LeNet-5 on FPGA

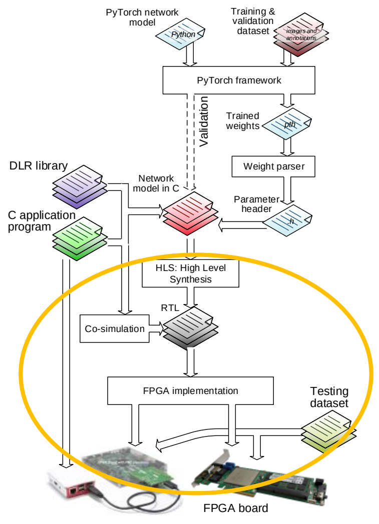
- Getting RTL, implementing on FPGA, running on FPGA -

2021

Ando Ki, Ph.D.
adki@future-ds.com

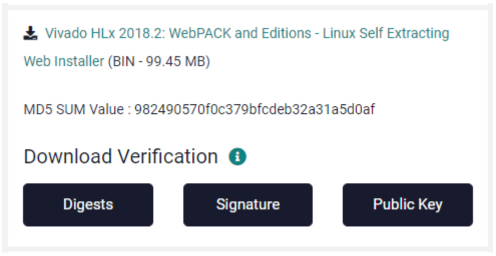
Table of contents

- 필요한 패키지들
- How to get the project
- directory structure
- HLS
- LeNet-5 HW block
- Internal register map
- Implementation
- Block diagram
- Vivado block design
- Programming FPGA and running LeNet-5
- HW setup
- main.cpp
- LeNet-5 handling routine
- Running LeNet-5 on FPGA



필요한 패키지들 (1/2)

- Xilinx Vivado 2018.2 Webpack
  - 반드시 Webpack을 사용해야 함 (라이센스 없이 사용 가능)
  - https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html


Vivado HLx 2018.2: WebPACK and Editions - Linux Self Extracting Web Installer (BIN - 99.45 MB)

MD5 SUM Value : 982490570f0c379bfcdeb32a31a5d0af

Download Verification

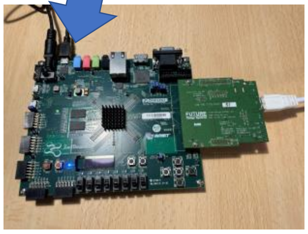
- Avnet ZedBoard File
  - refer to 'http://zedboard.org/sites/default/files/documentations/Installing-Board-Definition-Files.pdf'
  - check $XILINX_VIVADO/data/boards/board_files' directory.
    - There should be found 'zed' directory.

필요한 패키지들 (2/2)

- CON-FMC package
  - https://github.com/github-fds/confmc.x86_64.linux.2020.06
  - 위 GitHub 사이트의 'doc' 디렉토리 아래 'User Manual' 문서 '3. Software installation' 참고.

Xilinx JTAG USB driver

- Run ‘install_drivers’
  - $ cd /tools/Xilinx/Vivado/2018.3/data/xicom/cable_drivers/lin64/install_script/install_drivers
  - $ sudo ./install_drivers
- Unplug JTAG-USB cable from the board and plug again
- Check usb
  - $ lsusb
    - Bus 001 Device 004: ID **0403:6014 Future Technology Devices International** d FT232H Single HS USB-UART/FIFO IC



Make USB port available for ordinary user

- Run setup script
  - $ source /opt/confmc/2020.06/settings.sh
- Install required User-Level USB driver
  - $ sudo apt install libusb-1.0.0-dev
- Update udev without reboot
  - $ sudo udevadm control --reload-rules
  - $ sudo udevadm trigger

```
[socmgr@soc3w19] make run
./lenet ./images/0.png
libusb: error [_get_usbfs_fd] libusb couldn't open USB device /dev/bus/usb/001/005: Permission denied
libusb: error [_get_usbfs_fd] libusb requires write access to USB device nodes.
cannot initialize CON-FMC
./lenet ./images/1.png
libusb: error [_get_usbfs_fd] libusb couldn't open USB device /dev/bus/usb/001/005: Permission denied
libusb: error [_get_usbfs_fd] libusb requires write access to USB device nodes.
cannot initialize CON-FMC
```

- Check usb after plugging the CON-FMC USB cable
  - $ lsusb
    - Bus 001 Device 005: ID **04b4:00f3 Cypress Semiconductor Corp**.

