Project 20-1-1-2187

CNN Acceleration and Simulation on an FPGA

Chaim Gruda 312562721

Shay Tsabar 208723627

Instructor: Yoni Seifert

Tel Aviv University

Faculty of Engineering

Contents

List of Figures

List of Tables

List of Graphs

Abstract

In recent years image analyzing and recognition has become a vital feature in ever more applications, ranging from autonomous vehicles, security, healthcare and more. Convolution neural networks (CNN) algorithms are widely used in many such applications since its high accuracy for image recognition. However, with the high accuracy come long computation time and high power consumption. The need to include such image recognition capabilities in embedded systems with tight real-time and power constraints, lead the design for hardware accelerators for CNNs. Such accelerators may be implemented on an FPGA, GPU, or ASIC.

In this project we explored the potential of FPGA-based CNN acceleration, by implementing and simulating the CNN operation on hardware, and comparing it to software-based implementation of the same CNN on different processors, and regarding time, accuracy and power consumption.

The CNN accelerator was synthesized using High Level Synthesis (HLS). The FPGA platform used is the Zynq-7000 SoC based ZedBoard. the processors platforms tested in comparison with the CNN accelerator are the ARM Cortex A9 embedded in Zynq-7000 SoC, and an Intel i7-7500 on a laptop-PC. The CNN was trained using Python Keras AI library and the MNIST dataset.

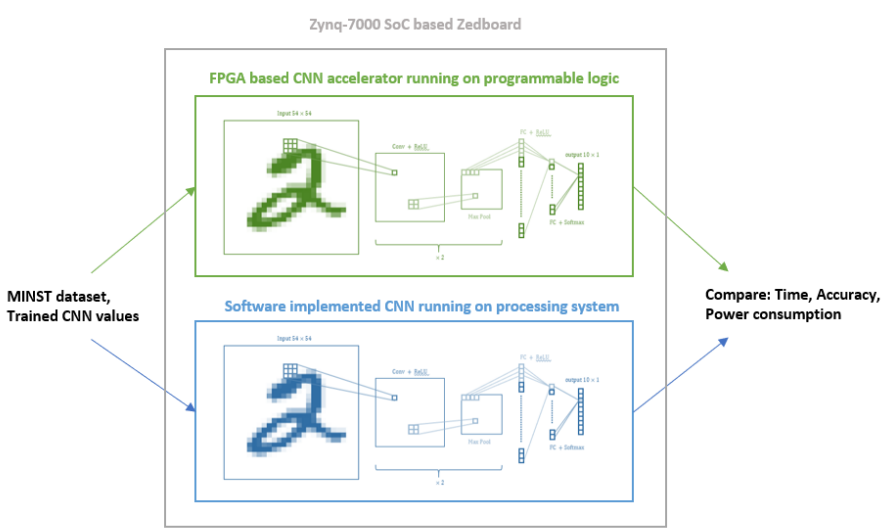


Figure 1: Project Idea ilustration

1 Background

1.1 Neural Networks

Neural networks are a family of computation architectures inspired by the biological nervous system. That system consists of a large amount[1] of neurons, which are electrically excitable cells, that are connected to each other in junctions named synapses[2], forming a massive network. Each neuron receives input signals and produces output signals which branch out and connect to the other neurons. The synapses connecting the neurons influence the transfer of information from one neuron to the other by amplifying, attenuating, or inhibiting the signals transfer. Together, the billions of simple neurons form an incredibly complex interacting network which enables all the brain activity.

The basic building block of an Artificial Neural Network (ANN) is the artificial neuron. The artificial neuron receives several input signals from other neurons. These input signals are multiplied with weights to simulate the synaptic interaction. The weighed input signals are summed, biased, and fed into a non-linear activation function, which produces the neuron's output signal. A neural network is formed by interconnecting many artificial neurons. Usually, the neurons are grouped into layers, and connections are only allowed between neurons of adjacent layers. Such layered connections are called Fully Connected layers.

When neural networks are employed for image-related tasks, their input usually consists of pixel data. Even for images with modest resolutions, the input consists of large amounts of elements (especially when dealing with multiple channel images, such as RGB). Large input data results in amounts of neurons and connections that are a challenge for computing systems, and place a burden on the efficiency, both in time and in power, of analyzing images effectively in such neural networks.

[1] For example: a mouse brain has approx. 70 million neurons, human brain approx. 100 billion neurons
[2] Mouse: approx. 1 trillion synapses, human: approx. 125 trillion synapses

1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a special class of neural networks, suited for operation on multi-dimensional input data such as images and audio. The idea behind the CNN is the locality of information in the input data, which means that important information can be captured from local neighborhood relations. In an image for example, strong contrasts indicate edges, aligned edges result in lines, combined lines can result in circles and contours, circles can outline a wheel and multiple nearby wheels can point to the presence of a car. This locality of information in images is exploited in convolutional neural networks by adding layers before the fully connected layers, namely convolutional layers, and pooling layers.

