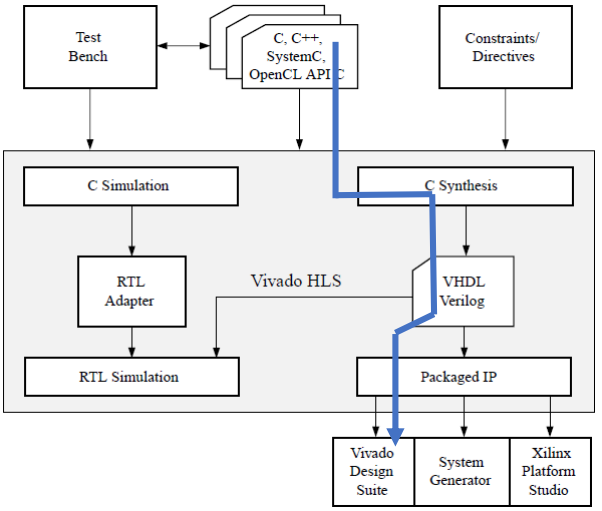
Deep Learning with FPGA
- LeNet-5 floating-point & Fixed-point -

2020

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

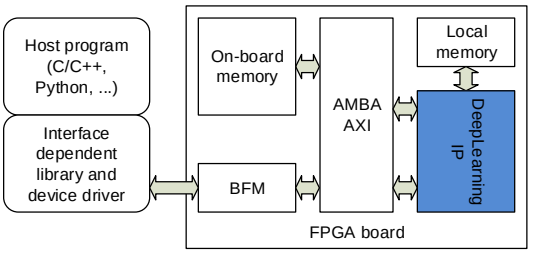
Contents

Xilinx HLS design flow

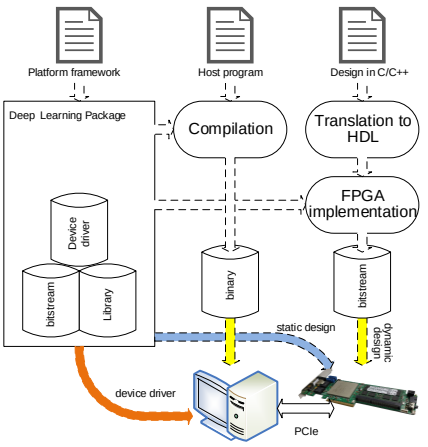


Straight forward solution

- Generic platform

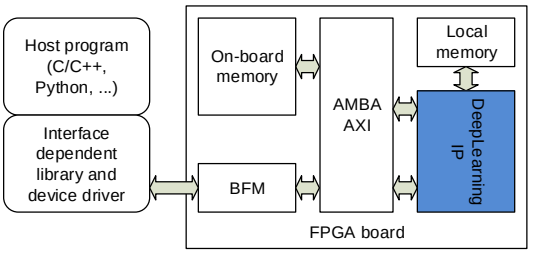


- Develop framework

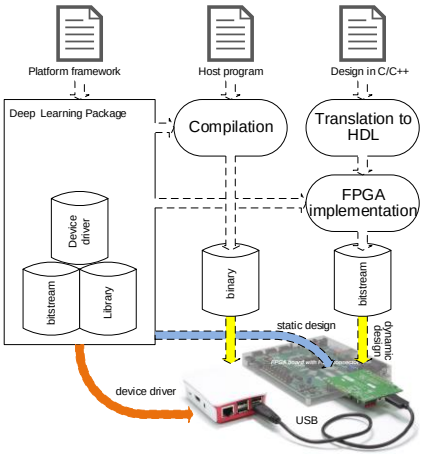


Straight forward solution

- Generic platform



- Develop framework



How to run in short (1/3)

- Prerequisites
  - Xilinx Vivado 2018.3
  - Xilinx Vivado HLS 2018.3
  - Xilinx SDK 2018.3
  - Xilinx PlatformUSB device driver installed (optional)
  - LibUSB-1.0.0 installed
  - Future Design Systems CON-FMC 2019.10
  - OpenCV 2

How to run in short (2/3)

$PROJECT/codes.fpga/LeNet_fpag

- 1. HLS Synthesis
- - go to 'hw/hls/tcl'
- - run 'make'

- 2. VIVADO IP Integrator
- - go to 'hw/impl/vivado.zed.confmc'
- - run 'make'

- 3. Boot file generation
- - go to 'hw/impl/vivado.zed.confmc/bootgen'
- - run 'make'
- - copy 'BOOT.bin' to the SD-CARD and then tun on ZedBoard

- Use Vivado 2018.3.
- Do not forget to set Vivado environment

  $ source /opt/Xilinx/Vivado/2018.3/settings64.sh

- Do not forget to set Vivado-SDK environment

  $ source /opt/Xilinx/SDK/2018.3/settings64.sh

How to run in short (3/3)

- 4. CON-FMC software (OpenCV is required)
- - It requires OpenCV and CON-FMC
- - go to 'sw.native/lenet.confmc' directory
- - run 'make clean; make'

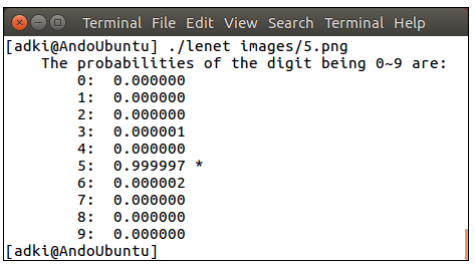
- 5. Run
- - go to 'sw.native/lenet.confmc' directory
- - run './lenet images/5.png‘

- ***Also Python-driven is available.***

- Do not forget to set CON-FMC environment

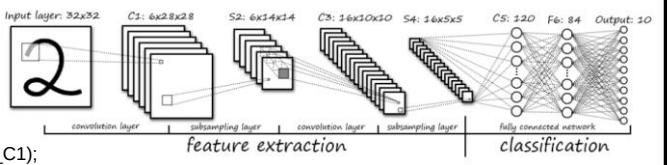
  $ source /opt/confmc/2019.10/settings.sh

- Do not forget to connect ZedBoard to the computer through USB
- Do not forget to turn on ZedBoard along with SD-Card containing proper BOOT.bin



LetNet-5 core in C++

```
main() @ tb/conv_net_tb.cpp
  |
  +-- lenet_wrapper() @ src/conv_net.cpp
        |
        +-- predict() @ src/conv_net.cpp
              |
              +-- convolution_c1(nml, weights_C1, layer1_out, biases_C1);
              +-- relu_a1(layer1_out, layer2_out);
              +-- pooling_p1(layer2_out, layer3_out);
              |
              +-- convolution_c2(layer3_out, weights_C2, layer4_out, biases_C2);
              +-- relu_a2(layer4_out, layer5_out);
              +-- pooling_p2(layer5_out, layer6_out);
              |
              +-- flatten(layer6_out, layer7_out);
              +-- vec_mat_mul_f1(layer7_out, weights_F1, biases_F1, layer8_out);
              +-- relu_a3(layer8_out, layer9_out);
              |
              +-- vec_mat_mul_f2(layer9_out, weights_F2, biases_F2, layer10_out);
              +-- relu_a4(layer10_out, layer11_out);
              |
              +-- vec_mat_mul_f3(layer11_out, weights_F3, biases_F3, layer12_out);
              +-- softmax(layer12_out, p);
```

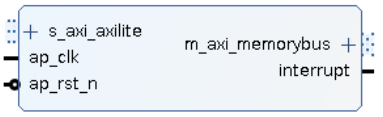


Use 'ReLU' instead of 'tanh'.