How to get the project

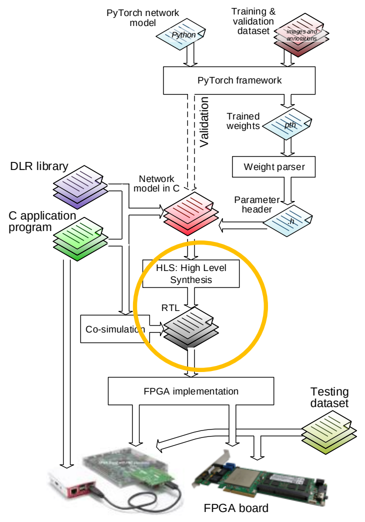
- Get a clone
  - ► $ git clone https://github.com/adki/DLR_Projects.git
- Go to ‘LeNet-5/LeNet-5.dlr.fpga’ directory

```
LeNet-5
  |-- LeNet-5.dlr
  |     +-- native.cpp
  |           +-- src
  |-- LeNet-5.dlr.fpga
  |   |-- hw
  |   |   |-- hls
  |   |   |     +-- tcl.float
  |   |   |-- impl
  |   |   |       +-- vivado.zed.confmc.float
  |   |   |              +-- xdc
  |   |   +-- iplib
  |   |          +-- bfm_axi
  |   +-- sw.native
  |          +-- lenet.confmc
  |                 +-- src
  |-- LeNet-5.pytorch
  |   |-- samples
  |   +-- src
  +-- LeNet-5.pytorch.dlr
         +-- src
```

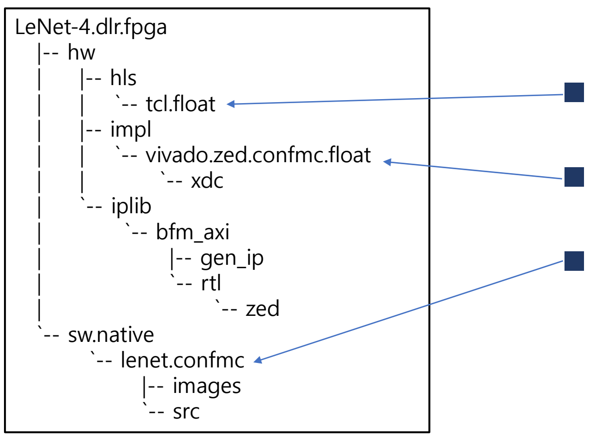
Table of contents

- 필요한 패키지들
- How to get the project
- directory structure
- **HLS**
- LeNet-5 HW block
- Internal register map
- Implementation
- Block diagram
- Vivado block design
- Programming FPGA and running LeNet-5
- HW setup
- main.cpp
- LeNet-5 handling routine
- Running LeNet-5 on FPGA

```
LeNet-5
  |-- LeNet-5.dlr
  |     \- native.cpp
  |
  |-- LeNet-5.dlr.fpga
  |
  |-- LeNet-5.pytorch
  |
  +-- LeNet-5.pytorch.dlr
```



Directory structure



```
LeNet-4.dlr.fpga
   |-- hw
   |     |-- hls
   |     |     `-- tcl.float
   |     |-- impl
   |     |     `-- vivado.zed.confmc.float
   |     |           `-- xdc
   |     `-- iplib
   |           `-- bfm_axi
   |                 |-- gen_ip
   |                 `-- rtl
   |                       `-- zed
   `-- sw.native
         `-- lenet.confmc
               |-- images
               `-- src
```

