ECE 532- Final Report: CNN-based Number Recognition Using FPGA Acceleration

Chen Chen

Lily Li

Bowen Liu

Ruoyi Xie

April, 2024

Table of Content

1 - Overview

1.1 - Motivation and Goals

The goal of this project is to create a hand-written recognition system. The project aimed to deploy a convolutional neural network (CNN) to perform the recognition task on the FPGA to exploit the capability of computing the convolutions in parallel. The motivation is to build a system that can work continuously in real time with high power efficiency to be useful in many embedded systems or edge devices. Such applications include mobile devices, internet of things, automotives, etc.

The high level system is shown in Fig. 1. The OV5640 camera data is fed into the FPGA and stored in the DDR memory using a MicroBlaze processor core and software; then, the camera data is processed and displayed on a VGA display, showing the user that the camera is picking up. Finally, the processed data is read by the CNN module to compute a recognition result before displaying it on a 7-segment display.

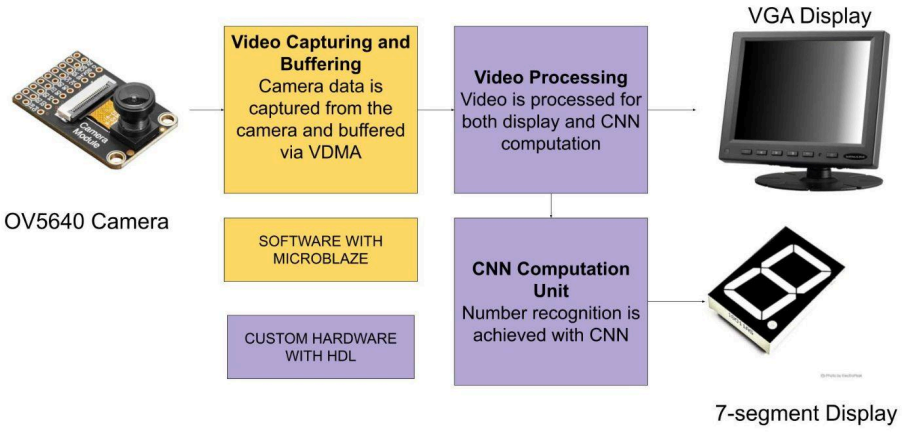

Figure 1. Design Overview

1.2 - Block Diagram

Fig. 2 is the block diagram of the system, the custom HDL blocks are shown in blue, whereas the Xilinx IPs are shown in white. The detailed functionality of each block is discussed in later sections.