LeNet-5 RTL: top

```
void lenet_wrapper(float *shared_mem) {
#pragma HLS INTERFACE m_axi depth = 1034 \
                  port = shared_mem \
                  offset = slave \
                  bundle = memorybus register
#pragma HLS INTERFACE s_axilite port = return \
                  bundle = axilite register
    DTYPE img[IMG_CHANNELS][IMG_DMNIN][IMG_DMNIN];
    DTYPE p[SFMX_SIZE];
    for (uint8_t l = 0; l < IMG_CHANNELS; l++) { // 1
      for (uint8_t r = 0; r < IMG_DMNIN; r++) { // 32
        for (uint8_t c = 0; c < IMG_DMNIN; c++) { // 3
          img[l][r][c] = (DTYPE)*shared_mem++;
        } } }
    predict(img,p);
    for (uint8_t i = 0; i < SFMX_SIZE; i++) { // 10
      *shared_mem++ = (float)p[i];
    }
}
```

$PROJECT/codes.fpga/LeNet_fpag/hls/src



input image: 32x32 (1024) ; result classes: 10

input image: 32x32 (1024)
typedef float DTYPE;

result classes: 10

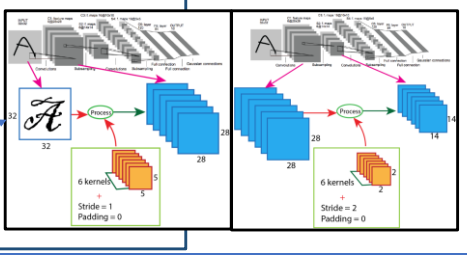
LeNet-5 inference engine

LeNet-5 RTL: predict (1/2)

```
void predict(DTYPE img[IMG_CHANNELS][IMG_DMNIN][IMG_DMNIN], DTYPE p[SFMX_SIZE]) {
    DTYPE nml[IMG_CHANNELS][IMG_DMNIN][IMG_DMNIN];
    DTYPE layer1_out[C1_N_FILTERS][C1_OUT_DMNIN][C1_OUT_DMNIN];
    DTYPE layer2_out[C1_N_FILTERS][A1_ROWS][A1_COLS];
    DTYPE layer3_out[C1_N_FILTERS][P1_DOWNSIZE][P1_DOWNSIZE];
    DTYPE layer4_out[C2_N_FILTERS][C2_OUT_DMNIN][C2_OUT_DMNIN];
    DTYPE layer5_out[C2_N_FILTERS][A2_ROWS][A2_COLS];
    DTYPE layer6_out[C2_N_FILTERS][P2_DOWNSIZE][P2_DOWNSIZE];
    DTYPE layer7_out[FLAT_VEC_SZ];
    DTYPE layer8_out[F1_ROWS];
    DTYPE layer9_out[F1_ROWS];
    DTYPE layer10_out[F2_ROWS];
    DTYPE layer11_out[F2_ROWS];
    DTYPE layer12_out[F3_ROWS];

    normalize(img, nml);

    convolution_c1(nml, weights_C1, layer1_out, biases_C1);
    relu_a1(layer1_out, layer2_out);
    pooling_p1(layer2_out, layer3_out);
```

normalize using mean and standard deviation

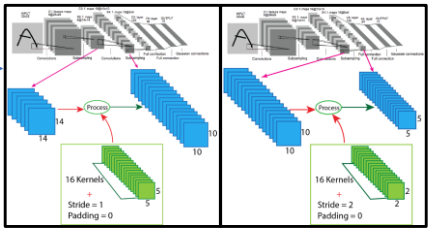
Use weight table

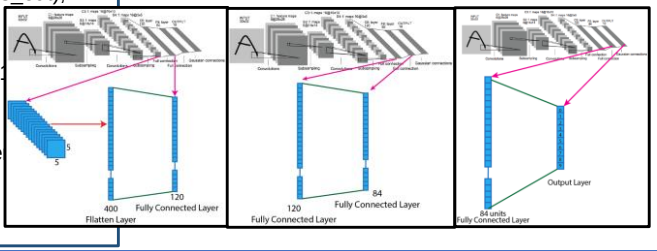


LeNet-5 RTL: predict (2/2)

```
    convolution_c2(layer3_out, weights_C2, layer4_out, biases_C2);
    relu_a2(layer4_out, layer5_out);
    pooling_p2(layer5_out, layer6_out);

    flatten(layer6_out, layer7_out);

    vec_mat_mul_f1(layer7_out, weights_F1, biases_F1, layer8_out);
    relu_a3(layer8_out, layer9_out);

    vec_mat_mul_f2(layer9_out, weights_F2, biases_F2, layer
    relu_a4(layer10_out, layer11_out);

    vec_mat_mul_f3(layer11_out, weights_F3, biases_F3, laye
    softmax(layer12_out, p);
}
```

Use weight table





LeNet-5 RTL: convolution

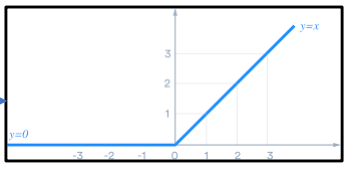
```
void convolution_c1 (
        DTYPE   X[C1_N_CHAN][C1_X_DMNIN][C1_X_DMNIN],
const DTYPE   W[C1_N_CHAN][C1_N_FILTERS][C1_W_DMNIN][C1_W_DMNIN],
        DTYPE out[C1_N_FILTERS][C1_OUT_DMNIN][C1_OUT_DMNIN],
const DTYPE bias[C1_N_FILTERS]) {
   uint8_t ch, f, i, j, r, c;
   for (f = 0; f < C1_N_FILTERS; ++f)  { // out put will be the number of filters
      for (r = 0; r < C1_OUT_DMNIN; ++r)  for (c = 0; c < C1_OUT_DMNIN; ++c) out[f][r][c] = bias[f]; // bias initialization
      for (ch = 0; ch < C1_N_CHAN; ++ch) {
         for (r = 0; r < C1_X_DMNIN - C1_W_DMNIN + 1; r += STRIDE) { // convolution
            for (c = 0, i = 0, j = 0; c < C1_X_DMNIN - C1_W_DMNIN + 1; c += STRIDE) {
               #pragma HLS PIPELINE
               for (i = 0; i < C1_W_DMNIN; ++i) {
                  for (j = 0; j < C1_W_DMNIN; ++j) {
                     out[f][r][c] += X[ch][r + i][j + c] * W[ch][f][i][j];
                  } }  }
         } // for (r
      } // for (ch
   } // for (f
}
```

