Acceleration of Digit Classification Using Custom CNN on an SoC FPGA

CSEE - 4840 Embedded Systems

Presenters:

Prathmesh Patel

Vasileios Panousopoulos

Rishit Thakkar

Tharun Kumar Jayaprakash

May 15, 2024

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

Introduction

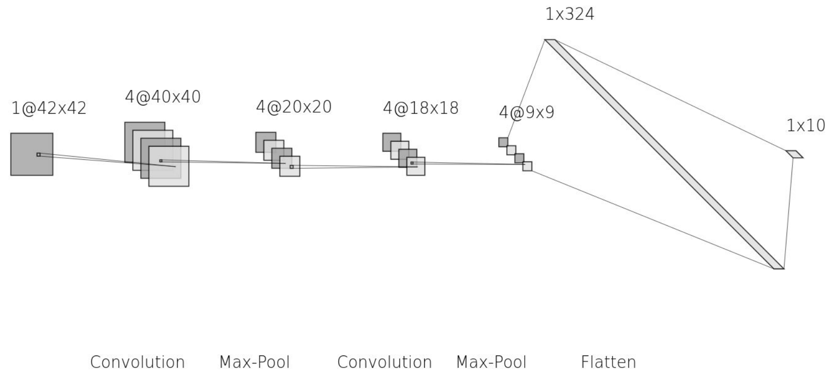
- Designed a simple custom Convolutional Neural Network
  - Application: Handwritten digit classification
  - Trained on the MNIST dataset with HW constraints in mind
  - Implemented a Fixed Point Simulator to verify hardware modules

- Implemented a real-time system om DE1-SoC FPGA Board with HW/SW Co-Design
  - Integrated the OV7670 camera module to capture handwritten digits
  - Implemented a parameterized accelerator of the entire CNN model
  - C code is used to control the execution of different layers

Convolutional Neural Network: Architecture

- Designed a lightweight network inspired by simple architectures as LeNet
  - Easy to achieve high accuracy in such an application

- Architecture:
  - 2 convolutional layers with 3x3 kernels of stride 1 and ReLU activation
    - 9 MAC (weights) and 1 extra addition (bias)
  - 2 max pooling layers with 2x2 kernels and stride 2
    - 4-number comparison to extract the maximum value
  - 1 output dense layer that gives the final classification
    - 324 MAC (weights) and 1 extra addition (bias)



Convolutional Neural Network: Details

- Input image size: 42x42 pixels
  - Number of weights: 3,438

- Floating point model trained on the MNIST dataset
  - Test accuracy: ~90%
  - Train dataset was modified to match application's data and HW constraints

- Fixed point simulator was implemented to verify the HW accelerator
  - Test accuracy: ~90%

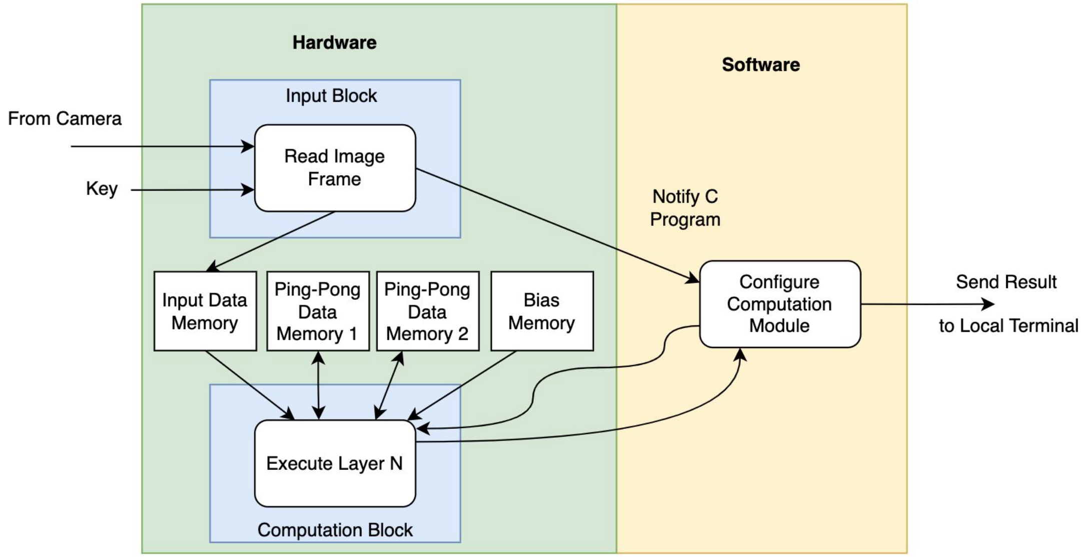
Convolutional Neural Network: Fixed Point Implementation

- 8 (4,4) bits per data (2’s complement)
  - Memory budget: 3.4KB (weights) + 1.7KB (data)

- Dataset manipulation:
  - Input images divided by 2 to match 8 bits to 8 signed fixed-point value
  - Trained weights scaled from (-1, 1) to (-8, 8)
  - Final result of MAC divided by 8 before stored

