Classic CNN for Handwritten Digits

Richard Lin
Jianxiong Xu

Yifeng Zhang
Genwen Zhao

Project Overview

Classic CNN handwritten digit recognizer implemented Nexys board. Able to classify user handwritten digit displayed in front of an HDMI camera.

Why FPGA?

- High computational capability on floating point matrix operation. Great for CNN application. Configurable HW for fast development & testing.

Main requirements:

- HLS to develop CNN core with FC, CONV and max pooling layers
- HDMI video IPs and Microblaze processor for data capture and control
- External DDR for main data storage

Design Environment

Hardware Platform - Nexys Video board - Artix-7 XC7A200T-1SBG484C

Software Tools - 2017.2 Vivado, HLS, & SDK. Python 3.7.3

Revision Control - Initially Dropbox and Google Drive. Github for submission.

- Design is partitioned into HLS CNN Core (Genwen & Yifeng) or HDMI System (Jianxiong & Richard)

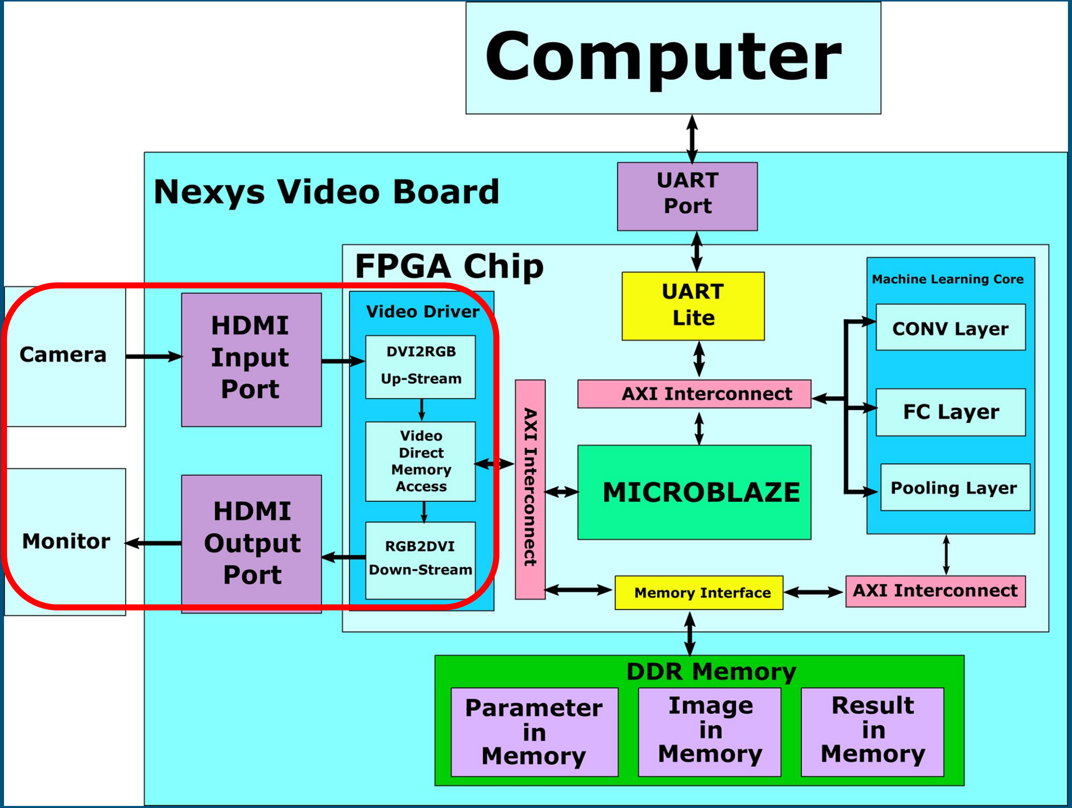
- Both sections are individually tested & demoed first
- Integrate & review design files through in-person team meetings

Overall System architecture - HDMI

- HDMI Input Datapath
  - Convert TMDS/DVI to AXI-4 Stream RGB Data
  - Store this 1920x1080 image in DDR memory

- Process image in SW, run CNN, write image of detected number to DDR (described in next slides)

- HDMI Output Datapath
  - Access DDR for above image output
  - Convert from AXI-4 Stream RGB Data to TMDS/DVI