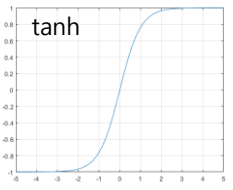
Use weight table

LeNet-5 RTL: ReLU

```
void relu_a1(DTYPE in[C1_N_FILTERS][A1_ROWS][A1_COLS], DTYPE out[C1_N_FILTERS][A1_ROWS][A1_COLS]) {

   uint16_t r;
   uint8_t c, m;

   for (m = 0; m < C1_N_FILTERS; ++m) {
      for (r = 0; r < A1_ROWS; ++r) {
         for (c = 0; c < A1_COLS; ++c) {
         #pragma HLS PIPELINE
            out[m][r][c] = (in[m][r][c] > 0) ? (in[m][r][c]) : 0;
         }
      }
   }
}
```

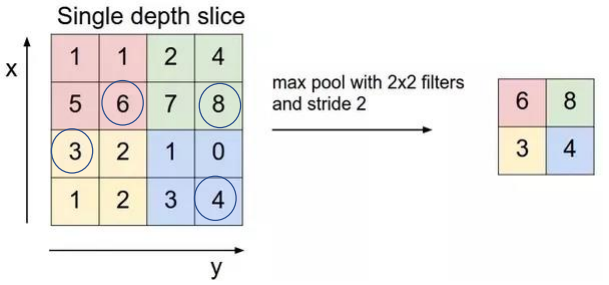
ReLU: rectified linear unit





LeNet-5 RTL: MAX Pooling

```
DTYPE maxFour(DTYPE a, DTYPE b, DTYPE c, DTYPE d) {
   return max(max(a,b), max(c,d));
}
void pooling_p1(DTYPE in[C1_N_FILTERS][P1_SIZE][P1_SIZE], // 1x28x28
                DTYPE out[C1_N_FILTERS][P1_DOWNSIZE][P1_DOWNSIZE]) { // 1x14x14
   uint8_t i, j, m;
   for (m = 0; m < C1_N_FILTERS; ++m) {
      for (i = 0; i < P1_DOWNSIZE; i++) { // 14
         for (j = 0; j < P1_DOWNSIZE; j++) { // 14
            out[m][i][j] = maxFour(
                  in[m][i << 1][j << 1],
                  in[m][(i << 1) + 1][j << 1],
                  in[m][i << 1][(j << 1) + 1],
                  in[m][(i << 1) + 1][(j << 1) + 1]
            );
         }
      }
   }
}
```



LeNet-5 RTL: Flatten

```
void flatten(
      DTYPE IN[C2_N_FILTERS][P2_DOWNSIZE][P2_DOWNSIZE],
      DTYPE OUT[FLAT_VEC_SZ]) {

   uint8_t i,j,k;
   uint16_t t;

   for (i = 0, t = 0; i < C2_N_FILTERS; ++i) {
      for (j = 0; j < P2_DOWNSIZE; ++j) {
         for (k = 0; k < P2_DOWNSIZE; ++k) {
            OUT[t++] = IN[i][j][k]; // 2-Dimentional to 1-Dimentional
         }
      }
   }
}
```

LeNet-5 RTL: Fully connection with ReLU

```
void vec_mat_mul_f1( DTYPE X[FLAT_VEC_SZ], const DTYPE W[F1_ROWS][F1_COLS],
                     const DTYPE bias[F1_ROWS], DTYPE Z[F1_ROWS]) {
    uint16_t c;
    uint8_t r;
    for (r = 0; r < F1_ROWS; ++r) {
        Z[r] = bias[r];
        for (c = 0; c < F1_COLS; ++c) {
        #pragma HLS PIPELINE
            Z[r] += W[r][c] * X[c];
        }
    }
}

void relu_a3(DTYPE in[F1_ROWS], DTYPE out[F1_ROWS]) {
    uint8_t i;
    for (i = 0; i < F1_ROWS; ++i) {
    #pragma HLS PIPELINE
        out[i] = (in[i] > 0) ? (in[i]) : 0;
    }
}
```


LeNet-5 RTL: Fully connection with softmax

```
void vec_mat_mul_f3(
        DTYPE X[F2_ROWS],
    const DTYPE W[F3_ROWS][F3_COLS],
    const DTYPE bias[F3_ROWS],
        DTYPE Z[F3_ROWS]) {
#pragma HLS INLINE

    uint8_t r, c;

    for (r = 0; r < F3_ROWS; ++r) {
        Z[r] = bias[r];
        for (c = 0; c < F3_COLS; ++c) {
        #pragma HLS PIPELINE
            Z[r] += W[r][c] * X[c];
        }
    }
}
```

```
void softmax(DTYPE Z[SFMX_SIZE], DTYPE P[SFMX_SIZE]) {

    uint8_t i;
    uint16_t idx[SFMX_SIZE];

    DTYPE denom = 0;
    for (i = 0; i < SFMX_SIZE; ++i) {
        idx[i] = (SFMX_RES>>1) + (int)(Z[i] * 10);
        denom += expZ[idx[i]];
    }

    for (i = 0; i < SFMX_SIZE; ++i) {
        P[i] = expZ[idx[i]] / denom;
    }
}
```

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

Use table

C-driven host program (1/2)

```
#include <opencv2/opencv.hpp>
using namespace cv;

#include "conapi.h"
#include "trx_axi_api.h“

int main(int argc, char *argv[]) {
    handle=conInit(card_id, CON_MODE_CMD,
            CONAPI_LOG_LEVEL_INFO);
    (void)host(argv[1]);
    return 0;
}
```

```
int host(char inputFileName[]) {
    // 1. get image data (gray scale [0-1] with black background
    unsigned int greyData[SIZE_IMG]; // it contains floating-point bit-pattern
    (void)get_image_data(inputFileName, greyData);

    int ap_idle, ap_idle_r;
    int ap_done, ap_done_r;
    int ap_start, ap_data;
    unsigned int ap_addr;
    // 2. check IP is ready
    ap_addr = ADDR_CSR;
    while (1) {
        MEM_READ(ap_addr, ap_idle_r);
        ap_idle = (ap_idle_r >> 2) && 0x1;
        if (ap_idle)
            break;
    }
```