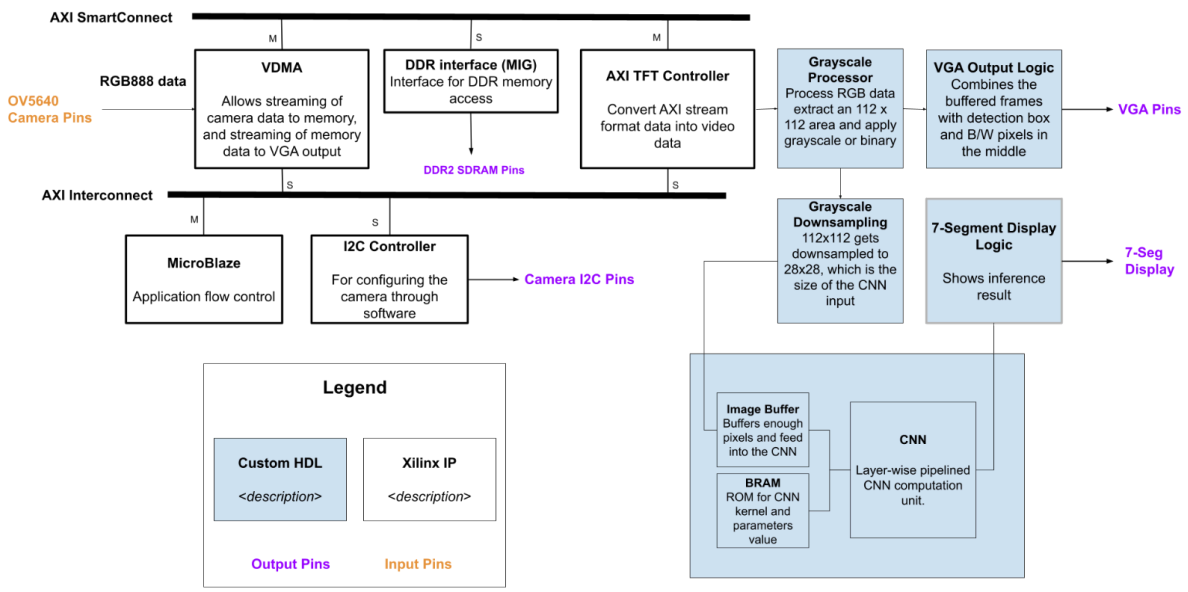

Figure 2. Block Diagram Describing Vivado Block Design

1.3 - Brief Description of IPs

1.3.1 Description of the Design Tree

Two AXIs are used in the design. The SmartConnect is used to connect the MIG, VDMA, and TFT controller, and the Interconnect is used to connect MicroBlaze to send commands to VDMA, TFT Controller, and IIC. The design requires 8-bit camera input with vsync, href, and pclk signals. The outputs include the IIC to the camera, DDR pins, VGA pins, and the 7-segment display. The design branches at the VGA output to make CNN computations of the captured data.

1.3.2 Xilinx IPs

MicroBlaze: used to control the VDMA and send I2C settings onto the camera. The source code is in test.c, and the I2C settings are referenced from the OV5640 datasheet and ov5640 software application notes. These settings are sent through the AXI_IIC IP.

AXI_IIC: used to send IIC settings to the OV5640 camera.

VDMA: used to facilitate the buffering of streamed video data in DDR, initiated and controlled by MicroBlaze. 5 frame buffers are used with a burst size of 8. The address and data widths are 32 bits.

Memory Interface Generator: generate logic used to interface with the DDR.

TFT Controller: used to convert video stream data to RGB565 for VGA display.

Microblaze Debug Module: helped debug the CNN calculations.

1.3.3 Custom IPs

OV5640 Capture: This custom IP combines every three cycles of 8-bit camera input to RGB888 pixels to be sent to the video stream. It pads 8'b0 in the front to fill up the 32-bit space for the AXI stream.

VGA Output: This IP takes the RGB565 data from the TFT controller and sends it to display on VGA. It counts using the vsync and hsync signals to locate every pixel, and draws a green detection box in the middle of the display, covering a 112x112 window. It also converts the area inside the green detection box to be grayscale or black and white. It then sends data inside the detection box to downstream IPs along with a valid signal indicating the pixels inside the box. It also sends a signal indicating the start of a frame.

Downsample: This module receives data from the VGA Output and marks pixels as valid once every 4x4 pixels, thus downsampling the 112x112 data to 28x28 for CNN input. This is done by counting the pixels received and comparing the last two bits of row count and column count.

BRAM: This module stores one frame of grayscale data (28x28) from the downsampling module, and notifies the CNN once the frame is received in full. Given the required addresses from the CNN, it will send 5 pixels at a time to CNN to perform inference (the first convolution layer of CNN has 5x5 kernels).

CNN: This module performs CNN inference. It consists of two convolution layers, one pooling layer and one fully connected layer. The computations within each layer are highly parallelized. It sends a request to BRAM when needed, and sends out the computation result with a valid signal indicating the computation is completed for the frame.

Selected Output: This module compares the 10 possibilities resulting from CNN and chooses the largest one to be displayed on the 7-segment display.

7-Segment Display: This module displays one of the 10 numbers on the 7-segment display LEDs.

OV5640 Power up: This module controls the power and reset of the camera.

2 - Outcome

2.1 - Results

The convolutional neural network (CNN), which was trained on the EMNIST dataset, has shown promising accuracy rates, affirming the model's efficacy in the FPGA setting. The system's ability to recognize handwritten numbers was bolstered by CNN's streamlined architecture, which was distilled to an efficient 1,158 parameters. This optimization ensured that the model remained lightweight, occupying a mere 2,316 bytes of memory—well within the FPGA's capacity. Such a configuration did not compromise robustness, making it a perfect fit for the memory constraints of the FPGA platform.