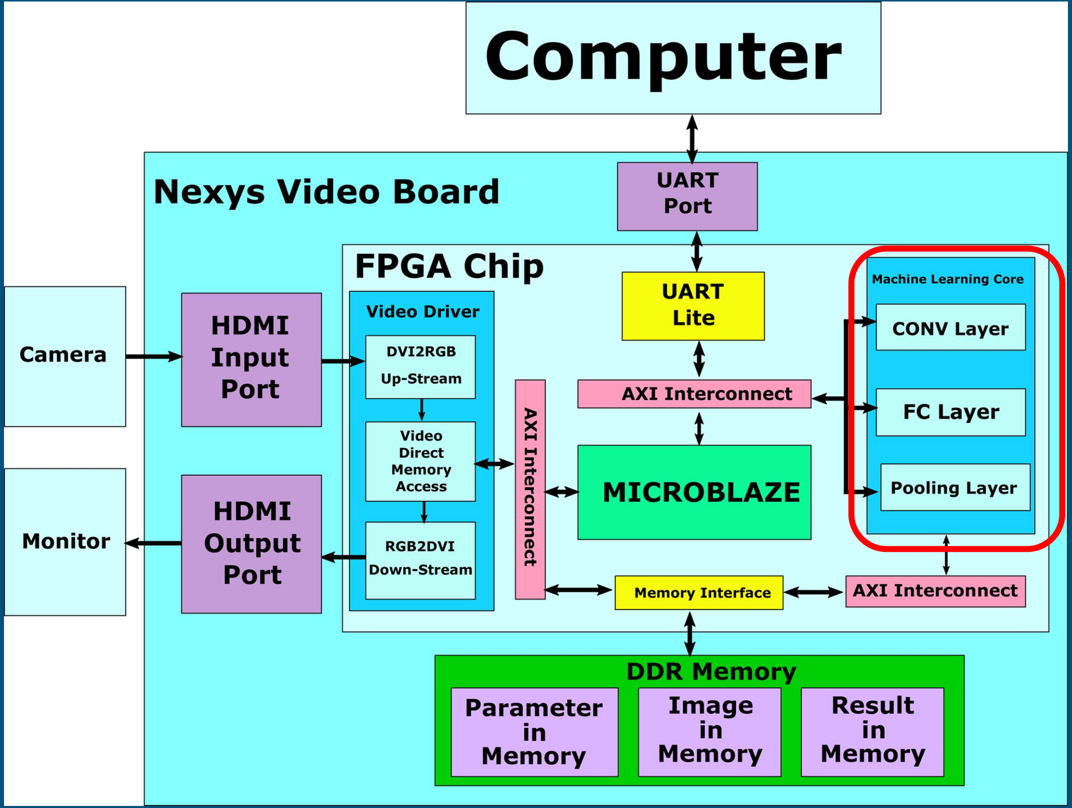


Overall System architecture - HLS Core

- CNN core - 1 of each in HW
  - Convolutional layer
  - Max pooling layer
  - Fully connected layer

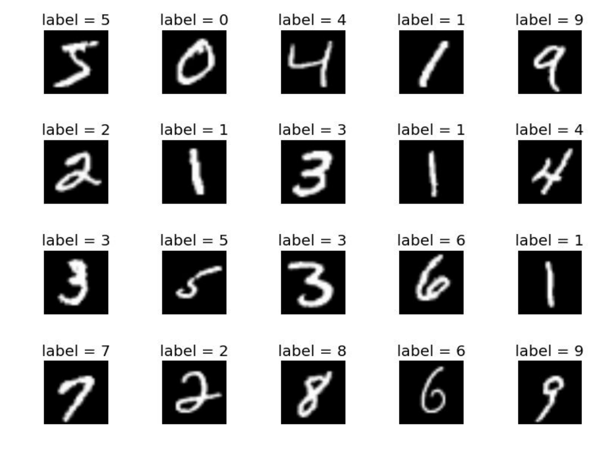
- Software sends data through CONV, POOL, CONV, POOL, FC, FC

