2.984560    %
3           3.674901    3.674901    %
4           3.222222    3.222223    %
5           2.906765    2.906765    %
6           2.833551    2.833551    %
7           3.107897    3.107897    %
8           1.606581    1.606581    %
9           2.257570    2.257570    %
10          2.642685    2.642685    %
11          2.914744    2.914744    %
12          3.035270    3.035270    %
13          2.768578    2.768578    %
14          2.426602    2.426603    %
15          2.922653    2.922653    %
16          0.980174    0.980174    %
17          1.811520    1.811519    %
```

acceptable

Verify your zynq.bit

- If it doesn't work, you need to re-check your IP generation.
  - Refer appendix in lab10

- If it gives weird value, you need to re-check your Verilog codes and implemented c codes.

```
program start
index     CPU        FPGA
0         1.836571   1.836571  %
1         2.588725   2.588725  %
2         2.984560   2.984560  %
3         3.674901   3.674901  %
4         3.222222   3.222223  %
5         2.906765   2.906765  %
6         2.833551   2.833551  %
7         3.107897   3.107897  %
8         1.606581   1.606581  %
9         2.257570   2.257570  %
10        2.642685   2.642685  %
11        2.914744   2.914744  %
12        3.035270   3.035270  %
13        2.768578   2.768578  %
14        2.426602   2.426603  %
15        2.922653   2.922653  %
16        0.980174   0.980174  %
17        1.811520   1.811519  %
```

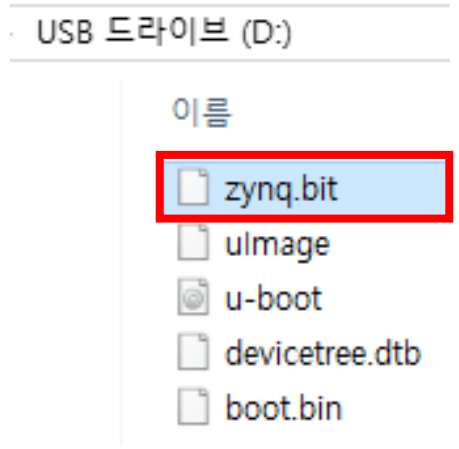
acceptable

How to run HW system with software

Editing custom IP

- Attach your custom IP (M*M) by editing custom IP
  - Processing System + BRAM + Connectivity + (Shifter => Custom IP (M*M))
  - Refer appendix on Lab10

- Prepare your bitstream file
  - Generate bitstream after editing custom IP
  - Change zynq.bit in sd card with generated bit file (.bit)
  - Insert sd card to board



Install

- **Connect your ZedBoard to the internet with LAN cable**
- Dependency download
  - $ sudo apt-get update -y
  - $ sudo apt-get install -y libprotobuf-dev protobuf-compiler python python-numpy
- Code download
  - Your project code
    - $ git clone https://github.com/tahsd/hsd21_project.git
  - $ cd hsd21_project
- Dataset download
  - $ bash download.sh

Convolution Lowering codes

- convLowering of two cpp files should be implemented in advance
  - `src/fpga_api.cpp`, `src/fpga_api_on_cpu.cpp`

```
// src/fpag_api.cpp, src/fpag_api_on_cpu.cpp
void FPGA::convLowering(const vector<vector<vector<vector<float>>>>& cnn_weights,
                        vector<vector<float>>& new_weights,
                        const vector<vector<vector<float>>>& inputs,
                        vector<vector<float>>& new_inputs) {
  /*
   * Arguments:
   *
   * conv_weights: [conv_channel, input_channel, conv_height, conv_width]
   * new_weights: [?, ?]
   * inputs: [input_channel, input_height, input_width]
   * new_inputs: [?, ?]
   *
   */
   ...
}
```

Run

- Code ready
  - Make sure `lab09/src/fpga_api.cpp`, `lab09/src/fpag_api_on_cpu.cpp` are completely implemented
    - void convLowering(…)
- Build
  - $ make
- Evaluate
  - $ **sh benchmark.sh**
  - sudo python eval.py --num_test_images 100 --v_size 8 --network cnn --run_type [cpu|fpga]

Example: Download datasets

```
root@debian-zynq:/sdcard/mm_result# sh download.sh
--2021-05-14 05:49:23--  https://dl.dropbox.com/s/mdwy0kzf57nfl5f/t10k-images.idx3-ubyte
Resolving dl.dropbox.com (dl.dropbox.com)... 162.125.82.15, 2620:100:6030:15::a27d:500f
Connecting to dl.dropbox.com (dl.dropbox.com)|162.125.82.15|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://dl.dropboxusercontent.com/s/mdwy0kzf57nfl5f/t10k-images.idx3-ubyte [following]
--2021-05-14 05:49:25--  https://dl.dropboxusercontent.com/s/mdwy0kzf57nfl5f/t10k-images.idx3-ubyte
Resolving dl.dropboxusercontent.com (dl.dropboxusercontent.com)... 162.125.82.15, 2620:100:6030:15::a27d:500f
Connecting to dl.dropboxusercontent.com (dl.dropboxusercontent.com)|162.125.82.15|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 7840016 (7.5M) [application/octet-stream]
Saving to: ‘data/t10k-images.idx3-ubyte’

data/t10k-images.idx3-ub 100%[=================================>]   7.48M  3.32MB/s   in 2.3s