On the hardware side, the video capture and processing system is performed in real time. The Microblaze core adeptly handled I2C communications, which was crucial for capturing clear video data. Video buffering was managed by the VDMA, facilitating uninterrupted data flow to the TFT controller for video output preparation.

The VGA display consistently showed the camera feed, including a detection box that signaled where the number recognition was taking place. The system's design allowed for grayscale and binary displays, ensuring the output remained visible under various lighting conditions.

Processing speed was a critical factor in the project's success. The system adeptly kept pace with the necessary 60 Hz frame rate, essential for real-time application. Through meticulous optimization of data downsampling and buffering, the system was configured to feed the CNN with minimal delay. CNN is able to complete inference within 151 μs. This allowed the recognition process to operate effectively in a continuous stream, further demonstrating the project's technical accomplishments.

Upon processing completion, the CNN's output was displayed on a 7-segment display with impressive accuracy. This immediate and clear feedback mechanism provided a user-friendly interface, showcasing the recognized number as the final product of the system's intricate computation processes.

Overall, the system achieved its goals, showcasing the potential for FPGA-accelerated CNNs in real-time embedded applications. The implementation serves as a proof of concept for the effective use of FPGA in edge computing scenarios, where power efficiency and speed are paramount.

2.2 - Potential Improvements and Future Steps

There are three major directions of improvement for this project. The first one is the video streaming and camera settings. Currently, the video suffers from contrast and exposure problems, making the VGA output too dark sometimes; this issue could potentially be solved by tweaking some of the register settings on the OV5640 camera and finding the best fit for the current setup. The VGA output also experiences some lagging, which could potentially be solved by adjusting some of the AXI settings, such as the burst sizes, or the number of frames buffered.

The second issue is the CNN computation unit layout. Since there are already 50% of the BRAMs and 70% DSPs utilized, the potential of fitting a bigger model for more challenging tasks, such as digit and letter recognition, is limited. Since the inference time now at 25 MHz is 150 us, it is possible to reuse some of the DSPs to still maintain real time processing, which is 16 ms for 60 Hz.

The third issue is about CNN itself. For relatively trivial applications like digit recognition, much fewer bits can be used to achieve the same accuracy. Since our custom neural network was trained using floating point numbers, the degradation was significant when quantized to 8-bit integers and beyond, which was the reason why 16-bit integers were chosen. However, there should exist some packages or methods to either train the neural network using integers or fine tune it to perform better with integers. Using such methods, the precision could potentially be reduced to 8-bit or even 4-bit integers, which shall greatly reduce the use of computation and storage units.

The above are the things that could be done if we could start over, or to be improved for someone taking over.

3 - Description of the System Components

3.1 - Video Capturing and VGA output

The system's video capturing and VGA output components are engineered to work in tandem, leveraging the FPGA's reconfigurability and efficient data handling. The Microblaze is configured to set up the I2C settings, which is a crucial step for initializing and controlling

the camera module. The camera captures high-resolution RGB888 video data, which is then encapsulated into an AXI Stream for reliable and sequential data transfer.

The Video Direct Memory Access (VDMA) plays a role in buffering this stream into the DDR memory, ensuring smooth data transmission by managing the high throughput and latency of video data. The buffered data is then channeled through a Thin Film Transistor (TFT) controller, which converts the AXI Stream into a format suitable for video display.

3.2 - Video Processing and Preparation

3.2.1 Detection Box

After the AXI TFT Controller, the captured frames undergo several steps before reaching the VGA output. The first step is adding the detection box onto the output frame. The AXI TFT Controller IP supplies three signals to help with locating the pixel coordinates. The horizontal and vertical synchronous signals help set the horizontal and vertical counters to zero, and the input valid signal is asserted for each valid pixel in the 640x480 frame. The counters are used to locate a 112x112 detection box in the center of the frame. The pixels around the detection box are changed to green color on the VGA output frame.

3.2.2 Grayscale and Binary Conversion

The 112x112 detected pixels are changed to grayscale by multiplying the RGB factors by 76, 150, and 29, respectively. The sum of these three numbers is 255, effectively shifting the scaled sum by eight bits. Therefore, the output is shifted back eight bits before moving on to the next module.

