Columbia University
Fu Foundation School of Engineering and Applied Science

# Acceleration of Digit Classification Using Custom CNN on a SoC FPGA

CSEE 4840 - Embedded Systems
Design Document

**Tharun Kumar Jayaprakash** (tj2557)
**Vasileios Panousopoulos** (vp2518)
**Prathmesh Patel** (pp2870)
**Rishit Thakkar** (rht2122)

Spring 2024

# 1 Introduction

This project presents an approach to digit classification leveraging the DE1-SoC FPGA board. The aim is to implement a fast solution for real-time digit recognition using the MNIST dataset and a simple convolutional neural network (CNN) architecture. The FPGA environment offers unique advantages for accelerating computationally intensive tasks like image processing. Through the utilization of SystemVerilog, the CNN model will be implemented directly on FPGA hardware, exploiting the parallelism and inherent efficiency of hardware-level computations. A critical aspect of this project is integrating a camera module with the system. This setup enables the capture of real-world handwritten digits for classification directly on the FPGA. The system will output the predicted class of the digit, providing a seamless and efficient solution for digit recognition applications. By showcasing the practical implementation of machine learning models on FPGA, particularly in the context of image classification, this project aims to demonstrate the feasibility and effectiveness of FPGA-based solutions for accelerating applications in the field of computer vision and pattern recognition.

# 2 Proposed System

## 2.1 Algorithm: CNN System

Since the primary objective of the project is to just implement a simple network, rather than training a network from scratch, an existing trained network can be leveraged. This network is similar to proven architectures, such as LeNet, and is a lightweight network made of 2 convolutional, 2 max-pooling, and 1 dense layer. The general network architecture can be seen in Figure 1.

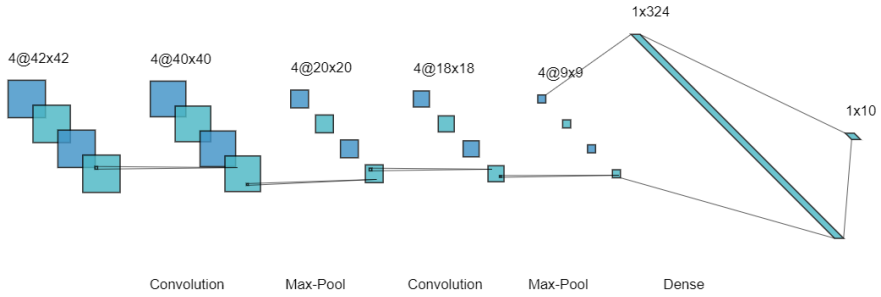


**Figure 1:** Model General Architecture

We retrained the model with QKeras, making the quantization bit width for weights and biases as eight (4 for integer part, 4 for fractional part) and got a testing accuracy of 95%. The total number of parameters in the network will equal 3,438 and will be stored on the on-chip memory in the accelerator. The input image, sized at 42 x 42 pixels(1Byte wide), undergoes the initial convolutional layer, yielding four output channels. Subsequently, the image is further processed through a second convolutional layer, with four input and output channels, and employing a 3x3 kernel for feature extraction. Max-pooling layers, with 2x2 kernel size, are integrated into the architecture to downsample the feature maps, facilitating computational efficiency while preserving essential features. After flattening(324 neurons), the network has a fully connected (dense) layer comprising ten output units corresponding to the ten distinct classes in the dataset. Throughout the architecture, Rectified Linear Unit (ReLU) activation functions are used to introduce non-linearity and

enable efficient gradient propagation. This design ensures a balance between computational efficiency(Resources) and performance, making it well-suited for deployment on the DE1-SoC FPGA.

### 2.2 Use Case

The goal of this project is to implement an interactive real-time application. Specifically, the user will be directly involved in the data pipeline by drawing a single digit on a blank paper. This paper will then be sampled by a camera sensor connected with the DE1-SoC Board and the captured frame will be processed by the SW/HW system to classify the drawing into one of the 10 different classes-digits. The result will then be provided back to the user through the terminal of the local PC.