The purpose of convolutional layers (CONV) and pooling layers (POOL) is to reduce the spatial data into smaller data sets, that contain the most distilled data from within the input data. After that process is done, the refined data can be passed through fully connected layers (FC), which will now be much smaller and tailored for the purified information. A CNN may have multiple CONV and POOL layers.

- **Convolutional layer** – This layer preforms the mathematical convolution. For 2D image this operation is described by Eq. (1):

$$g(x,y) = \sum_{s=0}^{a} \sum_{t=0}^{b} \omega(s,t) \cdot f(x,y) \tag{1}$$

  The operation basically slides filter of size $a \times b$, where each pixel in the filter has its own weight ω, over the 2D image, and computing the sum of the product of all overlapping pixels. The filter is also known as a kernel, and the convolution output is called a feature map. The operation is described in **Figure 1**.

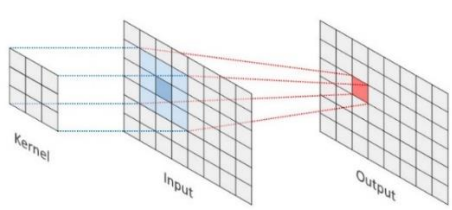


Figure 2: 2D convolution illustration

- **Pooling layer** – This layer reduces the spatial area of the given input, usually being the output of a convolutional layer. This is done by selecting the max value or calculating the average value from a 2D section. These are called Max-Pool and Avg-Pool, respectively. Max polling is visualized in **Figure 2**.

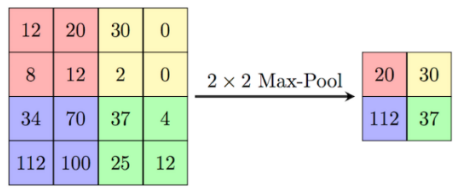


Figure 3: Max pooling illustration

- **Fully Connected layer** – The operation of FC layer can be expressed by Eq. (2). Input features N_i are fully connected to an output feature vector N_o by weights $(N_i \times N_o)$. The N_i and N_o are the lengths of the input and output feature vector and the number of weights is $N_i \times N_o$. $Bias_o$ is the corresponding bias term of the oth output feature vector. FC layers usually turn the input feature maps into a feature vector.

(2) $$Out_o = \sum_{i=1}^{N_i} In_i \cdot W_{io} + Bias_o$$

- **Activation Layer –** This layer introduces non-linearity to the CONV and FC outputs. The non-linearity increases the power of deepening the network. The most common activation functions used in CNNs are the Rectified Linear Unit (ReLU) function expressed in Eq. (3) and seen in **Figure 3**, and the SoftMax function expressed in Eq (4) and seen in **Figure 4.**

(3) $$f(x) = \max(0, x)$$

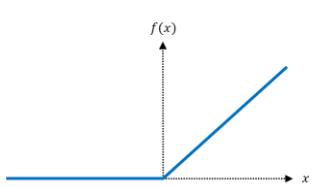


Figure 4: ReLU function

(4) $$f(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_i}}$$

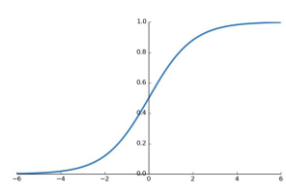


Figure 5: SoftMax function

1.3 CNN Acceleration

As CNNs achieve very high accuracy in image understanding and classification, they get embedded in more applications, some of which require tight real-time constraints. This leads to the need to accelerate the CNN calculations.

Most of the computations done in the CNN is in the convolutional layers, around 85-90%[3]. Since all layers: CONV, POOL and FC, basically deal with matrices, the software implementation of those layers consists of nested loops. This means all computations are done serially, and thus the time the network operates on a given output may be very long.

To reduce the computation time, and thus to accelerate the CNN performance, the different layers can be implemented in hardware. Since hardware can perform computations in parallel, this basically allows to expand the software loops so that the layer operations be performed in parallel.

Potential hardware platforms that can be used for accelerating the CNN are DSPs, GPUs, ASICs, and FPGAs. The advantages for using an FPGA are discussed in the following section.

1.4 Field-Programmable Gate Arrays

Field-Programmable Gate Arrays (FPGAs) are semiconductor devices consisting of a 2D array of configurable logic blocks which are connected via programmable interconnects. The interconnect can be thought of as a network of wire bundles running vertically and horizontally between the logic slices, with switchboxes at each intersection. Modern high-end FPGA generations feature hundreds of thousands of configurable logic blocks, and additionally include an abundance of hardened functional units which enable fast and efficient implementations of common functions. The logic blocks, the fixed-function units as well as the interconnect are programmed electronically by writing a configuration bitstream into the device. The configuration is typically held in SRAM memory, and the FPGAs can be reprogrammed many times.

The advantage of FPGA-based systems over traditional processor-based systems including GPUs, is the availability of freely programmable general-purpose logic blocks. These can be

[3] Science Direct – Convolutional Neural Networks

arranged into specialized accelerators for very specific tasks, resulting in improved processing speed, higher throughput, and power savings. This advantage comes at the price of reduced agility and increased complexity during the development, where the designer needs to carefully consider the available hardware resources and the efficient mapping of his algorithm onto the FPGA architecture. Further, some algorithmic problems do not map well onto the rigid block structures found on FPGAs. Compared to ASIC (Application-Specific Integrated Circuits) implementations, the FPGA advantages are re-programmability, that allows fast prototyping and short development cycles.