The binary conversion is applied when the grayscale does not work well under certain lighting situations. The user can change this setting by flipping one of the switches on the board. The binary conversion simply makes all pixels brighter than 50% white, and the rest black.

3.2.3 Downsampling Module

The 112x112 grayscale or binary data is downsampled to 28x28 to fit the input of the CNN module. This is done by sending both the vertical and the horizontal counter value to the downsampling module, if both counter values end with “00”, which means they are divisible by four, then the pixel is passed to the input buffer with a valid signal.

3.2.4 Input Buffer for CNN

The 28x28 pixels from the downsampling module with valid signals are stored in the buffer with a precision of 8 bits. The frame sync signal is used to reset the memory pointer. After all 784 pixels are stored, the buffer will assert a ready signal to the CNN module, and the module will start providing an address to the buffer to request the pixel five at a time. The reason for five pixels at a time is because of the 5x5 kernel size.

3.3 - Convolutional Neural Network

3.3.1 - CNN Overview

The idea of using CNN to recognize single digits is mainly from LeNet from Yann LeCun[1]. We have trained a custom 3-layers CNN with grayscale hand-written digit dataset, MNIST dataset [2], using a 3 layer architecture, with a total of 1158 parameters including kernels, weights and biases. The layer dimensions are as follow:

- Convolutional layer 1: 28x28 inputs, 5x5 kernel, 1 input channel & 4 output channels
- ReLU activation layer
- Max Pooling layer: 24x24 inputs, 12x12 outputs
- Convolutional layer 2: 12x12 inputs, 5x5 kernel, 4 input channels & 4 output channels
- ReLU activation layer
- Max Pooling layer: 24x24 inputs, 12x12 outputs
- Fully-Connected: 64 inputs, 10 outputs

The final test accuracy on the MNIST dataset reached ~97%, which is feasible for hardware implementation which could potentially introduce accuracy degradation

3.3.2 - Overall Structure

The convolutional neural network module is organized as shown in the figure below:

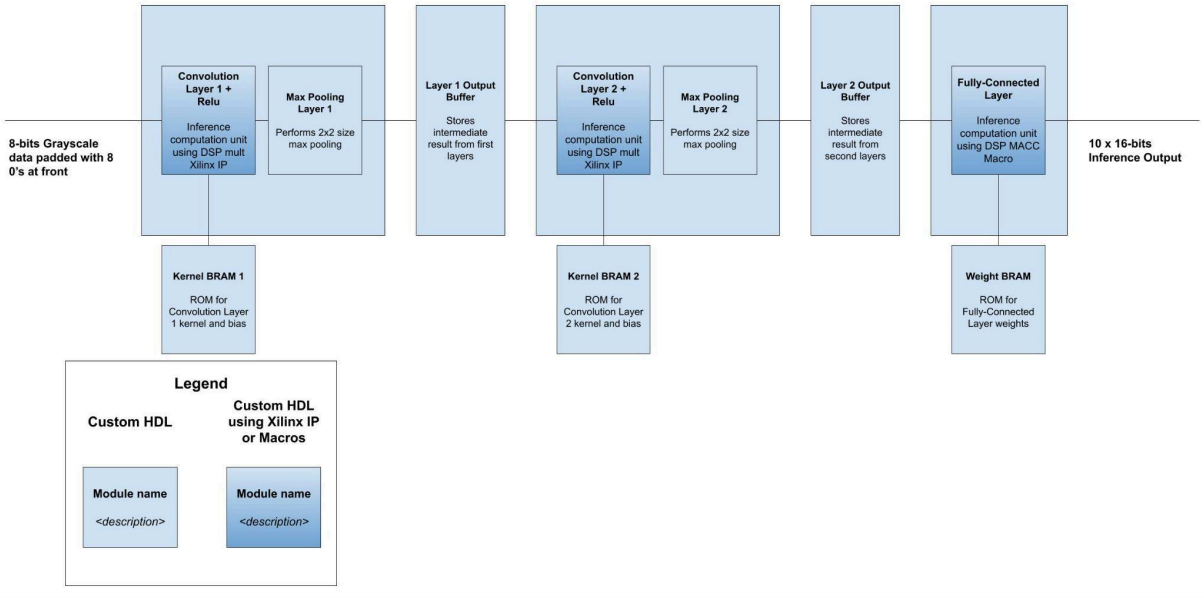

Figure 3. CNN module architecture

To precisely map the CNN architecture onto hardware, there are three computation units: Conv1+MaxPool1, Conv2+MaxPool2, and Fully-Connected Layer. Each computation unit is coupled with a storage unit that provides parameters needed for the computation, in the form of Block RAM. Between each computation unit there is a buffer to store intermediate results , and also helps with increasing throughput.