## 3 System Implementation

### 3.1 System Dataflow

The proposed system will be based on SW/HW Co-Design methodology, where the Software and Hardware sides of the SoC will need to communicate effectively to produce the expected result. A high-level block diagram of our system is given in Figure 2.

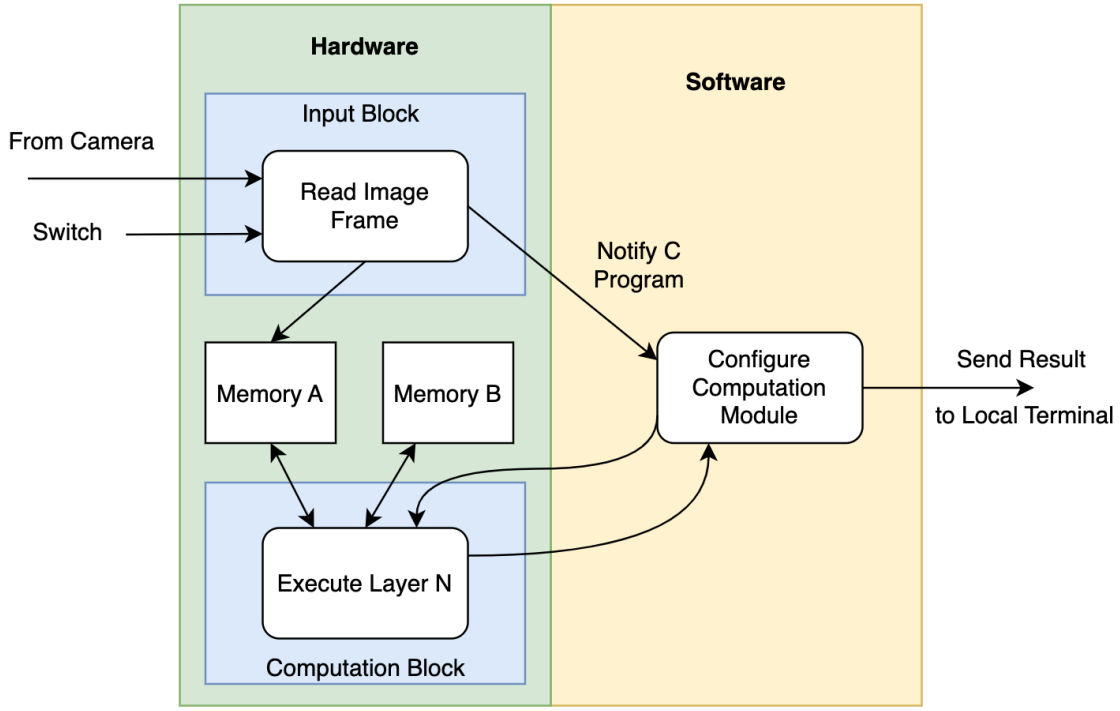


**Figure 2:** High-Level Block Diagram

The hardware side of the SoC will consist of 2 basic modules. The Input Block will implement the camera driver and will be instructed to read a frame once an on-board switch is toggled. Then, it will store the image in a predefined address space and will notify the processor that an image is available. The C program will in turn configure and execute the second hardware module, called Computation Block, multiple times to perform the necessary calculations throughout the network

pipeline. Once the last layer is executed, the C program will send the selected class (digit) to the user through the local terminal, as done in Lab 2.

## 3.2 Computation Block Architecture

The main feature of the proposed system is the implementation of the CNN architecture on the FPGA side of the SoC. More specifically, in order to achieve inference acceleration on hardware, we will design a module that can be configured by the software in run-time to perform the computations required by the convolutional layers, the max-pooling layers, and the output linear layer. The microarchitecture of the proposed module is shown in Figure 3.