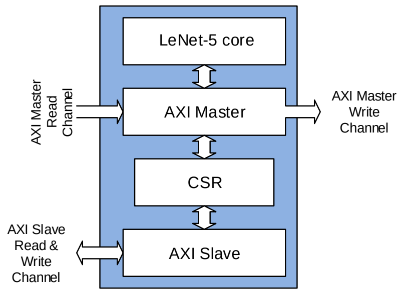
- It uses ‘lenet5.cpp’ in the following directory.
  - $PROJECT/LeNet-5/LeNet-5.dlr/native.cpp/src
- High level synthesis
- FPGA implementation
- Running with C and FPGA

HLS

- $ cd $PROJECT/LeNet-5/LeNet-5.dlr.fpga/hw/hls/tcl.float
- $ source $XILINX_VIVADO/settings64.sh
- $ make

LeNet-5 HW block

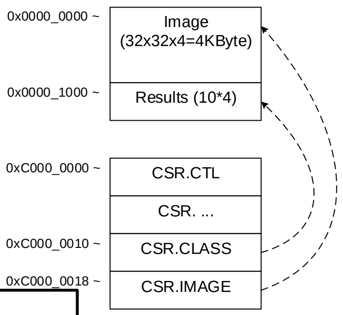
```
module lenet5 (
          input       ap_clk
        , input       ap_rst_n
        , output      m_axi_memory_bus_AWVALID
        , input       m_axi_memory_bus_AWREADY
        , output      m_axi_memory_bus_AWADDR
        , output      m_axi_memory_bus_AWID
        , output      m_axi_memory_bus_AWLEN
        , output      m_axi_memory_bus_AWSIZE
        , output      m_axi_memory_bus_AWBURST
        , output      m_axi_memory_bus_AWLOCK
        , output      m_axi_memory_bus_AWCACHE
        , output      m_axi_memory_bus_AWPROT
        , output      m_axi_memory_bus_AWQOS
        , output      m_axi_memory_bus_AWREGION
        , output      m_axi_memory_bus_AWUSER
        , output      m_axi_memory_bus_WVALID
        , input       m_axi_memory_bus_WREADY
        , output      m_axi_memory_bus_WDATA
        , output      m_axi_memory_bus_WSTRB
        , output      m_axi_memory_bus_WLAST
        , output      m_axi_memory_bus_WID
        , output      m_axi_memory_bus_WUSER
        , output      m_axi_memory_bus_ARVALID
        , input       m_axi_memory_bus_ARREADY
        , output      m_axi_memory_bus_ARADDR
        , output      m_axi_memory_bus_ARID
        , output      m_axi_memory_bus_ARLEN
        , output      m_axi_memory_bus_ARSIZE
        , output      m_axi_memory_bus_ARBURST
        , output      m_axi_memory_bus_ARLOCK
        , output      m_axi_memory_bus_ARCACHE
        , output      m_axi_memory_bus_ARPROT
        , output      m_axi_memory_bus_ARQOS
        , output      m_axi_memory_bus_ARREGION
        , output      m_axi_memory_bus_ARUSER
        , input       m_axi_memory_bus_RVALID
        , output      m_axi_memory_bus_RREADY
        , input       m_axi_memory_bus_RDATA
        , input       m_axi_memory_bus_RLAST
        , input       m_axi_memory_bus_RID
        , input       m_axi_memory_bus_RUSER
        , input       m_axi_memory_bus_RRESP
        , input       m_axi_memory_bus_BVALID
        , output      m_axi_memory_bus_BREADY
        , input       m_axi_memory_bus_BRESP
        , input       m_axi_memory_bus_BID
        , input       m_axi_memory_bus_BUSER
        , output      s_axi_axilite_AWVALID
        , input       s_axi_axilite_AWREADY
        , output      s_axi_axilite_AWADDR
        , output      s_axi_axilite_WVALID
        , output      s_axi_axilite_WREADY
        , output      s_axi_axilite_WDATA
        , output      s_axi_axilite_WSTRB
        , output      s_axi_axilite_ARVALID
        , input       s_axi_axilite_ARREADY
        , output      s_axi_axilite_ARADDR
        , input       s_axi_axilite_RVALID
        , output      s_axi_axilite_RREADY
        , input       s_axi_axilite_RDATA
        , input       s_axi_axilite_RRESP
        , input       s_axi_axilite_BVALID
        , output      s_axi_axilite_BREADY
        , input       s_axi_axilite_BRESP
        , output      interrupt
);

endmodule
```