System Architecture

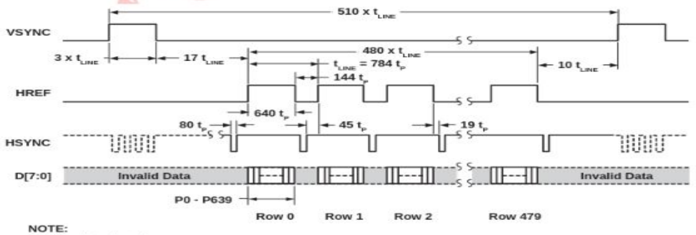
- Image is captured with a key press and stored on-chip
- SW is notified and invokes the Computation Block for each layer
  - Different configuration for each layer through Avalon Bus
- Computation Block uses Ping-Pong memories to store intermediate results



Camera Sensor

- Using OV7670 to derive the grayscale image
  - Operating at 12MHz
  - Configuring 79 registers to receive images in YUB422 format in size 80x60
  - Sending frames at 30fps

Figure 6 VGA Frame Timing



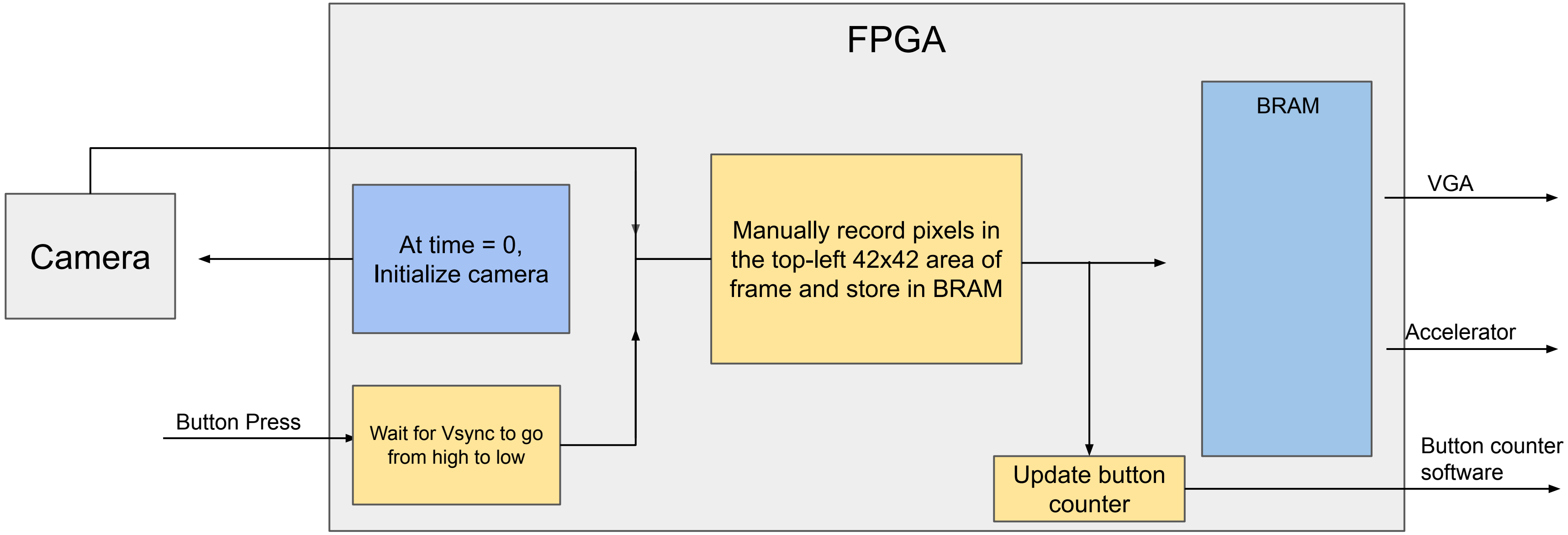
NOTE:
For Raw data, $t_P = t_{PCLK}$
For YUV/RGB, $t_P = 2 \times t_{PCLK}$

7670CSP_DS_006

7 *Proprietary to OmniVision Technologies, Inc.* Version 1.4, August 2

COLUMBIA | ENGINEERING
The Fu Foundation School of Engineering and Applied Science