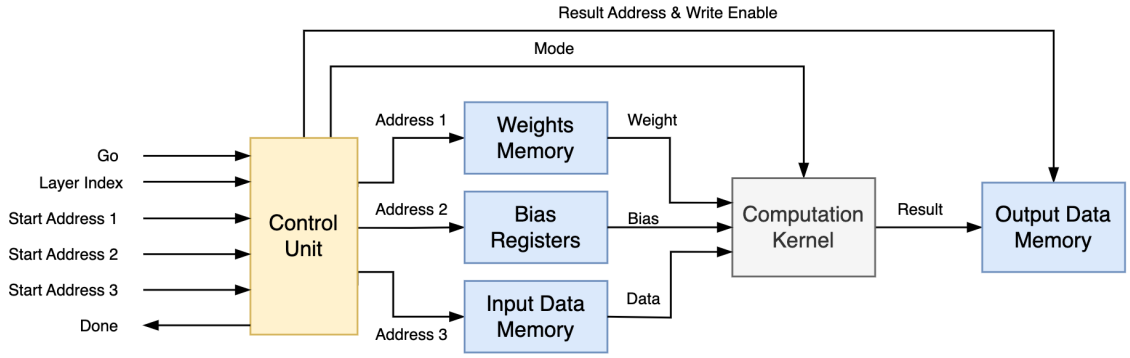


**Figure 3:** Hardware System Microarchitecture

A Control Unit (CU), implemented as an FSM, will be the front end of the HW module and will be responsible for communication with the C program. In particular, once data required by the network's layer are available in the respective memories, the C program will initialize the HW module and will provide the appropriate configuration information. The input signals consist of a layer index and 3 memory indices. The `Layer Index` indicates which layer will be run by the module, i.e., one of the two convolutional layers, one of the two max-pooling layers, or the output linear layer. This signal will be used internally by the CU to select which memories will be accessed for input data and which for results and to determine the timing details of how the computation kernel will be controlled. Regarding memory indexing, the system will use two different memory spaces, Memory A and Memory B, for storing the input and output data of each layer. For instance, while the first convolutional layer will see the input image in Memory A and will store the computed feature maps in Memory B, the max pooling process that follows will read its inputs from Memory B and will store the decimated data in Memory A. Regarding the control of the computation kernel, it is clear that each layer needs to process different amounts of data. For example, the first convolutional layer processes 42x42 pixels while the max pooling layer needs to access 40x40x4 pixels, which means that the number of cycles the CU should wait in order to enable writing to the Result Memory (the valid result) differs across different executions of the HW module (different layers).

The Computation Kernel is responsible for performing the arithmetic operations required by each layer. On the one hand, the convolutional and linear layers need to perform large numbers of multiplications and additions, which can be easily done by a Multiplier-Accumulator Unit. On the other hand, during max pooling, 4 different numbers are compared, and the highest is selected. The microarchitecture of the Computation Kernel is displayed in Figure 4.

As explained, this kernel will be configurable to be able to perform all of the operations required