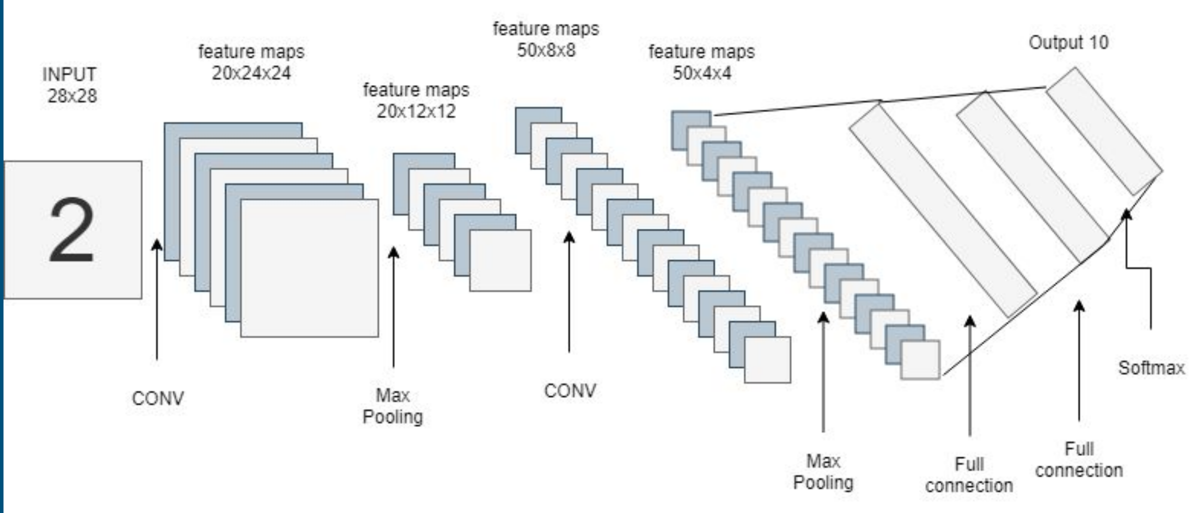
- Microblaze provides a processed image (28x28, black and white)
- Outputs final results to Microblaze



Overall System architecture - HLS Core

- Trained LeNet CNN Architecture with MNIST database (60,000 training examples) on Caffe framework
- Achieved 99.05% accuray with the test data (10,000 test images) in Caffe framework





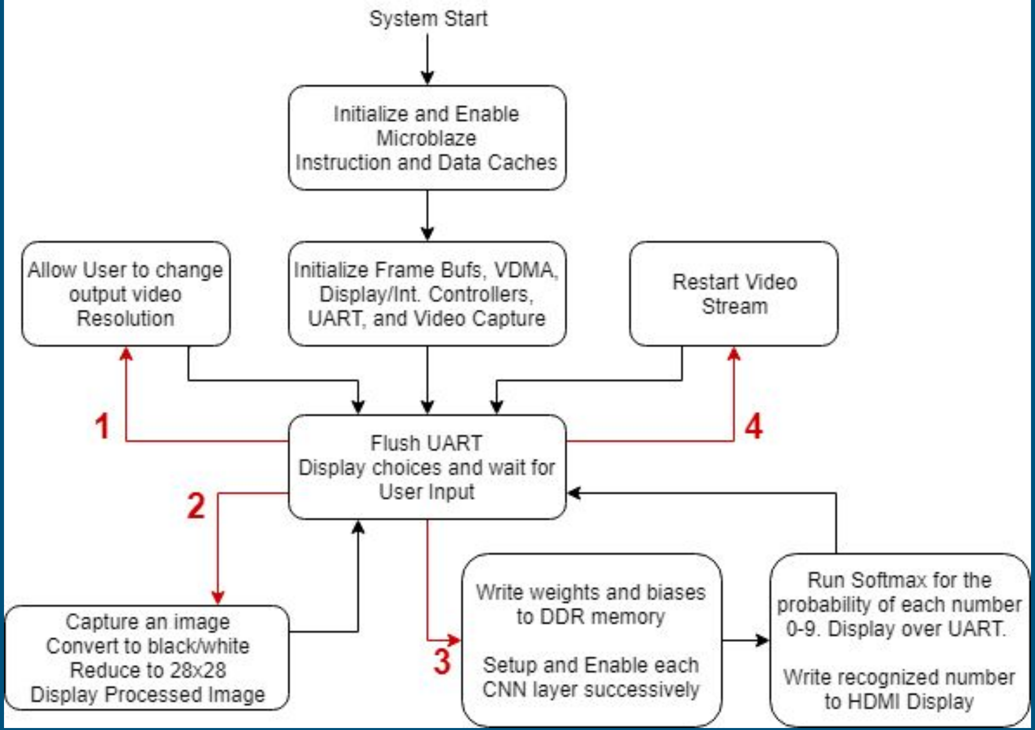
Overall System architecture - Microblaze

- CPU to schedule tasks, initialize HW, and manage UART

- Convert HDMI Image:
  - RGB to B/W
  - 1920x1080 to 28x28

- Pass processed image and enable HLS CNN Core layers successively

- Use softmax to process HLS CNN core output. Display final recognized number to DDR for HDMI and sent probabilities of each number over UART