2 Design

2.1 CNN Model

The implemented CNN model is a classification network, which accepts a 2D black and white input image of size 54x54 pixels, and outputs a classification vector of size 10, i.e., a probability distribution vector. The network consists of 6 layers:

Layer name	Operation	Input size	Output size
Conv0	3x3 convolution + ReLU activation	54x54	52x52
Pool0	2x2 max pooling	52x52	26x26
Conv1	3x3 convolution + ReLU activation	26x26	24x24
Pool1	2x2 max pooling	24x24	12x12
Fc0	Fully connected + ReLU activation	144	50
Fc1	Fully connected + SoftMax activation	50	10

Table 1: CNN layers

The model is illustrated in **Figure 10**, with a simulated input from the MNIST dataset. See more about that in the simulation section (4.1):

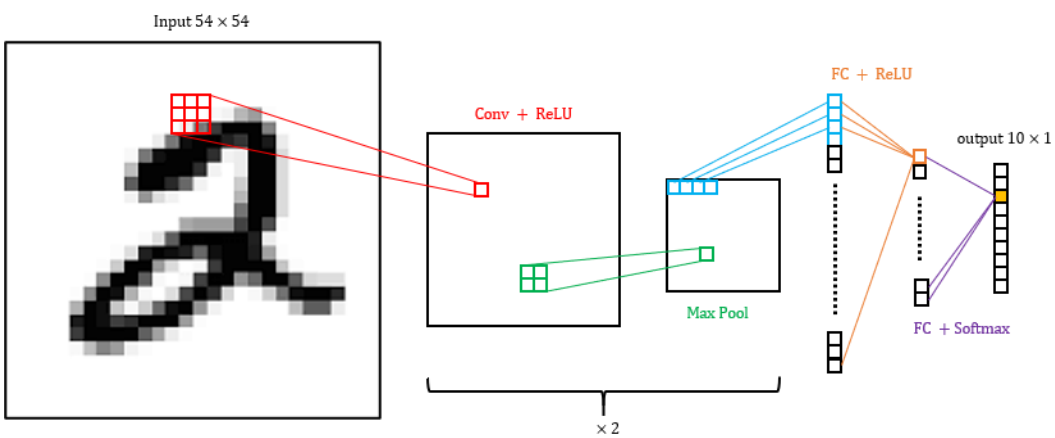


Figure 6: Implemented CNN model

2.2 ZedBoard FPGA

The FPGA platform used for this project is the Xilinx Zynq-7000 SoC based ZedBoard, seen in **Figure 6**. The Zynq-7000 contains two sub-systems:

1. Processing System (PS) – an ARM Cortex A9 CPU.
2. Programmable Logic (PL) – Artix-7 FPGA with 6.6 million logic cells.

The two subsystems are utilized and work together in the CNN implementation. The PL containing the CNN acceleration cores, and the PS running the cores drivers – initializing and configuring the CNN, feeding it data, and receiving the results. The communication between the systems is by AXI interface. Both systems also have access to the DDR3 memory using an AXI-DMA core controlled by the PS. The communication with the PS is done using an SD card reader and a UART serial port.

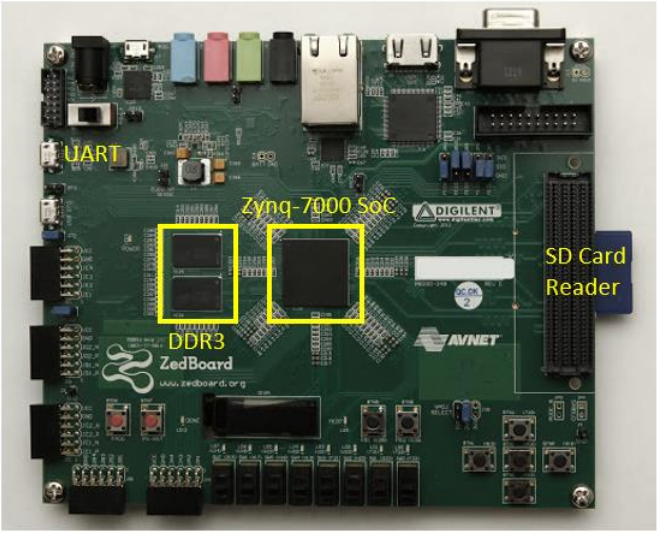


Figure 7: ZedBoard

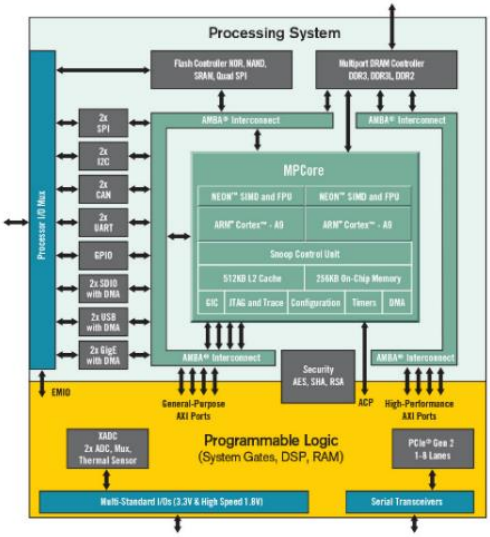


Figure 8: Zynq-7000 SoC

2.3 Modularity

Each CNN layer is represented in hardware by a Processing Element (PE). Each PE is tailored to do specific calculations. This design allows to simply synthesize different CNN models other than the specific one we focused on in this project, by simply connecting different PEs to create a new model.