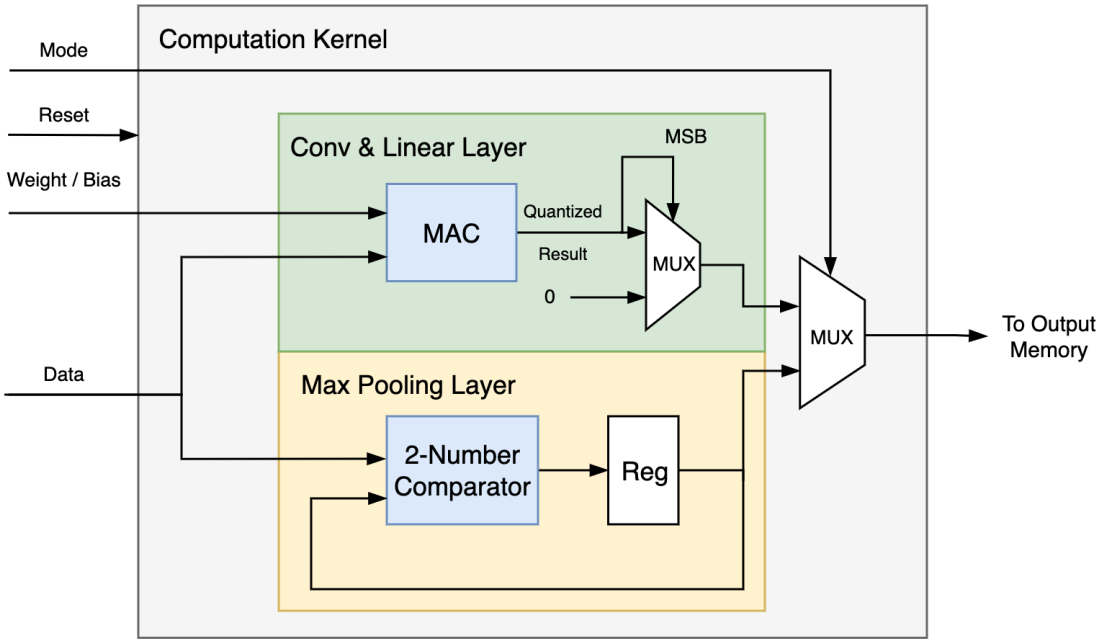


**Figure 4:** Computation Kernel Microarchitecture

by a CNN. The configuration is controlled by the CU through the `Mode` signal, which determines whose subsystem's result is passed through the output multiplexer and written to the result memory. As shown, the first subsystem will be used when executing a convolutional or linear layer, while the second subsystem will be used to implement the max pooling layers. In the first case, the MAC Unit receives the input data and weights in consecutive cycles to calculate the layer's output. Since both the input data and weights will be represented by 8 bits, the result of each multiplication will be 16 bits wide. Considering the fact that these partial results will be small numbers, their accumulation can be modeled with 16 bits without loss in accuracy. Once the layer's result is calculated and quantized by retaining its middle 8 bits, a MUX controlled by the result's MSB performs the ReLU operation, as its output will be either 0 if the MSB is 1 (negative number) or the original result if the MSB is 0. In the second case, a 2-number comparator will be used to extract the highest number from a 2x2 window in 4 cycles. In both cases, the CU is responsible for enabling writing to memory by asserting the respective signal in the correct cycle.

Finally, once the required operations of the layer are completed, the Control Unit will inform the processor about the event by writing to a specific register and will move to an Idle state.

## 3.3 The Hardware/Software Interface

Upon startup of the FPGA, the software module will be responsible for initializing and configuring the camera. A register will be set to 1 to trigger the camera hardware module to send the configuration writes to specific registers. These values will be based on parameters such as clock speed, image capture mode, and desired resolution. The camera configuration and operation are described in subsequent sections.

To listen for user input, a switch will be used as a toggle signal, which will trigger the software program to initiate the accelerator when the user has finished drawing their number. To initiate the calculation in each layer, the C-based software will send a `go` signal, along with `layer index`,

and addresses of the appropriate weight, bias, and input data location. This software will also look for the done signal to write to another reserved register, which will signal the software to move to the next layer in the CNN architecture.

Finally, when the last dense layer has finished its calculation, it will write the resulting 10 probabilities, corresponding to each class, in 10 separate registers. The software program will read these values, determine the class with the highest probability, and print that result on the terminal.

The IOCTL system call and Intel Avalon Bus will be used to transfer the data between the software and hardware layers.

# 4 OV7670 Camera

The OV7670 camera, shown in Figure 5, is a lightweight module based on the Serial Camera Control Bus (SCCB) protocol, which is a subset of the I2C protocol. It has 18 pins with the configuration seen in Figure 6. Similar to I2C, SCCB has a `XCLK` signal, which is supplied by the master device to synchronize the data transmission of the slave. In exchange, the module uses the `PCLK` signal, in conjunction with the `D0-D7`, data lines to supply an output of 8 bits. Since our application requires single-channel grayscale input, the YUV configuration can be used to configure the camera and retrieve the output image. The luminescence value can be retrieved from the Y component of this format, and used as grayscale intensity of an individual pixel.