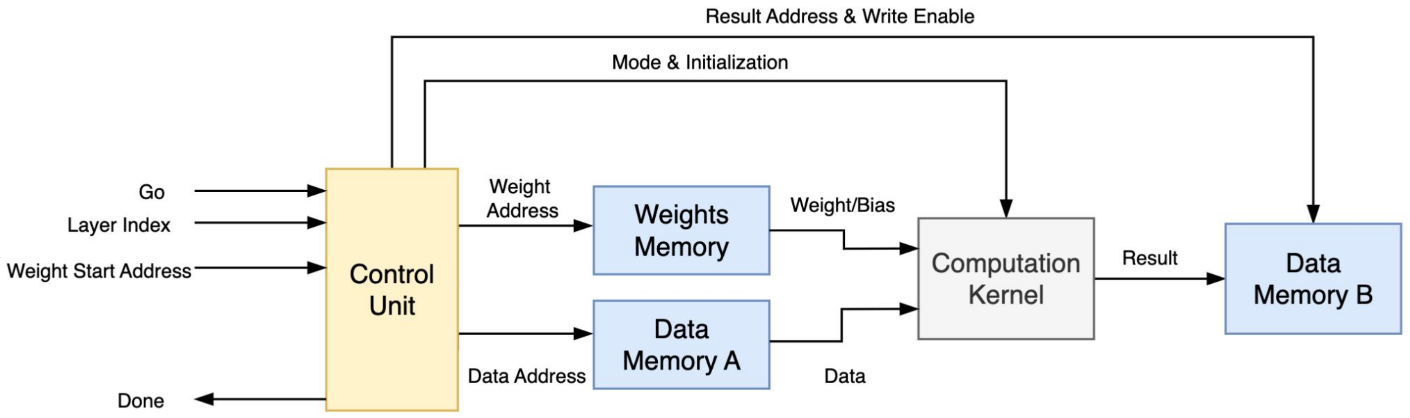
Camera Sensor Workflow


FPGA
Camera
At time = 0, Initialize camera
Wait for Vsync to go from high to low
Button Press
Manually record pixels in the top-left 42x42 area of frame and store in BRAM
BRAM
Update button counter
VGA
Accelerator
Button counter to software

CNN Accelerator: Overview

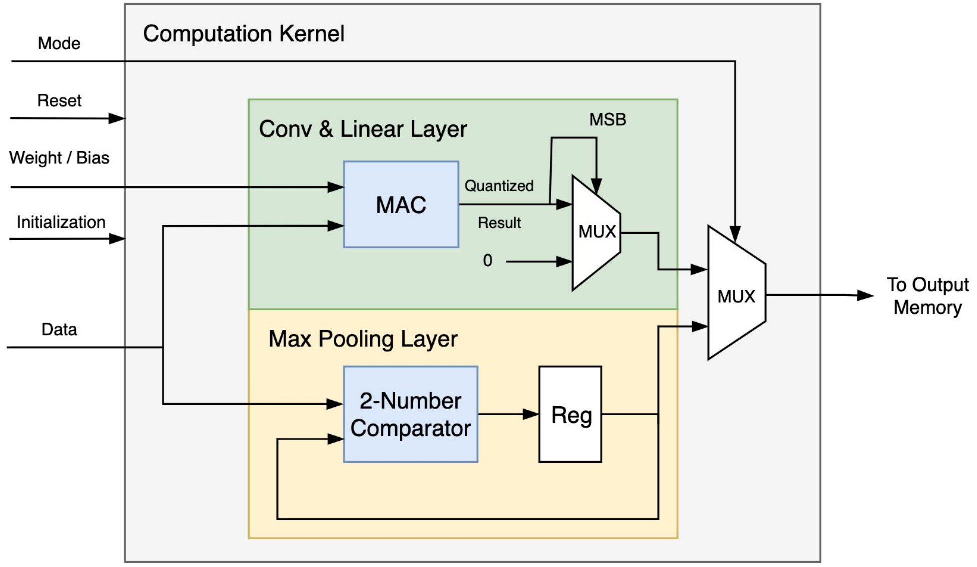
- Accelerator performs one layer computation per execution
  - Layer index indicates the layer to be executed
  - Control Unit is implemented as an FSM that drives other modules accordingly

- 2 Ping-Pong memories are used for data transfer and 1 ROM memory for weights
  - Input image data are read by a different memory



CNN Accelerator: Computation Kernel

- Performs both MAC and comparison operations in parallel
  - Final result is selected based on the currently running layer
    - CU is responsible for sampling at the correct cycle
  - MAC module is mapped to a DSP block



CNN Accelerator: Verification

1. RTL testbench (Modelsim) to compare against SW results per layer
2. Verilator testbench to match the way SW handles the accelerator
    a. Automated process of verifying new versions fast

- Biggest Challenge:
    - Correct Implementation of arithmetic operations in 2’s complement
    - Bad overflow detection
        - Used for ReLU and max-pooling layers

Software - Hardware Interface

- Write from SW
  - RegisterA: 1 byte
    - RegisterA[0]: Go
    - RegisterA[3:1]: Layer index
  - RegisterB: 1 byte
    - RegisterB[7]: Whether read will return Done signal or classification value
    - RegisterB[3:0]: Address to read classification value

- Read from HW
  - RegisterC: 1 byte
    - Key counter for debugging
  - RegisterD: 1 byte
    - RegisterD[0]: Done signal
    - RegisterD[7:0]: Classification result

Compromises and Limitations

- Input image scaling limited to 60x80 pixels
  - Only the upper 42x42 pixels are stored in memory
- Problem with increasing number of registers written/read through Avalon Bus
  - Only 2 write and 2 read addresses were used
  - Weight Memory starting address was not passed from SW but hardwired
  - Not able to add more functionalities (real-time comparison with SW results)
- 2 memories were mapped to 2 BRAMs each
  - Read by different modules both sequentially and combinationally

Thank you !