Potential hardware platforms that can be used for accelerating the CNN are DSPs, GPUs, ASICs, and FPGAs. The advantages for using an FPGA are discussed in the following section.

2.4 Sliding Window

Since the input data is held in PS memory as an ordered array, when preforming CONV or POOL operations on the data, out of order pixels need to be extracted and operated on. Preforming the extraction in software is quite simple since the array is stored in RAM. In HW this is more complex, and thus there are 2 possible solutions. The simple one is to utilize the SW to extract the needed data per operation and feed it to the HW to calculate. However, this means data pixels are passed from SW to HW multiple times, causing a bottle neck that may reduce the acceleration affect.

The more advanced way to perform the operation, is to have a sliding window mechanism implemented in HW. That way the ordered array can be passed in an ordered stream to HW, and HW will know what pixels to operate on at any given cycle. The sliding window concept is show in **Figure 8** and the neighborhood extraction logic is shown in **Figure 9**.

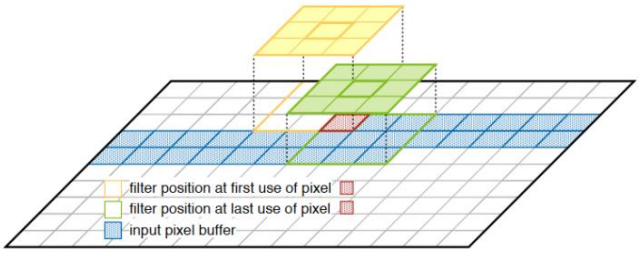


Figure 9: Pixel ruse

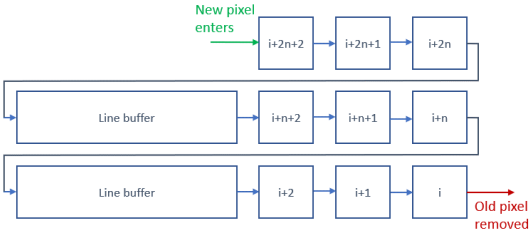


Figure 10: Neighborhood extraction

The main advantage of the sliding window mechanism is that each pixel is passed to HW only one singe time. Another advantage is that the CONV and POOL layers can start

preforming the operation on the full image, even as soon as the first needed data pixels arrive and pass their results to the next layers, which in their turn can start calculating even while prior layers are still calculating. This increases the acceleration and allows high data throughput in the CNN.

2.5 Fixed Point Calculations

To simplify HW implementation, data in HW is represented in fixed point format. Out of 32 bits of word length, 10 bits are used to represent the fraction (formal notation: Q32.10). This gives a resolution of $\frac{1}{2^{10}} = 0.0009765625$. The downside of using fixed point is the loss of accuracy in calculations, which may lead to less accuracy of the accelerated network.

2.6 High Level Synthesis

High level synthesis (HLS) is a design process in which a high-level description of a design is compiled into RTL implementation that meets specified constraints. The design is written in C and C++, rather than RTL cycle by cycle specification. The higher languages allow the use of more advanced language features like loops and arrays. User specified constraints help HLS construct the desired architecture. These constraints include memory architecture that specify how multi-dimensional arrays are mapped to memories, Interface constraint that specify what protocols, ports, and handshake logic to create at each input and output.

The output of HLS includes the RTL implementation that contains the data path, control logic, interfaces to I/O and memories. HLS also outputs a report on performance bottlenecks and hardware costs of the design. Benefits of designing at a high level of abstraction:

- Allows focus on designing core functionality, not implementation details.
- Easily explore different architectures and evaluate algorithmic changes.
- Easily retarget for different hardware constraints or performance from the same input description.

3 Implementation

3.1 Hardware CNN Process

The block diagram in **Figure 11** describes the systems operation. Control path is marked in black arrows, and data path is in red. Full description of the process flow is described below the figure.