System Start
Initialize and Enable Microblaze Instruction and Data Caches
Allow User to change output video Resolution
Initialize Frame Bufs, VDMA, Display/Int. Controllers, UART, and Video Capture
Restart Video Stream
1
4
Flush UART Display choices and wait for User Input
2
Capture an image Convert to black/white Reduce to 28x28 Display Processed Image
3
Write weights and biases to DDR memory
Setup and Enable each CNN layer successively
Run Softmax for the probability of each number 0-9. Display over UART.
Write recognized number to HDMI Display

Verification & Testing of the IP Modules

IP level:

- Each HLS/IP module has its own test bench, and verification was done individually before system integration.
    - Hardware modules
        - HDMI image capture block
        - HLS CNN core
        - Microblaze
        - DDR access
    - Software modules
        - Image compression engine and preprocessing function
        - CNN core drivers
- Defined clear I/O interface for each module, and data/control paths between IPs based on our top level system diagram.

Verification & Testing of the System

System level:

- Our system level testing strategy is to integrate one block at a time
  - First - test the Microblaze and DDR access part together
  - Second - add the HLS core to the system
  - Third - integrate the image compression engine into the system
  - Last - add the HDMI module to perform the whole system testing
- Our integration and testing strategy helps to reduce the complexity and time spent on debugging

Overall HLS experience:

HLS in project:

- Main CNN Core is implemented using HLS.
- Develop CNN in algorithmic level using C/C++, save 60 - 70% of time compared to traditional HDL

Advantage:

- Easy to implement complex algorithm in hardware using high level programming language, without taking care of low level hardware interface

- Fast turnaround time for hardware changes. Minimal change in main algorithm, use HLS Pragma to generate different hardware architectures for specific requirements

Overall HLS experience - cont’

Disadvantage:

- Unpredictable hardware from HLS Pragma - Pipeline & unrolling can generate unnecessary hardware for low efficient data parallelism due to other coding limitation
- Timing sensitive logic, FSM control logic are still easier to implement in HDL for more predictable hardware
- Debugging HLS generated hardware can be challenging compared to HDL

Project Schedule and Accomplishment

Milestone 1: Research and Setup Phase - March 20th (Completed)

- Richard & Jianxiong researched for Vivado Project, board IO, memory
- Yifeng & Genwen researched on CNN optimization technique and exiting CNN training methodology
- Overall system architecture, design partition and interface definition

Milestone 2: Development Phase - April 15th (Completed)

- Successfully bring up HDMI video streaming (direct Video Input to output on monitor)
- Successfully bring up CNN core and verify functionality in IP level
- Successfully bring up LeNet training model and obtain trained parameter sets

Milestone 3: Integration and Test phase - April 30th(Completed)

- Successfully package and Integrate CNN core to Nexys board with HDMI video IPs
- Developed a full working CNN application in Xilinx SDK flow- including CNN core drivers, HDMI image capture and process functions
- Able to achieve >90% classification accuracy on HDMI camera captured handwritten digits!