$PROJECT/codes.fpga/LeNet_fpag/sw.native/lenet.confmc/src/main.cpp

C-driven host program (2/2)

```
    // 3. write image data to the IP
    MEM_WRITE_G(ADDR_IMG, &greyData[0], 4,
                    SIZE_IMG)
    // 4. let IP go and wait for completion
    ap_addr = ADDR_CSR;
    ap_data = 0x1;
    MEM_WRITE(ap_addr, ap_data);//Start
    while (1) {
        MEM_READ(ap_addr, ap_done_r);
        ap_done = (ap_done_r >> 1) && 0x1;
        if (ap_done) break;
    }
    // 5. read results from the IP
    unsigned int resultClasses[10];
    MEM_READ_G(ADDR_RESULT, &resultClasses[0],
                    4, 10);
    // note that 'resultClasses' carries floating-point
    // contents
```

```
    // 6. print the results
    printf("    The probabilities of the digit being 0~9 are:\n");
    float maxVal=0.0;
    int   maxId=0;
    for (int i = 0; i < 10; i++) {
        // note that how to get floating point-contents
        float val = *(float*)&resultClasses[i];
        if (maxVal<val) {
            maxVal = val;
            maxId  = i;
        }
    }
    for (int i = 0; i < 10; i++) {
        float result = *(float*)&resultClasses[i];
        printf("        %d:  %f %s\n", i, result, (i==maxId) ? "*" : "");
    }

    return 0;
}
```

$PROJECT/codes.fpga/LeNet_fpag/sw.native/lenet.confmc/src/main.cpp

Prepare LeNet-5 IP

- 1. HLS Synthesis
- - go to 'hw/hls/tcl'
- - run 'make'

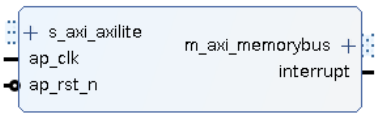
$PROJECT/codes.fpga/LeNet_fpag

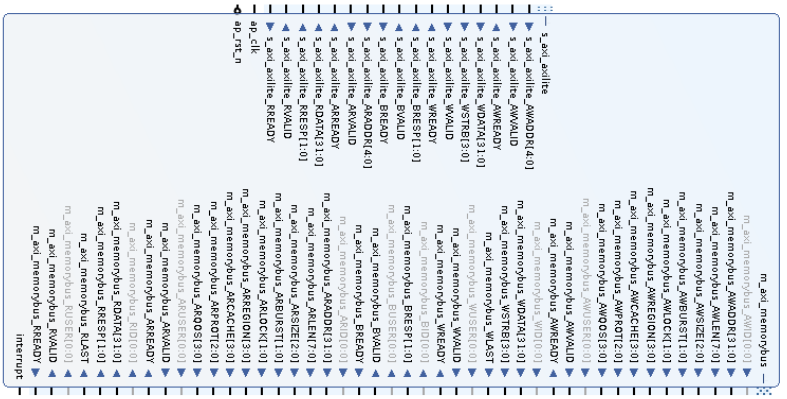
add following in '.bashrc' file.

```
alias set_vivado='source /opt/XilinxWebpack/Vivado/2018.3/settings64.sh;₩
                  export XILINX_VIVADO_HLS=/opt/XilinxWebpack/Vivado/2018.3;₩
                  export XILINX_SDK=/opt/Xilinx/SDK/2018.3;₩
                  source ${XILINX_SDK}/settings64.sh'
```

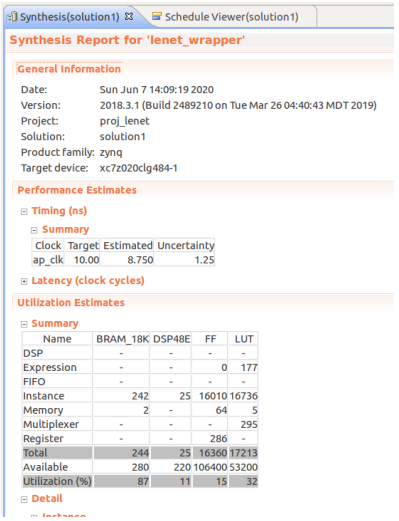
```
... ...
$ source /opt/XilinxWebpack/Vivado/2018.3/settings64.sh
$ export XILINX_VIVADO_HLS=/opt/XilinxWebpack/Vivado/2018.3
... ...
$ cd hw/hls/tcl
$ make
```


LeNet-5 core block





LeNet-5 synthesis report



Synthesis(solution1) | Schedule Viewer(solution1)

Synthesis Report for 'lenet_wrapper'

General Information

Date: Sun Jun 7 14:09:19 2020
Version: 2018.3.1 (Build 2489210 on Tue Mar 26 04:40:43 MDT 2019)
Project: proj_lenet
Solution: solution1
Product family: zynq
Target device: xc7z020clg484-1

Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.750	1.25

Latency (clock cycles)

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	177
FIFO	-	-	-	-
Instance	242	25	16010	16736
Memory	2	-	64	5
Multiplexer	-	-	-	295
Register	-	-	286	-
Total	244	25	16360	17213
Available	280	220	106400	53200
Utilization (%)	87	11	15	32

Detail

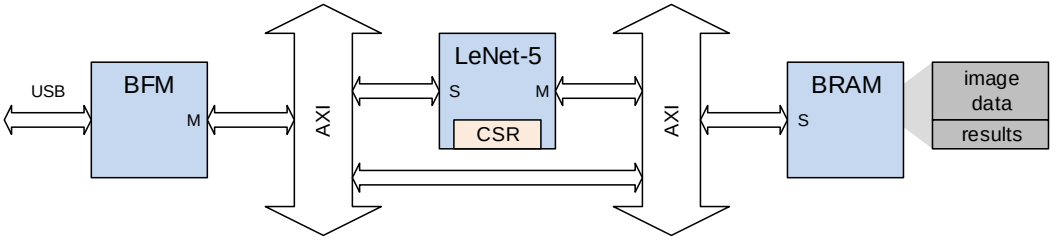
Prepare whole design

- 2. VIVADO IP Integrator
- - go to 'hw/impl/vivado.zed.confmc'
- - run 'make'