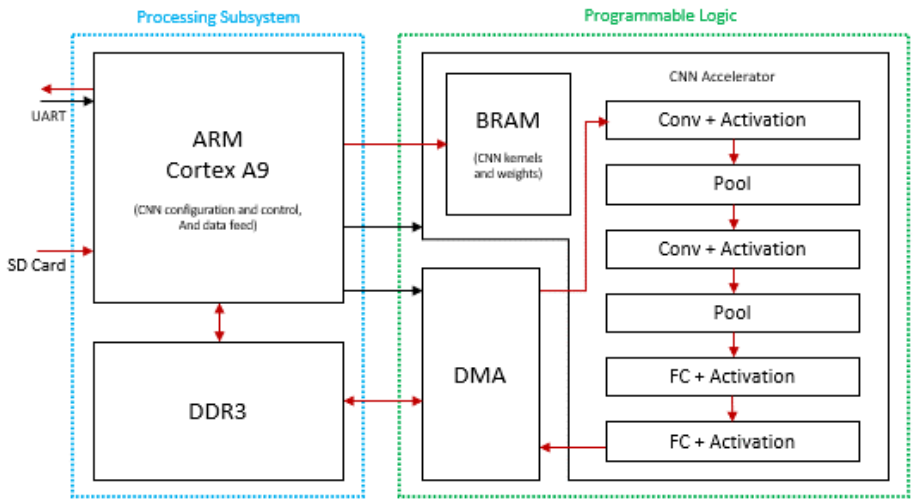


Figure 11: System block diagram (black: control, red: data)

The PS is responsible of handling I/O, configuring the CNN cores, reading data from the SD card, and transferring it to the CNN core by programming the DMA or by direct writes to the BRAM. The input to the PS is through a UART serial port that is connected to a terminal. Data input to the PS is through an SD card reader that's part of the ZedBoard. The PS directs the CNN output to the UART terminal.

At startup, the PS reads CNN kernels and weights from a specific path in the SD card, and configures the PEs in the CNN core. After configuration is completed, the PS loads an image from the SD card and stores it in the DDR memory. After that the PS sends start signals to the CNN core, and programs the DMA to transfer the image from the DDR to into the first CNN layer.

As soon as the neighborhood-extraction buffers of the first CONV PE are filled, the CONV operation starts, and the results are streamed to the next PEs, which also start operating as soon as their buffers are full. The last layer streams the data back to the PS using the DMA which is programmed by the PS to get read the fetch the data back to the DDR.

Once the last CNN layer is done calculating, and has streamed all results to the PS, a done signal is sent to the PS, that may now fetch a new image from the SD card and start the process again.

3.2 Processing Elements

Each of the layers consists of BRAM memory to hold kernels and weights. The CONV and POOL layers each consists of a sliding window mechanism and then a calculation unit to perform the convolution or pooling. The CONV blocks support 3x3 convolutions and can be configured to apply ReLU activation on the output. The POOL layers support 2x2 max or average pooling.

The FC blocks consist of large BRAM to hold the weights matrices, as well as computation unit and a ReLU activation possibility. Since there are limits on the available resources in the FPGA, especially limited amount of DSP48 cells that perform addition and multiplication, the FC blocks can calculate the output in a few stages depending on the input vector size.

The blocks are generated using tcl scripts that automate the HLS code generation. this allows creating CONV and POOL blocks of different input and kernel sizes in simple and fast way. These scripts accelerate the creation of different CNN models, as each model has specific input and operation dimensions.

3.3 System Block Design

The Vivado block design is shown in **Figure 12**. The Zynq Processing system, DMA engine and CNN cores are marked. **Figure 13** shows the expanded CNN core.