Internal register map

```
// 0x00 : Control signals
//        bit 0  - ap_start (Read/Write/COH)
//        bit 1  - ap_done (Read/COR)
//        bit 2  - ap_idle (Read)
//        bit 3  - ap_ready (Read)
//        bit 7  - auto_restart (Read/Write)
//        others - reserved
// 0x04 : Global Interrupt Enable Register
//        bit 0  - Global Interrupt Enable (Read/Write)
//        others - reserved
// 0x08 : IP Interrupt Enable Register (Read/Write)
//        bit 0  - Channel 0 (ap_done)
//        bit 1  - Channel 1 (ap_ready)
//        others - reserved
// 0x0c : IP Interrupt Status Register (Read/TOW)
//        bit 0  - Channel 0 (ap_done)
//        bit 1  - Channel 1 (ap_ready)
//        others - reserved
// 0x10 : Data signal of classes
//        bit 31~0 - classes[31:0] (Read/Write)
// 0x14 : reserved
// 0x18 : Data signal of image_r
//        bit 31~0 - image_r[31:0] (Read/Write)
```

```
// (SC = Self Clear, COR = Clear on Read, TOW = Toggle on Write, COH = Clear on Handshake)
```

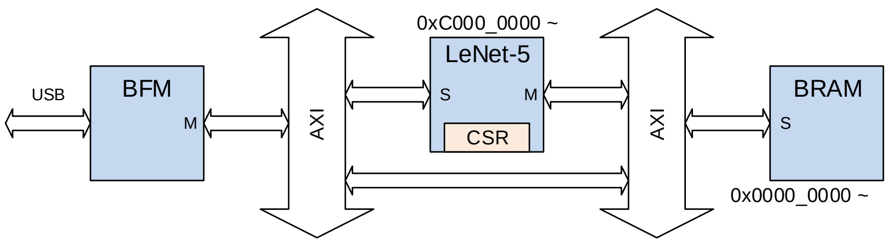


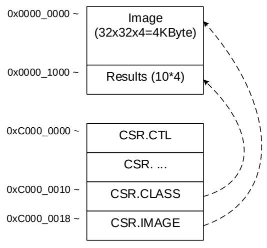
```
#define XLENET5_CTL_ADDR_AP_CTRL       0x00
#define XLENET5_CTL_ADDR_GIE           0x04
#define XLENET5_CTL_ADDR_IER           0x08
#define XLENET5_CTL_ADDR_ISR           0x0c
#define XLENET5_CTL_ADDR_CLASSES_DATA  0x10
#define XLENET5_CTL_BITS_CLASSES_DATA  32
#define XLENET5_CTL_ADDR_IMAGE_R_DATA  0x18
#define XLENET5_CTL_BITS_IMAGE_R_DATA  32
```

Implementation

- $ cd $PROJECT/LeNevi t-5/LeNet-5.dlr.fpga/hw/impl/vivado.zed.confmc.float
- $ source $XILINX_VIVADO/settings64.sh
- $ make