$PROJECT/codes.fpga/LeNet_fpag

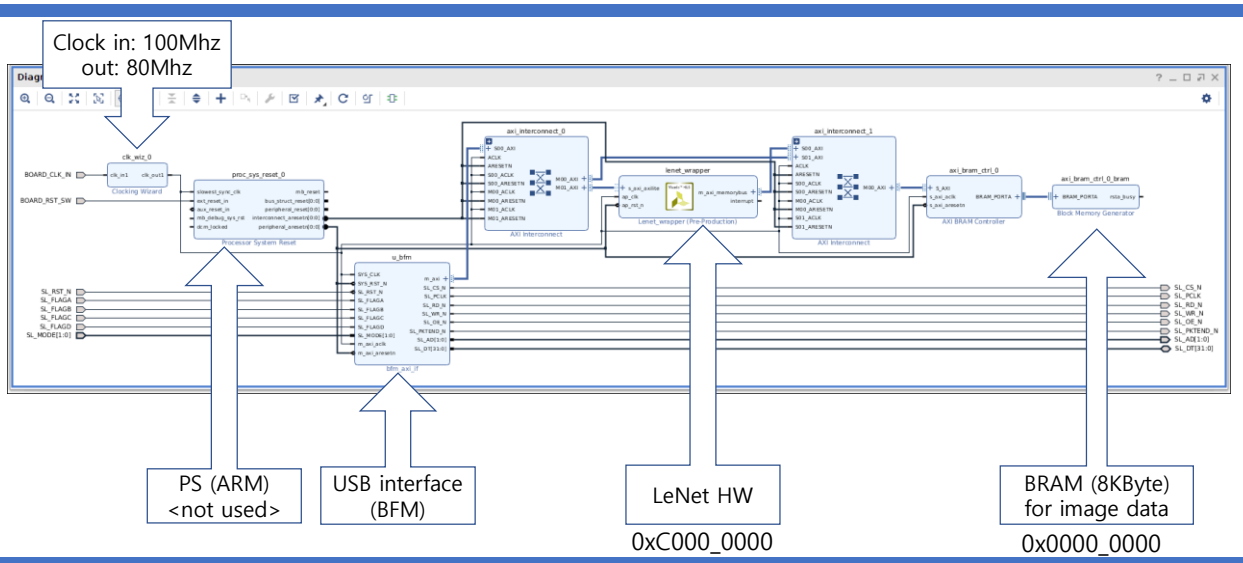
```
... ...
$ source /opt/XilinxWebpack/Vivado/2018.3/settings64.sh
... ...
$ cd hw/impl/vivado.zed.confmc
$ make
```

Block diagram

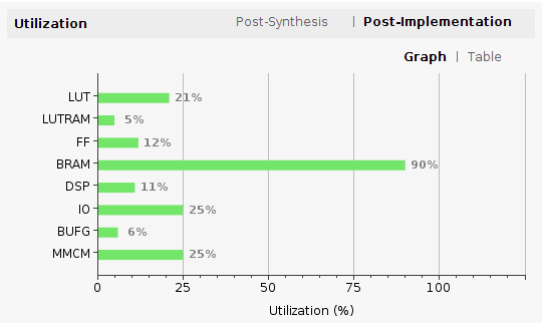


CSR: 0xC000_0000
Image Data: 0x0000_0000 ~ (1024*4-1)
Results: (1024*4)~(1034*4-1)

Block diagram



Implementation report



Timing	Setup | Hold | Pulse Width
Worst Negative Slack (WNS):	1.981 ns
Total Negative Slack (TNS):	0 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	33543

Implemented Timing Report

Power	Summary | On-Chip
Total On-Chip Power:	**0.479 W**
Junction Temperature:	**30.5 °C**
Thermal Margin:	54.5 °C (4.5 W)
Effective θJA:	11.5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Implemented Power Report

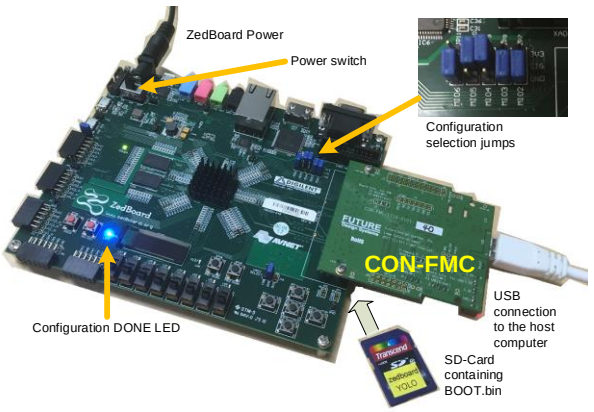
Prepare FPGA image

- 3. Boot file generation
- - go to 'hw/impl/vivado.zed.confmc/bootgen'
- - run 'make'
- - copy 'BOOT.bin' to the SD-CARD and then tun on ZedBoard

$PROJECT/codes.fpga/LeNet_fpag

```
... ...
$ export XILINX_SDK=/opt/Xilinx/SDK/2018.3
$ source ${XILINX_SDK}/settings64.sh
... ...
$ cd hw/impl/vivado.zed.confmc/bootgen
$ make
```

HW setup



- 1. Turn off ZedBoard
- 2. Check configuration jumps
- 3. Insert SD-Card
- 4. Connect USB port
- 5. Turn on ZedBoard

You should see followings on you host computer.

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

Running the host program through USB

- 4. CON-FMC software (OpenCV is required)
- - It requires OpenCV and CON-FMC
- - go to 'sw.native/lenet.confmc' directory
- - run 'make clean; make'
- 5. Run
- - go to 'sw.native/lenet.confmc' directory
- - run './lenet images/5.png'

```
Terminal File Edit View Search Terminal Help
[adki@AndoUbuntu] ./lenet images/5.png
    The probabilities of the digit being 0~9 are:
        0:  0.000000
        1:  0.000000
        2:  0.000000
        3:  0.000001
        4:  0.000000
        5:  0.999997 *
        6:  0.000002
        7:  0.000000
        8:  0.000000
        9:  0.000000
[adki@AndoUbuntu]
```

```
... ...
$ souce /opt/confmc/2919.10/sttings.sh
... ...
$ cd sw.native/lenet.confmc
$ make
$ ./lenet images/5.png
```