We chose the modular approach for implementing each layer for easier debugging and better testability, as well as the possibility to experiment with various structures of CNN

3.3.3 - Convolution Layers

The convolution operation can be exploited to benefit heavily from the spatial parallelism of DSP blocks available on the FPGA fabric. We have chosen to implement a more fine-grained parallelism, which is to parallelize the multiplications at each convolution step. For instance, for the 5x5 kernel at time step n, all 25 multiplications are done in parallel. Thus, we stored the input data and feature map into the BRAM in terms of rows, for parallel access of each row to enable parallel computation, and at each clock cycle , a column of 5 values is extracted, stored into a matrix of registers for immediate calculation. When all 5 columns are accumulated , multiplication is performed, followed by a summation. Each stage is pipelined to increase throughput. Matrix of registers can be understood as the window that the kernel can see at that step of sweeping of convolutional operation, therefore it is

implemented in a FIFO style. The ReLU layer can be easily implemented after the summation to check if the value is larger than 0 or not. For the different convolution layers, they all follow the same control and data flow, the main differences lie within the kernel dimensions and the amount of computation for each stage, thus more pipelining stages are needed to account for more strict timing requirement

3.3.4 - Pooling Layers

Due to the sequential nature of yielding output on the convolution layer's end , the max pooling layer processes the input feature map also in a sequential manner. With each input received, it undergoes a series of fixed-point value manipulation to ensure the incoming value is transformed into the format to achieve best accuracy before entering the next computation layer. Then, a counter for both row and columns within this module keeps track of the amount of data values and serves as a index provider, and by manipulating the lower bit of the indices, we perform max pooling in sequential manner

3.3.5 - Intermediate Result Buffers

The Intermediate result buffer is allocated as FPGA block memory following hardware coding style and directive. This buffer is a means to provide a simple handshake protocol between the input and output of this module, ensuring the correct synchronization. It increments counter as it stores value into the memory and provides a signal to indicate that the intermediate feature map is ready .

3.3.6 - Fully-Connected Layer

Since it is directly connected to the processing to be displayed on 7-segments, it is optimal to yield output in the same clock cycle. Therefore, more coarse-grained parallelism is applied. The size-64 input is partitioned into 4 streams of data for multiply-accumulate (MACC) operation. Therefore, for each of the 10 output nodes, 4 DSP blocks are assigned to execute the 4 streams of MACC operations. As result, 16 cycles and some additional pipelining overhead is required to complete 640 MACC computations

3.4 - 7-Segment Output

The output of the CNN are ten 16-bit signed numbers. The output module first takes a few clock cycles to find the max number out of the ten and set the output pin to display the number on the 7-segment respectively.

4 - Schedule

4.1 Proposed Milestones

Milestone	Hardware	Software
Milestone 1	Understand computation and memory limitations of FPGA.	CNN model selection and training
Milestone 2	Set up VGA connections and study camera I2C setup.	Finalize CNN model, per-layer statistic analysis, extract parameter
Milestone 3	Setup camera connection and VDMA.	Implement layer modules(conv, FC), generate testbench for layer unit test
Milestone 4	Finalize VDMA connection, make sure captured video can display on VGA.	test and debug for each layer
Milestone 5	Integrate all layers, test and debug entire CNN module on hardware	
Milestone 6	System integration	
Milestone 7	Buffer week, further enhancements	