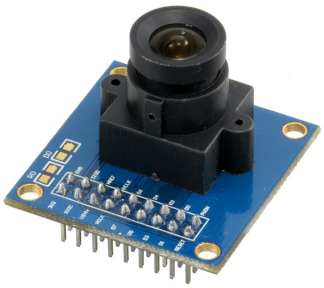

**Figure 5:** OV7670 Camera Module

The data will arrive in the configuration shown in Figure 7. The `U` and `V` fields are shared between every two pixels; therefore, to reconstruct a pixel into `RBG` values, the operation will have to be done in pairs. However, since our model requires grayscale images, we can ignore this format and sample the Y value without any further calculation. The sequence of data arrival is shown in Figure 8, along with the high-level VGA timing shown in Figure 9. It can be observed that the `tp` is twice the time of `tpclk`, due to the shared use of U and V fields between adjacent pixels. From this incoming data stream, every alternate byte can be stored, which will correspond to the Y value of every pixel.

As described in the CNN model architecture, the input image size is 42x42. To achieve this, the DSP unit on board the OV7670 will need to be leveraged. As seen in Figure 10, the default resolution of the camera is passed into the downsampling unit, which reduces the resolution by 1/8, and the subsequent digital zoom-out unit will resize the horizontal and vertical sizes to achieve the 42x42 size for the model. The hex values for configuring the horizontal and vertical scaling can also be seen in the diagram. There is a combination of device control registers, which will need to be configured upon startup by the software, to achieve the final resolution. These include `COM3`,

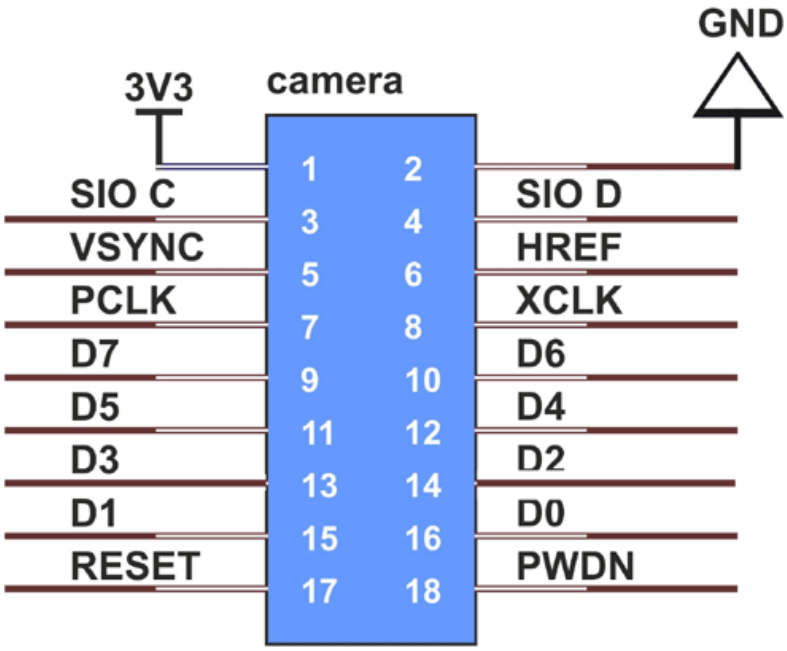


**Figure 6:** OV7670 Pinout

| PIXEL | INPUT CONFIG |
|---|---|
| 0 | U0Y0V0 |
| 1 | U0Y1V0 |
| 2 | U2Y2V2 |
| 3 | U2Y3V2 |
| 4 | U4Y4V4 |
| 5 | U4Y5V4 |
| ... | ... |

**Figure 7:** YUV Pixel Data Arrangement

`COM14`, `SCALING_XSC`, `SCALING_YSC`, `SCALING_DCWCTR`, `SCALING_PCLK_DIV`, `REG74`, and `REG75`. This step will occur only once during startup by the camera hardware module.