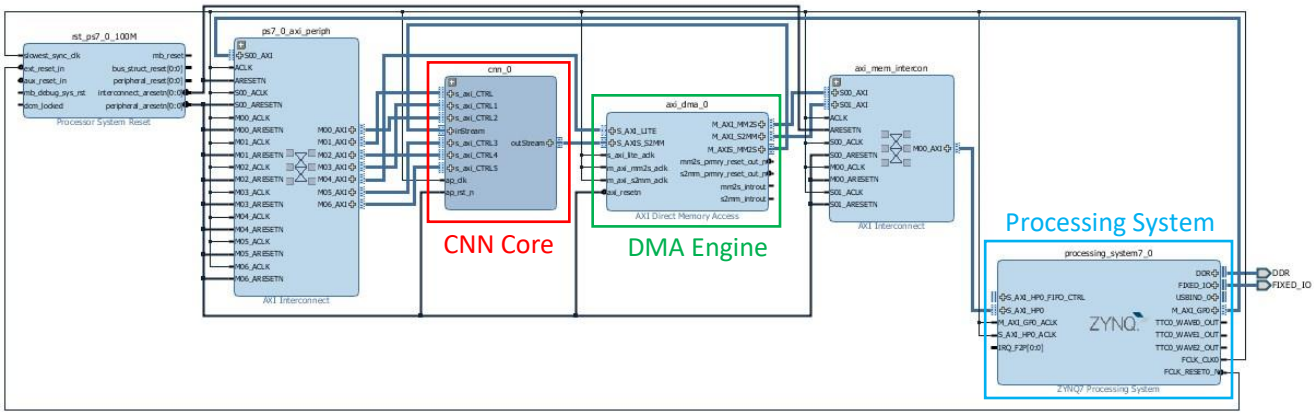


Figure 12: System block design

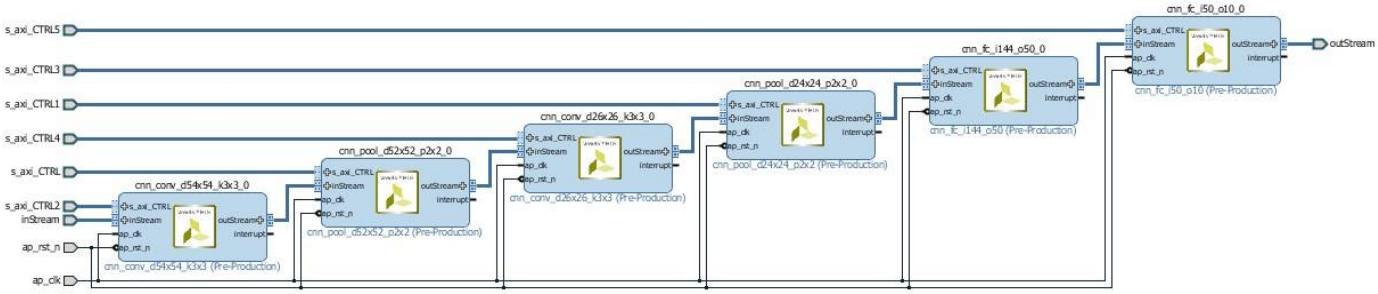


Figure 13: CNN core block design

The Vivado implemented design is shown in **Figure 14.** the implemented design is the mapping of the synthesized design to the available logic resources onboard the Zynq-7000 SoC.

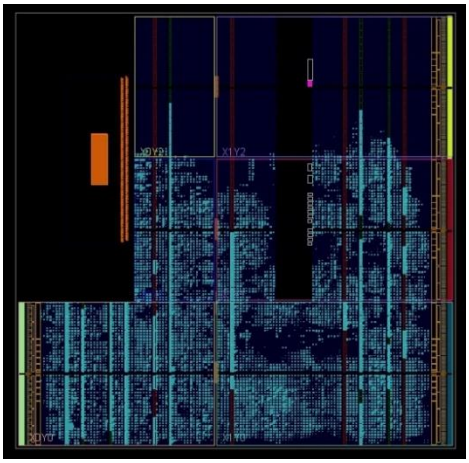
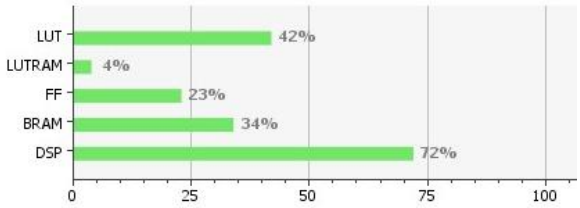


Figure 14: Implemented design

3.4 Hardware Utilization

The utilization of the Zynq-7000 SoC resources are given in **Graph 1** and **Table 2** that were generated by Vivado. The most used resource is the DSP48 logic cell, which performs the addition and multiplication operations of the CNN.



Graph 1: FPGA Utilization

Resource	Utilization	Available
LUT	22562	53200
LUTRAM	764	17400
FF	24825	106400
BRAM	47	140
DSP	159	220

Table 2: FPGA Utilization

3.5 Software

The software for this project consists of several modules which can be compiled for either the ZedBoard processing system bare-metal (i.e., no operating system) platform, as well as any Linux based operating system PC.

The modules consist of the software implementation of the CNN, hardware control modules for controlling and configuring the hardware cores, and data handling modules. Data handling includes also the FatFS free software module, which allows accessing the SD card as a file system when running code with no underlying OS.

HAL (hardware arbitration layer) header files that define hardware addresses and various register I/O operations were generated by Vivado tools. Other modules such as the DMA controller and PS configuration modules and APIs were included from the Vivado SDK libraries.

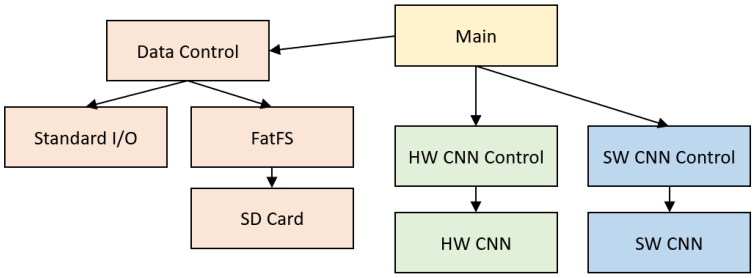


Figure 15: Software block diagram

4 Results

4.1 Simulation

To test the advantage of the implemented CNN accelerator, a simulation for the system was performed. The CNN model was trained with the MNIST data set, which contains 60,000 labeled images of handwritten digits from 0 to 9. Out of these images, 50,000 were used to train the model, and 10,000 were used to test it.

The CNN model was compiled in python using the Keras AI library and was trained with the MNIST data set to achieve an accuracy of 96.08%. After training, a python script extracted the MNIST images and the trained CNN kernels and weights into csv files, so that they can be easily parsed using C code.