Python-driven host program

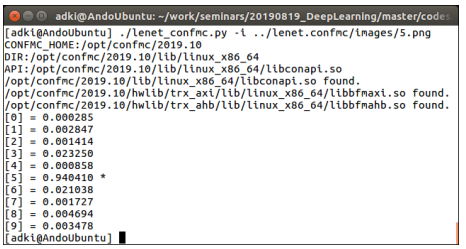
```
import sys
import cv2
import ctypes
import numpy as np
import confmc.pyconfmc as confmc
import confmc.pyconbfmaxi as axi
...
...

dep main(prog, argv):
   ....
    hdl=confmc.conInit()
    results = lenet_test(hdl, imageFile, verbose)
    confmc.conRelease(hdl)
```

```
def lenet_test(hdl, imageFile, verbose=0):
    # 1: get grey-scale image data in 1D array with black background
    fdata = get_image_data(imageFile, verbose=verbose)
    # 2: cast float to unsigned int (presever bit-pattern)
    udata = cast_float_to_uint(fdata)
    # 3: wait for IP ready
    wait_for_ready(hdl)
    # 4: push image data
    axi.BfmWrite(hdl, CONST_ADDR_IMG, udata, 4, CONST_SIZE_IMG)
    # 5: let IP run and wait for completion
    go_and_wait_for_done(hdl)
    # 6: get results
    uresults = [0]*CONST_NUM_CLASSES
    axi.BfmRead(hdl, CONST_ADDR_RESULT, uresults, 4,
                        CONST_NUM_CLASSES)
    # 7: cast unsigned int to float while keeping bit-pattern
    fresults = cast_uint_to_float(uresults)
    return fresults
```

$PROJECT/codes.fpga/LeNet_fpag/sw.native/lenet.confmc.python/lenet_confmc.py

Running Python program

- Make sure HW has been properly set up.
- 4. CON-FMC software (OpenCV is required)
- - It requires Python Version 2.x and CON-FMC
- - go to 'sw.native/lenet.confmc.python' directory
-
- 5. Run
- - run './lenet_confmc.py -i ../lenet.confmc/images/5.png'

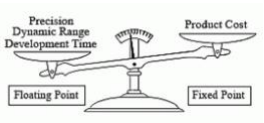



```
... ...
$ souce /opt/confmc/2919.10/sttings.sh
... ...
$ cd sw.native/lenet.confmc.python
$ ./lenet_confmc.py –i ../lenet.confmc/images/5.png
```

Contents

Fixed-point or floating point

- Floating point number
  - Slower
  - Accuracy varies
  - Represent very large number set
  - Radix point encoded
  - Complex logic required
  - When use floating point
    - Accuracy is important
    - Range of numbers unpredictable
    - Development time is short
- Fixed-point number
  - Very fast when based 2
  - No complicated logic
  - Radix point not encoded
  - Fixed accuracy
  - Can only represent small number set
  - When use fixed point
    - Low resources
    - Low gate delay
    - Simple implementation of HW components



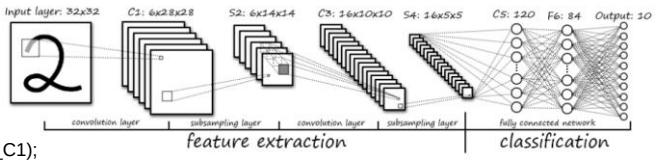
Fixed-point or floating point

- Floating point number
  - Slower
  - Accuracy varies
  - Represent very large number set
  - Radix point encoded
  - Complex logic required
  - When use floating point
    - Accuracy is important
    - Range of numbers unpredictable
    - Development time is short
- Fixed-point number
  - Very fast when based 2
  - No complicated logic
  - Radix point not encoded
  - Fixed accuracy
  - Can only represent small number set
  - When use fixed point
    - Low resources
    - Low gate delay
    - Simple implementation of HW components

Fixed Point	Floating point
Limited Dynamic range	Large Dynamic Range
Overview flow and quantization errors must be resolved	Easier to program since no scaling is required
Long product development time	Quick time to market
Cheaper	More expensive
Lower power consumption	High Power consumption

LetNet-5 core in C++

```
main() @ tb/conv_net_tb.cpp
  |
  +-- lenet_wrapper() @ src/conv_net.cpp
        |
        +-- predict() @ src/conv_net.cpp
              |
              +-- convolution_c1(nml, weights_C1, layer1_out, biases_C1);
              +-- relu_a1(layer1_out, layer2_out);
              +-- pooling_p1(layer2_out, layer3_out);
              |
              +-- convolution_c2(layer3_out, weights_C2, layer4_out, biases_C2);
              +-- relu_a2(layer4_out, layer5_out);
              +-- pooling_p2(layer5_out, layer6_out);
              |
              +-- flatten(layer6_out, layer7_out);
              +-- vec_mat_mul_f1(layer7_out, weights_F1, biases_F1, layer8_out);
              +-- relu_a3(layer8_out, layer9_out);
              |
              +-- vec_mat_mul_f2(layer9_out, weights_F2, biases_F2, layer10_out);
              +-- relu_a4(layer10_out, layer11_out);
              |
              +-- vec_mat_mul_f3(layer11_out, weights_F3, biases_F3, layer12_out);
```



Softmax is not implemented, but done by software, since it requires large number that is not suitable for fixed-point.

Use 'ReLU' instead of 'tanh'.

LeNet-5 RTL: header (cnn_net.h)

```
#ifndef CONV_NET_H
#define CONV_NET_H

#define NORMALIZED  1
#define FIXED_POINT 1

#include <stdint.h>

#if defined(FIXED_POINT)
#include "ap_fixed.h"
//#define DTYPE ap_fixed<32,8>
#else
typedef float DTYPE;
#endif
```

$PROJECT/codes.fpga/LeNet_fpag/hls/src.fixed

Fixed-point library
/opt/Xilinx/Vivado/2018.3/include

#define DTYPE ap_fixed<32,8>
➔ 8-bit out of 32-bit is integer

LeNet-5 RTL: top (conv_net.cpp)

```
void lenet_wrapper(DTYPEt *shared_mem) {
#pragma HLS INTERFACE m_axi depth = 1034 \
                  port = shared_mem \
                  offset = slave \
                  bundle = memorybus register
#pragma HLS INTERFACE s_axilite port = return \
                  bundle = axilite register
    DTYPE img[IMG_CHANNELS][IMG_DMNIN][IMG_DMNIN];
    DTYPE p[SFMX_SIZE];
    for (uint8_t l = 0; l < IMG_CHANNELS; l++) { // 1
      for (uint8_t r = 0; r < IMG_DMNIN; r++) { // 32
        for (uint8_t c = 0; c < IMG_DMNIN; c++) { // 3
          img[l][r][c] = (DTYPE)*shared_mem++;
        } } }
    predict(img,p);
    for (uint8_t i = 0; i < SFMX_SIZE; i++) { // 10
      *shared_mem++ = (float)p[i];
    }
}
```