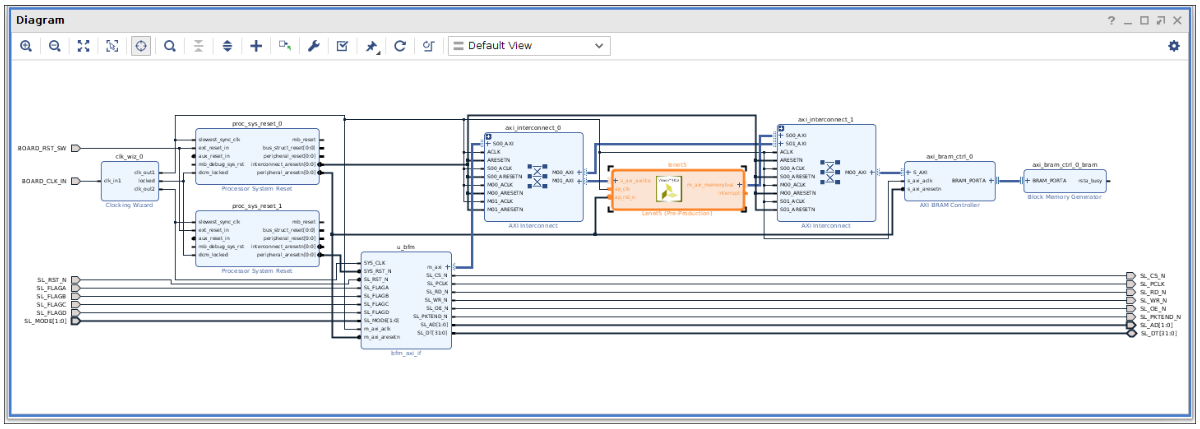
Block diagram





- 0x0000_0000 ~ : BRAM memory to store input image and output results
- 0xC000_0000 ~ : LeNet-5 internal registers
    - LeNet-5 internal registers locate in this address space.
    - Refer to CSR address map.
    - 0xC000_0010: should be written 0x0000_1000. (at least a size of image frame apart from 0x0000_0000)
    - 0xC000_0018: should be written 0x0000_0000.

Vivado block design



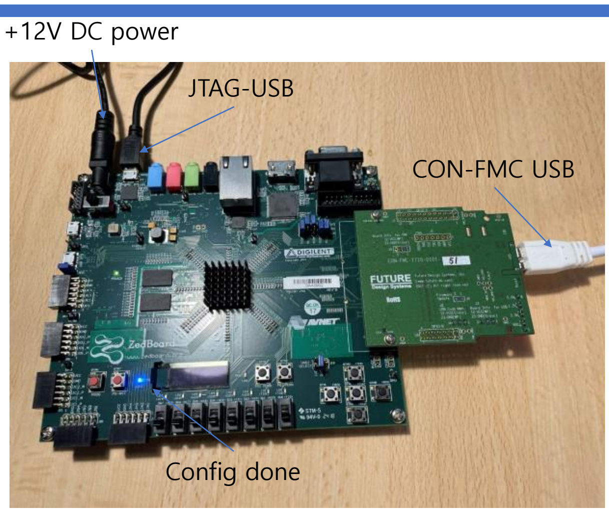
Programming FPGA and running LeNet-5

- Programming FPGA
  - $ source $XILINX_VIVADO/settings64.sh
  - $ vivado

bitstream file at $PROJECT/LeNevi t-5/LeNet-5.dlr.fpga/hw/impl/vivado.zed.confmc.float zed_example/zed_example.runs/impl_1/zed_bd_wrapper.bit

- Running LeNet-5
  - $ cd $PROJECT/LeNevi t-5/LeNet-5.dlr.fpga/sw.native/lenet.confmc
  - $ source /opt/confmc/2020.06/setup.sh
  - $ make
  - $ make run

HW setup



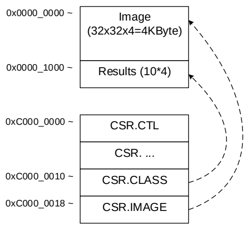
- 1. Turn off ZedBoard
- 2. Check configuration jumps
  - (MIO2 should be GND)
- 3. Connect JTAG-USB port
- 4. Connect CON-FMC USB port
- 5. Turn on ZedBoard
- 6. Download bi-stream using Vivado Hardware Manager