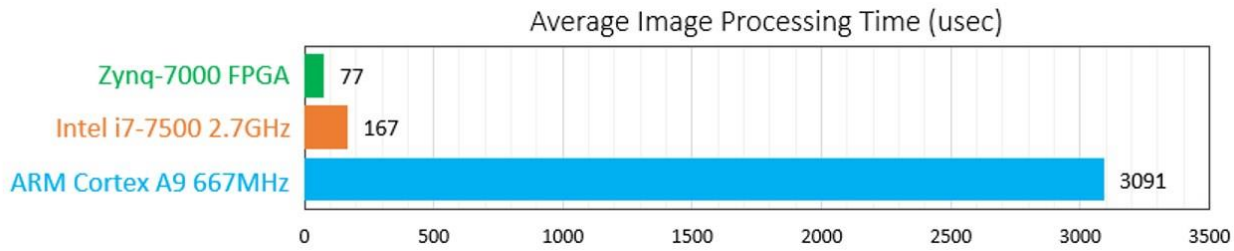
The csv files were loaded into an SD card, and using the ZedBoard integrated SD card reader and the FatFS free software module, the SD was mounted and accessed from within the processing system.

After the software loaded the csv image files, they were passed once to the CNN hardware accelerator, and once to the software CNN module, and various statistics were gathered, such as classification correctness, classification certainty and processing time

The statistics were accumulated over the 10,000 test images, and the following sections show the processed results.

4.2 Time

To quantify the performance and efficiency of the hardware accelerator, the CNN model was tested on 3 different platforms. **Graph 2** shows the average processing time it took from the moment an input image entered the network and until the classification result was outputted, for the different platforms. The first row shows the results of the HW accelerator, the second row shows the software implementation run time on an Intel i7 laptop PC running Linux OS, and the same software implementation on the bare metal ARM Cortex A9 (the ZedBoard PS).



Graph 2: Time results

4.3 Accuracy

Table 3 shows the network accuracy, i.e., the network's success rate in correct classification of the input images, and the networks certainty in the correctness of the classification, while performing calculations with fixed-point and floating-point arithmetic.

Fixed-point arithmetic was used in the HW implementation (see section 2.5), and floating-point arithmetic was used in software implementation.

	Floating point (FP64)	Fixed point (Q32.10)
Accuracy	96.03%	96.02%
Certainty	98.83%	98.82%

Table 3: Precision comparison

4.4 Power

Table 4 shows the estimated power consumption of the PL and PS, gathers by Vivado power estimation tool. For more details see appendix. The table shows total power and dynamic power, which is the power consumed when the network is active.

	Processing System	Programmable Logic
Dynamic Power	77%	23%
Total Power	71%	29%

Table 4: Power estimation consumption

5 Discussion

5.1 Conclusions

Based on the results we conclude the following:

- **Run time:** The results show that running the CNN HW accelerator processed images about 40 times faster than running the software implementation on the embedded ARM processor (ZedBoard PS), and about 2.5 times faster than the high-end Intel processor. Based on these results we conclude that hardware implementations of CNNs have great benefits regarding processing time.

  This conclusion may have a great impact when considering adding CNNs to real-time systems that have limited processing powers, such as various embedded systems.

- **Accuracy:** From the results we can see that the use of fixed-point arithmetic resulted in a 0.01% loss of accuracy and certainty compared to the accuracy achieved when using floating-point arithmetic. However, this loss is negligible as it is several orders of magnitude smaller than the accuracy percentages. Moreover, even if the loss would have been bigger, some usages of CNNs may trade some loss in the CNN accuracy in exchange for significantly shortening the processing time.

- **Power:** The estimated power consumption of the ZedBoard PS is greater than the power consumed by the PL: 3.3 times more in dynamic power, and 2.4 times more in total power. From this we can conclude that dedicated processing elements are more power efficient compared to a CPU.

To summarize, the hardware accelerator does indeed shorten the calculation and run time, and consumes less power compared to the same network that runs on a processor. In addition, the accuracy loss when using fixed-point arithmetic (as the HW accelerator does) is negligible.

5.2 Future work

Following the conclusions, we can say that the project achieved its primary goals – acceleration, accuracy, and power reduction. However, it is possible to improve the accelerator, and to delve deeper into the following:

- The CNN HW Core currently supports only single channel images. Adding hardware that could handle multi-channel images (such as RGB) would most probably show even better acceleration results compared to software. This would also allow to simultaneously pass several filters on a single channel input image, adding more free variables for the network to train, increasing accuracy.
- The Current software implementation of the network is single threaded. Adding multi-threading to the software may allow accelerating the software implementation as parallelization is introduced. As The ZedBoard PS includes two ARM Cores, also the PS might show better time performances and give more insight when comparing the HW accelerator to multi-core CPUs and multi-threaded SW CNN implementations.
- The current fixed-point scheme uses 32-bit long words, out of which 10 are used for the fraction. As we saw in the results this allowed keeping the accuracy of the CNN HW core. It would however be interesting to test the systems with smaller fixed-point schemes, specifically schemes that use shorter word lengths. For example, using 16-bit long words would multiply the systems throughput, as less data will need to be transferred around the system. This will most probably reduce processing time even more but may cause the accuracy loss to be greater as well. These tests may add more insights to the time and accuracy conclusions.