2021-05-14 05:49:28 (3.32 MB/s) - ‘data/t10k-images.idx3-ubyte’ saved [7840016/7840016]

--2021-05-14 05:49:28--  https://dl.dropbox.com/s/q6gmxa2euc2bv98/t10k-labels.idx1-ubyte
Resolving dl.dropbox.com (dl.dropbox.com)... 162.125.82.15, 2620:100:6030:15::a27d:500f
Connecting to dl.dropbox.com (dl.dropbox.com)|162.125.82.15|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://dl.dropboxusercontent.com/s/q6gmxa2euc2bv98/t10k-labels.idx1-ubyte [following]
--2021-05-14 05:49:29--  https://dl.dropboxusercontent.com/s/q6gmxa2euc2bv98/t10k-labels.idx1-ubyte
Resolving dl.dropboxusercontent.com (dl.dropboxusercontent.com)... 162.125.82.15, 2620:100:6030:15::a27d:500f
Connecting to dl.dropboxusercontent.com (dl.dropboxusercontent.com)|162.125.82.15|:443... connected.
```

Example: Build your implementations

```
root@debian-zynq:/sdcard/mm_result# make
protoc -I=./proto --cpp_out=./proto proto/caffe.proto
g++ -fPIC -std=c++11 -O3 -I ./include -I./proto -o build/caffe_dnn.o -c ./src/caffe_dnn.cpp
g++ -fPIC -std=c++11 -O3 -I ./include -I./proto -o build/tf_dnn.o -c ./src/tf_dnn.cpp
g++ -fPIC -std=c++11 -O3 -I ./include -I./proto -o build/py_lib.o -c ./src/py_lib.cpp
g++ -fPIC -std=c++11 -O3 -o build/caffe.pb.o -c proto/caffe.pb.cc
g++ -fPIC -std=c++11 -O3 -I ./include -I./proto -o build/common_dnn.o -c ./src/common_dnn.cpp
g++ -fPIC -std=c++11 -O3 -I ./include -I./proto -o build/fpga_api_on_cpu.o -c src/fpga_api_on_cpu.
cpp
g++ -shared -o build/libpylib_cpu.so  build/py_lib.o build/caffe.pb.o build/caffe_dnn.o build/tf_d
nn.o build/common_dnn.o build/fpga_api_on_cpu.o -lprotobuf
g++ -fPIC -std=c++11 -O3 -I ./include -I./proto -o build/fpga_api.o -c src/fpga_api.cpp
g++ -shared -o build/libpylib_fpga.so  build/py_lib.o build/caffe.pb.o build/caffe_dnn.o build/tf_
dnn.o build/common_dnn.o build/fpga_api.o -lprotobuf
```

Example: Evaluation

```
root@debian-zynq:/sdcard/mm_result# python eval.py --num_test_images 100 --v_size 8 --network cnn --run_type fpga
[*] Arguments: Namespace(m_size=8, network='cnn', num_test_images=100, run_type='fpga', v_size=8)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 2206,
 'm_size': 8,
 'total_image': 100,
 'total_time': [redacted]
 'v_size': 8}
```

- Accuracy(100 images)
  - CNN(Lab 09): around 100%

Example: Benchmark

- $ sudo bash benchmark.sh
  - Benchmark results
    - Cpu convolution
    - Fpga convolution

```
root@debian-zynq:/sdcard/mm_result# sh benchmark.sh
[*] Arguments: Namespace(m_size=8, network='cnn', num_test_images=100, run_type='cpu', v_size=8)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 2206,
 'm_size': 8,
 'total_image': 100,
 'total_time': [redacted]
 'v_size': 8}
-e
=> Accuracy should be 1.0

[*] Arguments: Namespace(m_size=8, network='cnn', num_test_images=100, run_type='fpga', v_size=8)
[*] Read MNIST...
[*] The shape of image: (100, 28, 28)
[*] Load the network...
[*] Run tests...
[*] Statistics...
{'accuracy': 1.0,
 'avg_num_call': 2206,
 'm_size': 8,
 'total_image': 100,
 'total_time': [redacted]
 'v_size': 8}
-e
=> Accuracy should be 1.0
```

Custom IP (M*M) works on MLP!

Term Project v0 최종제출

- Requirements
  - Result
    - Attach your project folder with all your verilog codes
    - Attach a screenshot of benchmark.sh result with your [student-number, name]
    - Attach a video that can show your [ZedBoard + benchmark.sh] both
      - Below is a description of the scenario in which you take a video
        - 1. Keep the camera away and make sure your host computer and ZedBoard are all visible
        - 2. Turn on your ZedBoard
        - 3. Zoom that camera into the host monitor & start minicom
        - 4. Run benchmark.sh

    Report
    - Explain HW System that you implemented
    - In your own words
    - Either in Korean or in English
    - # of pages does not matter
    - **PDF only!!**
  - **Result + Report to one .zip file**
- Upload (.zip) file on ETL
  - Submit one (.zip) file
    - zip file name : [Term0-final]name.zip (ex : [Term0-final]홍길동.zip)
  - Due: 6/19(SAT) 23:59
    - **No Late Submission**