$PROJECT/codes.fpga/LeNet_fpag/hls/src.fixed

#define DTYPE ap_fixed<32,8>
➔ 8-bit out of 32-bit is integer

input image: 32x32 (1024) ; result classes: 10

input image: 32x32 (1024)

result classes: 10

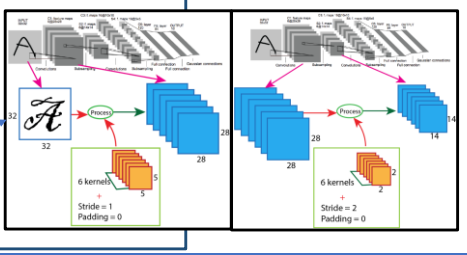
LeNet-5 inference engine

LeNet-5 RTL: predict (1/2)

```
void predict(DTYPE img[IMG_CHANNELS][IMG_DMNIN][IMG_DMNIN], DTYPE p[SFMX_SIZE]) {
    DTYPE nml[IMG_CHANNELS][IMG_DMNIN][IMG_DMNIN];
    DTYPE layer1_out[C1_N_FILTERS][C1_OUT_DMNIN][C1_OUT_DMNIN];
    DTYPE layer2_out[C1_N_FILTERS][A1_ROWS][A1_COLS];
    DTYPE layer3_out[C1_N_FILTERS][P1_DOWNSIZE][P1_DOWNSIZE];
    DTYPE layer4_out[C2_N_FILTERS][C2_OUT_DMNIN][C2_OUT_DMNIN];
    DTYPE layer5_out[C2_N_FILTERS][A2_ROWS][A2_COLS];
    DTYPE layer6_out[C2_N_FILTERS][P2_DOWNSIZE][P2_DOWNSIZE];
    DTYPE layer7_out[FLAT_VEC_SZ];
    DTYPE layer8_out[F1_ROWS];
    DTYPE layer9_out[F1_ROWS];
    DTYPE layer10_out[F2_ROWS];
    DTYPE layer11_out[F2_ROWS];
    DTYPE layer12_out[F3_ROWS];

    normalize(img, nml);

    convolution_c1(nml, weights_C1, layer1_out, biases_C1);
    relu_a1(layer1_out, layer2_out);
    pooling_p1(layer2_out, layer3_out);
```

normalize using mean and standard deviation

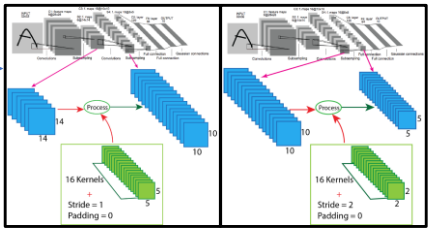
Use weight table

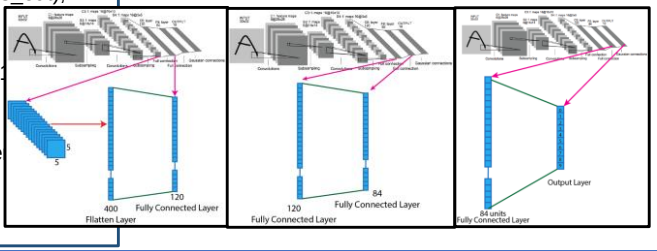


LeNet-5 RTL: predict (2/2)

```
    convolution_c2(layer3_out, weights_C2, layer4_out, biases_C2);
    relu_a2(layer4_out, layer5_out);
    pooling_p2(layer5_out, layer6_out);

    flatten(layer6_out, layer7_out);

    vec_mat_mul_f1(layer7_out, weights_F1, biases_F1, layer8_out);
    relu_a3(layer8_out, layer9_out);

    vec_mat_mul_f2(layer9_out, weights_F2, biases_F2, layer
    relu_a4(layer10_out, layer11_out);

    vec_mat_mul_f3(layer11_out, weights_F3, biases_F3, laye
    // softmax(layer12_out, p); excluded
}
```

Use weight table





C-driven host program (1/2)

```
#include <opencv2/opencv.hpp>
using namespace cv;

#if defined(FIXED_POINT)
#include "ap_fixed.h"
#define DTYPE ap_fixed<32,8>
#endif

#include "conapi.h"
#include "trx_axi_api.h“

int main(int argc, char *argv[]) {
    handle=conInit(card_id, CON_MODE_CMD,
            CONAPI_LOG_LEVEL_INFO);
    (void)host(argv[1]);
    return 0;
}
```

```
int host(char inputFileName[]) {
    // 1. get image data (gray scale [0-1]
    unsigned int greyData[SIZE_IMG]; // it contains floating-point bit-pattern
    (void)get_image_data(inputFileName, greyData);
#if defined(FIXED_POINT)
    float *floatPt=(float*)greyData;
    DTYPE greyDataFixed[SIZE_IMG];
    for (int i=0; i<SIZE_IMG; i++) greyDataFixed[i] = (DTYPE)floatPt[i];
    unsigned int greyDataUint[SIZE_IMG];
    for (int i=0; i<SIZE_IMG; i++)
        greyDataUint[i] = *(unsigned int*)&(greyDataFixed[i]);
#endif

    ...
    // 2. check IP is ready
    ...
```

How to get 32-bit bit-pattern from fixed-point

How to make fixed-point from floating point

$PROJECT/codes.fpga/LeNet_fpag/sw.native/lenet.confmc/src.fixed/main.cpp

C-driven host program (2/2)

```
    // 3. write image data to the IP
#if !defined(FIXED_POINT)
    MEM_WRITE_G(ADDR_IMG, &greyData[0], 4, SIZE_IMG);
#else
    MEM_WRITE_G(ADDR_IMG, &greyDataUint[0], 4, SIZE_IMG);
#endif
    // 4. let IP go and wait for completion
    ...
    // 5. read results from the IP
    unsigned int resultClasses[10];
    MEM_READ_G(ADDR_RESULT, &resultClasses[0],
                    4, 10);
    // note that 'resultClasses' carries fixed-point
    // contents
```

```
#if defined(FIXED_POINT)
    // carry out softmax, which is not implemented in RTL
    DTYPE resultFixed[NUM_CLASSES];
    for (int i=0; i<NUM_CLASSES; i++)
        resultFixed[i] = *(DTYPE *)&(resultClasses[i]);
    softmax(resultClasses, resultFixed);
#endif

    // 6. print the results
    ...

    return 0;
}
```

Softmax

$PROJECT/codes.fpga/LeNet_fpag/sw.native/lenet.confmc/src.fixed/main.cpp

Prepare LeNet-5 IP

- 1. HLS Synthesis
- - go to 'hw/hls/tcl.fixed'
- - run 'make'

$PROJECT/codes.fpga/LeNet_fpag

add following in '.bashrc' file.

```
alias set_vivado='source /opt/XilinxWebpack/Vivado/2018.3/settings64.sh;₩
                  export XILINX_VIVADO_HLS=/opt/XilinxWebpack/Vivado/2018.3;₩
                  export XILINX_SDK=/opt/Xilinx/SDK/2018.3;₩
                  source ${XILINX_SDK}/settings64.sh'
```

```
... ...
$ source /opt/XilinxWebpack/Vivado/2018.3/settings64.sh
$ export XILINX_VIVADO_HLS=/opt/XilinxWebpack/Vivado/2018.3
... ...
$ cd hw/hls/tcl.fixed
$ make
```