You should see followings on you host computer.

```
$ lsusb -d 04b4:
Bus 001 Device 017: ID 04b4:00f3 Cypress Semiconductor Corp.
```

main.cpp

```
#define NUM_ROWS     32 // IMG_DMNIN
#define NUM_COLS     32 // IMG_DMNIN
#define SIZE_IMG     (NUM_ROWS*NUM_COLS) // num of pixels
#define NUM_CLASSES  10 // SFMX_SIZE
#define ADDR_CSR     0xC0000000
#define ADDR_IMG     0x00000000
#define ADDR_RESULT  (ADDR_IMG+SIZE_IMG*NBYTES_OF_DTYPE)

#define ADDR_CSR_AP_CTRL       (ADDR_CSR+0x00)
#define ADDR_CSR_GIE           (ADDR_CSR+0x04)
#define ADDR_CSR_IER           (ADDR_CSR+0x08)
#define ADDR_CSR_ISR           (ADDR_CSR+0x0c)
#define ADDR_CSR_AP_RETURN     (ADDR_CSR+0x10)
#define ADDR_CSR_CLASSES_DATA  (ADDR_CSR+0x10)
#define ADDR_CSR_IMAGE_R_DATA  (ADDR_CSR+0x18)
```



```
int main(int argc, char *argv[]) {
    handle=conInit(card_id, CON_MODE_CMD, CONAPI_LOG_LEVEL_INFO);
    unsigned int dataW, dataR;
    dataW = ADDR_IMG;
    MEM_WRITE(ADDR_CSR_IMAGE_R_DATA   , dataW);
    dataW = ADDR_RESULT;
    MEM_WRITE(ADDR_CSR_CLASSES_DATA, dataW);
    (void)lenet(argv[1]);
    return 0;
}
```

LeNet-5 handling routine

```
int lenet(char inputFileName[]) {
    float greyDataFloat[SIZE_IMG]; // note that it carries float [0~1]
    (void)get_image_data(inputFileName, greyDataFloat);

    WAIT_FOR_READY

    MEM_WRITE_G(ADDR_IMG, (unsigned int*)&greyDataFloat[0], NBYTES_OF_DTYPE, SIZE_IMG);

    GO_AND_WAIT_COMPLETE

    float resultFloat[NUM_CLASSES];
    MEM_READ_G(ADDR_RESULT, (unsigned int*)&resultFloat[0], NBYTES_OF_DTYPE, NUM_CLASSES);

    float resultClasses[NUM_CLASSES];
    softmax(resultClasses, resultFloat);

    return 0;
}
```

Get image

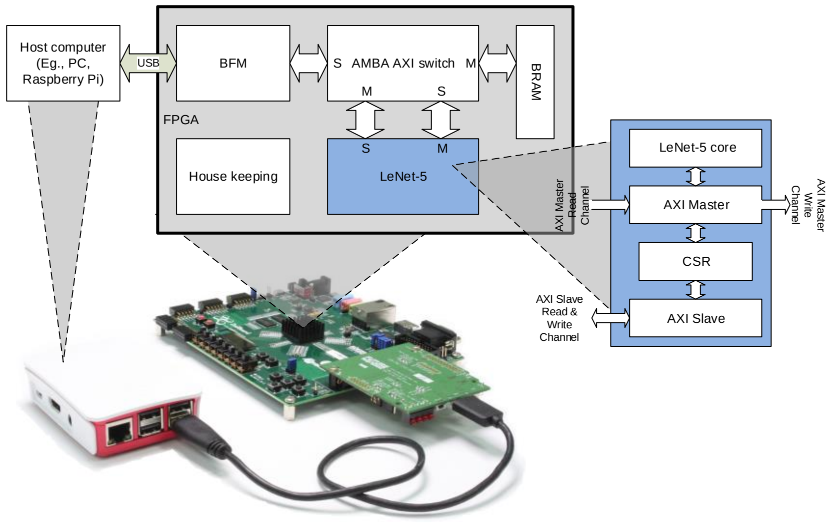
Wait until LeNet-5 IP is ready

Put image data in to BRAM

Wait for completion

Get results

Running LeNet-5 on FPGA


Host computer (Eg., PC, Raspberry Pi)
USB
BFM
S AMBA AXI switch M
M
S
BRAM
FPGA
S
M
House keeping
LeNet-5
LeNet-5 core
AXI Master
CSR
AXI Slave
AXI Master Read Channel
AXI Master Write Channel
AXI Slave Read & Write Channel