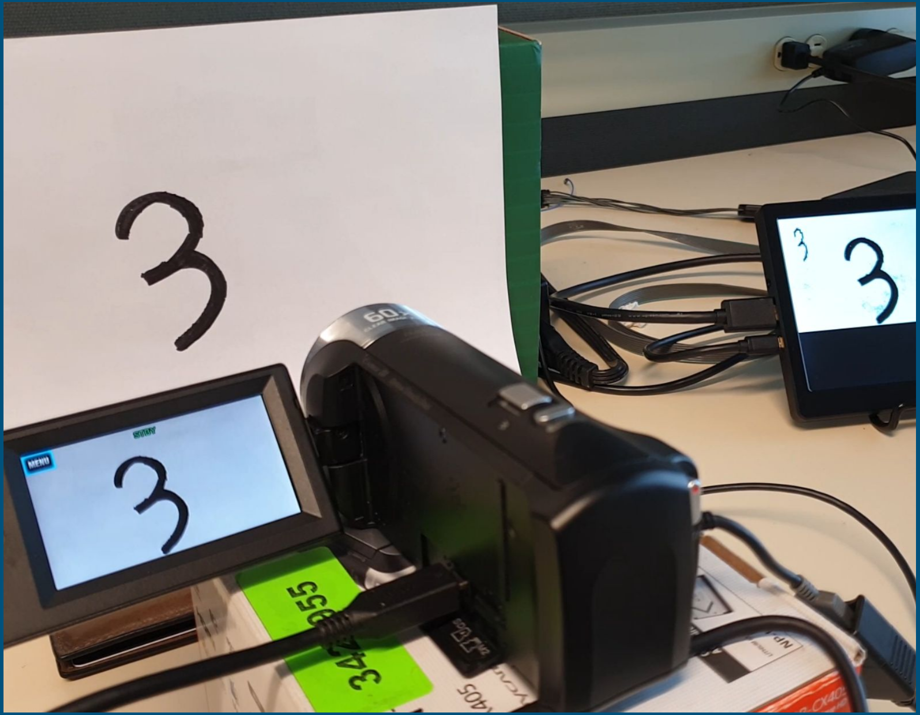
Project Demo and Q/A

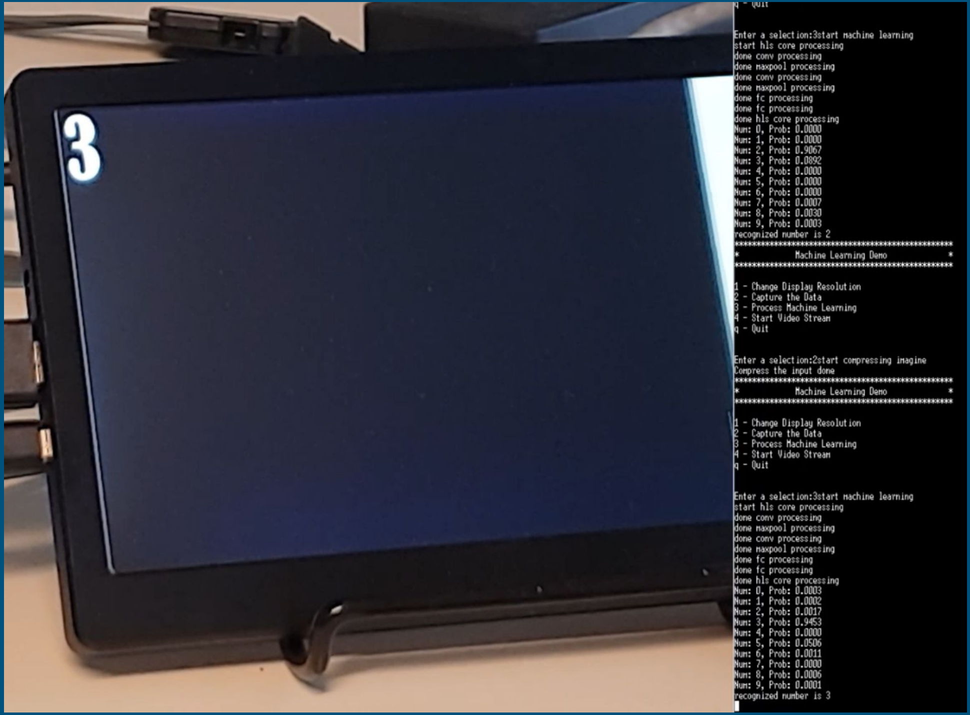
Github Link: https://github.com/richardlin23/ECE1373_CNN_Digits

Demo with Camera Input: https://youtu.be/ZbqtS1RtxsI

Demo with Computer Input: https://youtu.be/WsqfhlvcGVA

Questions for the team ?





```
q - Quit


Enter a selection:3start machine learning
start hls core processing
done conv processing
done maxpool processing
done conv processing
done maxpool processing
done fc processing
done fc processing
done hls core processing
Num: 0, Prob: 0.0000
Num: 1, Prob: 0.0000
Num: 2, Prob: 0.9067
Num: 3, Prob: 0.0892
Num: 4, Prob: 0.0000
Num: 5, Prob: 0.0000
Num: 6, Prob: 0.0000
Num: 7, Prob: 0.0007
Num: 8, Prob: 0.0030
Num: 9, Prob: 0.0003
recognized number is 2
************************************************************
*               Machine Learning Demo                      *
************************************************************


1 - Change Display Resolution
2 - Capture the Data
3 - Process Machine Learning
4 - Start Video Stream
q - Quit


Enter a selection:2start compressing imagine
Compress the input done
************************************************************
*               Machine Learning Demo                      *
************************************************************


1 - Change Display Resolution
2 - Capture the Data
3 - Process Machine Learning
4 - Start Video Stream
q - Quit


Enter a selection:3start machine learning
start hls core processing
done conv processing
done maxpool processing
done conv processing
done maxpool processing
done fc processing
done fc processing
done hls core processing
Num: 0, Prob: 0.0003
Num: 1, Prob: 0.0002
Num: 2, Prob: 0.0017
Num: 3, Prob: 0.9453
Num: 4, Prob: 0.0000
Num: 5, Prob: 0.0506
Num: 6, Prob: 0.0011
Num: 7, Prob: 0.0000
Num: 8, Prob: 0.0006
Num: 9, Prob: 0.0001
recognized number is 3
```