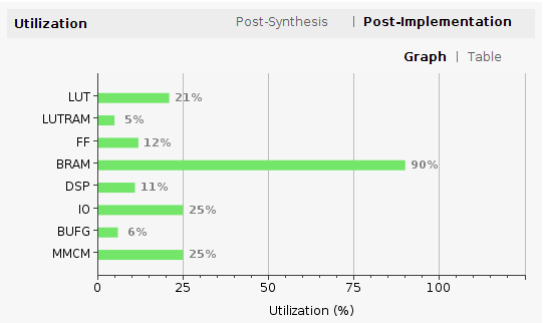

Prepare whole design

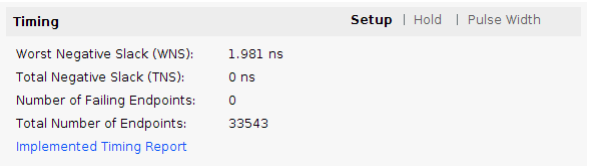
- 2. VIVADO IP Integrator
- - go to 'hw/impl/vivado.zed.confmc'
- - run 'make FIXED_POINT=1'

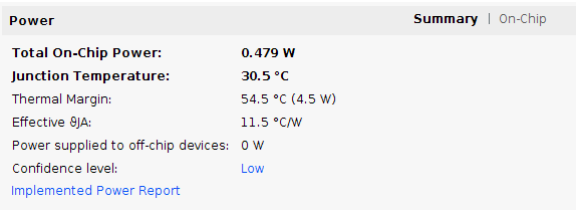
$PROJECT/codes.fpga/LeNet_fpag

```
... ...
$ source /opt/XilinxWebpack/Vivado/2018.3/settings64.sh
... ...
$ cd hw/impl/vivado.zed.confmc
$ make FIXED_POINT=1
```

Implementation report

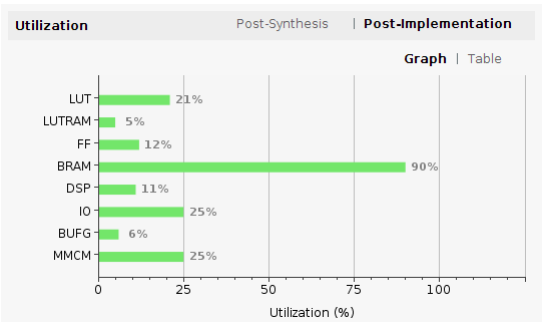




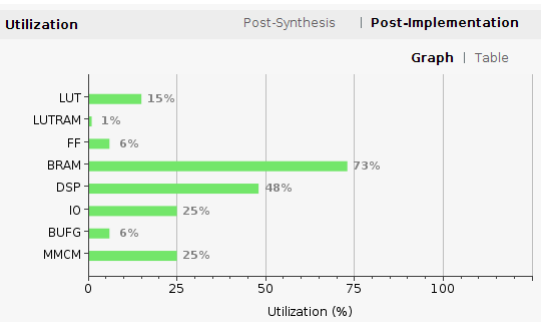


fixed-point v.s. floating-point

- Fixed-point



- Floating-point



Note that fixed-point version does not use exponent table.

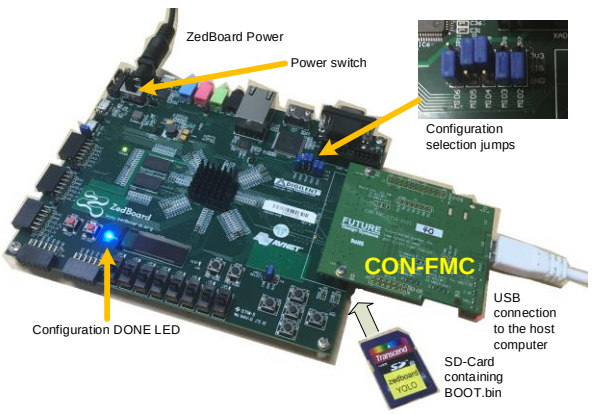
Prepare FPGA image

- 3. Boot file generation
- - go to 'hw/impl/vivado.zed.confmc/bootgen'
- - run 'make FIXED_POINT=1'
- - copy 'BOOT.bin' to the SD-CARD and then tun on ZedBoard

$PROJECT/codes.fpga/LeNet_fpag

```
... ...
$ export XILINX_SDK=/opt/Xilinx/SDK/2018.3
$ source ${XILINX_SDK}/settings64.sh
... ...
$ cd hw/impl/vivado.zed.confmc/bootgen
$ make FIXED_POINT=1
```


HW setup



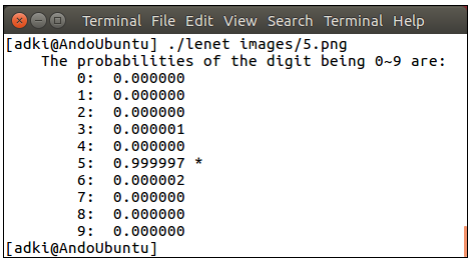
- 1. Turn off ZedBoard
- 2. Check configuration jumps
- 3. Insert SD-Card
- 4. Connect USB port
- 5. Turn on ZedBoard

You should see followings on you host computer.

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

Running the host program through USB

- 4. CON-FMC software (OpenCV is required)
- - It requires OpenCV and CON-FMC
- - go to 'sw.native/lenet.confmc' directory
- - run 'make clean; make FIXED_POINT=1'
- 5. Run
- - go to 'sw.native/lenet.confmc' directory
- - run './lenet images/5.png'




```
... ...
$ souce /opt/confmc/2919.10/sttings.sh
... ...
$ cd sw.native/lenet.confmc
$ make FIXED_POINT=1
$ ./lenet images/5.png
```


Projects

- Make a more compact version using less bits of data
  - ap_fixed<16,8>
- Make a training version to get new bias and weight.

References

- Changwoo Lee, Jeonghyun Woo, FPGA Accelerator for CNN using Vivado HLS, https://github.com/changwoolee/lenet5_hls
- Lenet for MNIST handwritten digit recognition using Vivado hls tool, https://github.com/FloyedShen/mnist_hls
- https://github.com/a2824256/HLS-LeNet, https://github.com/a2824256/HLS-LeNet

- CON-FMC User Manual, FDS-TD-2018-03-001, Future Design Systems.
- CON-FMC API based on LIBUSB, FDS-TD-2018-04-004, Future Design Systems.
- PyCONFMC: CON-FMC Python Binding, FDS-TD-2019-12-002, Future Design Systems.