# 5 Resource Budgets

The OV7670 Camera module has a resolution of 640x480 pixels, and each pixel is 16 bits(2 Bytes), which is huge for the on-chip FPGA memory provided by the DE1-SoC, i.e. 256KB (BRAM). However, after employing the down-sampling and digital zoom-out, we will achieve a resolution of 42x42, and taking the image in YUB format, incorporating just the luminance/brightness will give 8 bits(1 Byte) for a pixel, making the image frame(1,764 bytes) stored in the on-chip BRAM. For the weights and biases of the model, we are representing those in 8 bits(4 for integer part and 4 for fractional part). The total trainable parameters of the model are 3,438 (total number of weights and biases), with each 8-bit wide, making it 3,438 bytes of memory for the weights and biases. Also, the MAC unit of the Computation Block will be mapped to a DSP unit so that the operations can be performed efficiently.

| INPUT | DATA |
|---|---|
| 1 | U0 |
| 2 | Y0 |
| 3 | V0 |
| 4 | Y1 |
| 5 | U1 |
| 6 | Y2 |
| 7 | V2 |
| 8 | Y3 |
| 9 | U2 |
| ... | ... |

**Figure 8:** YUV Data Arrival Sequence

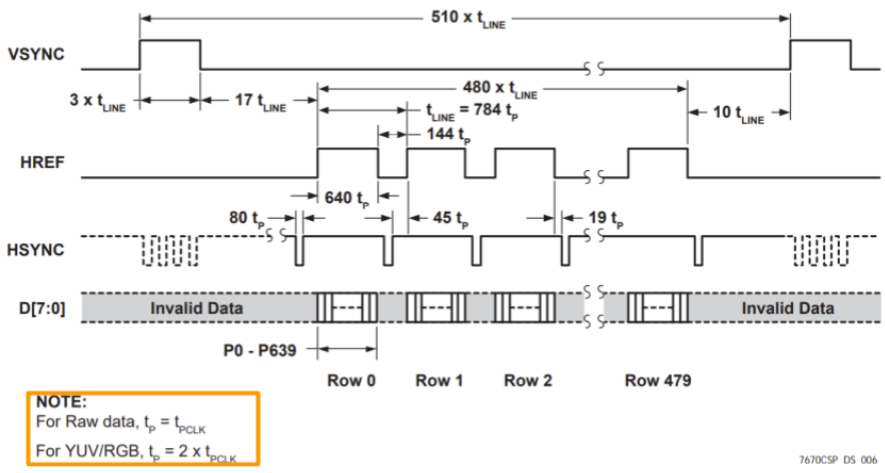


**Figure 9:** VGA Timing Based on OV7670 Datasheet

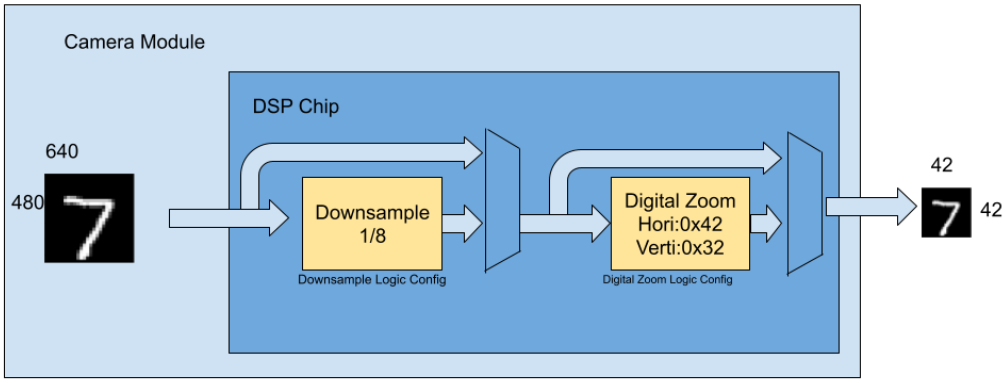


**Figure 10:** Resize Logic in OV7670