4.2 Actual Milestones

Milestone	Hardware	Software
Milestone 1	Understood computation and memory limitations of FPGA.Set up the Vivado project, starting with Microblaze and gradually adding necessary IPs for streaming camera video to output.	Developed a deep learning script using PyTorch and trained a Convolutional Neural Network (CNN) model on the EMNIST dataset, tailored for the 128MB DDR on the Nexys board.
Milestone 2	Set up the camera using the I2C protocol and set up VGA output. Studied and integrated various IPs such as the VTC, TFT, VDMA. Learned about the process for displaying camera video to VGA, implemented Verilog for data conversion.	The CNN architecture was optimized to just 6,000 parameters, requiring about 12,000 bytes of memory, while maintaining high accuracy.
Milestone 3	Integrated camera video input with VDMA, DDR and VGA, overlaying a green detection box on video. Configuration is managed via software using an I2C IP for camera settings.	Implement layer modules(conv, FC), generate testbench for layer unit test
Milestone 4	Capture the detection box data, process the RGB to grayscale, and downsample the data	Complete CNN Implementation (One more Conv layer and one FC layer)
Milestone 5	On-FPGA testing with some pictures taken by the camera Configure BRAM between camera data and CNN input	
Milestone 6	Full system integration ● Camera data to CNN ● CNN to 7-Segment display	

Milestone 7	Buffer week and final demo preparation

4.3 Evaluation

The project maintained a steady course, with the actual milestones mirroring the proposed ones, yet it incorporated additional refinements for a more robust system. Notably, the software component not only met the expected model training but also enhanced the CNN to be leaner on memory usage. These enhancements likely contributed to smoother subsequent phases. Progress in hardware, like integrating specific IPs and achieving the camera-to-VGA output, was more granular than initially outlined, indicating a methodical and precise approach. The incremental improvements in both hardware setup and software optimization led to an effectively integrated system, aligning with the original schedule and setting the stage for the final demonstration.

5 - Design Tree Description

Key files in project folder

Root directory

- CNN_0328_bowen_stream_ila.xpr - Vivado project file
- CNN_0328_bowen_stream_ila.srcs - sources folder
    - constrs_1/new/nexys4ddr.xdc - constraints file
    - sources_1/bd/design1/hdl/design_1_wrapper.v - wrapper for block design
    - sources_1/new - custom verilog modules
        - capture_to_vdma.v
        - downsample.v
        - downscale_to_CNN.v
        - grayscale_bram_freeze.v
        - ov5640_capture.v
        - ov5640_capture_new.v
        - ov5640_powerup.v
        - ov5640_top.v
        - vga_out.v

    - sources_1/imports/sources_1/new - all CNN files are in here
    - sources_1/imports/sources_1/imports/CNN_hw - some later added verilog modules
        - choose_output.v
        - grayscale_rom.v
        - seven_seg.v
- CNN_0328_bowen_stream_ila.sdk/project_1/src/test.c - MicroBlaze program code

6 - Tips and Tricks

Here are some practical tips and tricks for future students. Hope you enjoy digital system design!

1. Plan and Document Thoroughly: Before diving into coding, spend time planning your design. Use block diagrams and flowcharts to outline your project's architecture. Document your design decisions and keep a log of changes as you progress. This documentation will be invaluable for debugging, future enhancements, and for others who may inherit your project.

2. Understand Your Tools: Familiarize yourself with the FPGA development environment, have a comprehensive understanding of the tutorials and assignments. Knowing the ins and outs of your design software can save you a considerable amount of time.

3. Start Small and Scale: Begin with a simple version of your design and test it extensively before adding complexity. Validate each module separately before integrating them into the larger system.

4. Learn to Read Timing Reports: Timing issues can be cryptic and challenging. Understanding how to interpret timing reports and constraints is crucial for ensuring your design meets the required performance.

5. Leverage IP Cores: Don’t reinvent the wheel. Make use of available IP cores for standard functions like communication protocols, memory interfaces, and mathematical operations. This can drastically reduce your development time.

6. Optimize Resource Usage: Be mindful of your FPGA's resources. Look for opportunities to optimize your design for space, whether it's through more efficient coding or by leveraging specific features of your FPGA.

7. Test on Hardware Regularly: While simulations are essential, regularly test your design on actual hardware. Some issues only surface when running on real-world hardware.

References

[1] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

[2] L. Deng, "The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]," in IEEE Signal Processing Magazine, vol. 29, no. 6, pp. 141-142, Nov. 2012, doi: 10.1109/MSP.2012.2211477.