6 References

[1] David Gschwend "ZynqNet: An FPGA-Accelerated Embedded Convolutional Neural Network", Department of Information Technology and Electrical Engineering, ETH Zurich, Switzerland, 2016.

[2] Wenquan Du, Zixin Wang and Dihu Chen "Optimizing of Convolutional Neural Network Accelerator", Sun Yat-Sen University, Guangzhou, China, 2018.

[3] Kingshuk Majumder and Uday Bondhugula "A Flexible FPGA Accelerator for Convolutional Neural Networks", Dept of CSA, Indian Institute of Science, Bengaluru, India, 2019.

[4] Reinier Kruisbrink "An analysis of partial reconfiguration of convolutional neural networks in FPGAs", Faculty of Science, University of Amsterdam, Amsterdam, Netherlands, 2020.

Appendix

1. Power setting for Vivado power estimation tool:

Device	
Part:	xc7z020clg484-1
Temp grade:	commercial
Process:	typical
Characterization:	Production (v1.0, 2012-07-11)

Environment	
Output Load:	0 pF
Ambient temperature:	25.0 °C
Airflow:	250 LFM
Heat sink:	none
ϑSA:	0.0 °C/W
Board selection:	medium (10"x10")
Number of board layers:	8to11 (8 to 11 Layers)
ϑJB:	7.4 °C/W
Board temperature:	25.0 °C

2. Power consumption Hierarchy:

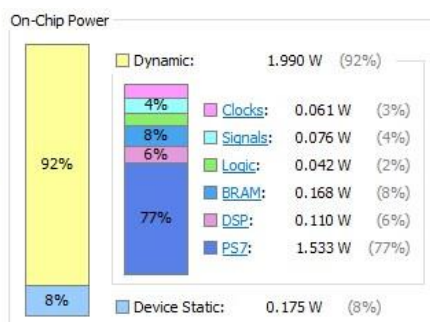
Utilization	Name	Clocks (W)	Signals (W)	Data (W)	Clock Enable (W)	Logic (W)	BRAM (W)	DSP (W)	PS7 (W)	Processor (W)	Memory Interface (W)	I/O Interface (W)	PLLs (W)	AXI (W)
1.99 W (92% of total)	design_1_wrapper													
1.99 W (92% of total)	design_1_i (design_1)	0.061	0.076	0.074	0.002	0.042	0.168	0.11	1.533	0.277	0.889	0.014	0.35	0.006
1.534 W (71% of total)	processing_system7_0 (design_1_...	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	1.533	0.277	0.889	0.014	0.35	0.006
0.246 W (11% of total)	cnn_fc_i144_o50_0 (design_1_cn...	0.028	0.034	0.034	0.001	0.019	0.112	0.052	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.097 W (4% of total)	cnn_fc_i50_o10_0 (design_1_cnn...	0.007	0.014	0.014	<0.001	0.009	0.046	0.021	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.039 W (2% of total)	cnn_conv_d26x26_k3x3_0 (desig...	0.006	0.009	0.009	<0.001	0.003	0.002	0.018	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.038 W (2% of total)	cnn_conv_d54x54_k3x3_0 (desig...	0.006	0.009	0.009	<0.001	0.003	0.002	0.018	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.013 W (1% of total)	axi_dma_0 (design_1_axi_dma_0_0)	0.006	0.003	0.003	<0.001	0.003	0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.007 W (1% of total)	cnn_pool_d52x52_p2x2_0 (design...	0.002	0.002	0.002	<0.001	0.002	0.002	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.007 W (1% of total)	cnn_pool_d24x24_p2x2_0 (design...	0.002	0.002	0.002	<0.001	0.001	0.002	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.005 W (<1% of total)	ps7_0_axi_periph (design_1_ps7_...	0.002	0.002	0.002	<0.001	0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.005 W (<1% of total)	axi_mem_intercon (design_1_axi_...	0.004	0.001	0.001	<0.001	0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	rst_ps7_0_100M (design_1_rst_ps...	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001

3. Power summary:

Power estimation from Synthesized netlist. Activity derived from constraints files, simulation files or vectorless analysis. Note: these early estimates can change after implementation.

Total On-Chip Power:	**2.164 W**
Junction Temperature:	**50.0 °C**
Thermal Margin:	35.0 °C (2.9 W)
Effective ϑJA:	11.5 °C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity



4. Utilization hierarchy:

Name	Slice LUTs (53200)	Slice Registers (106400)	F7 Muxes (26600)	F8 Muxes (13300)	Block RAM Tile (140)	DSPs (220)	Bonded IOPADs (130)	BUFGCTRL (32)
design_1_wrapper	22562	24825	431	126	47	159	130	1
design_1_i (design_1)	22562	24825	431	126	47	159	0	1
axi_dma_0 (design_1_a...	1732	2227	1	0	2	0	0	0
axi_mem_intercon (desi...	1174	1413	0	0	0	0	0	0
cnn_conv_d26x26_k3x3...	2336	2297	2	0	2	27	0	0
cnn_conv_d54x54_k3x3...	2359	2307	2	0	2	27	0	0
cnn_fc_i50_o10_0 (desi...	3261	2850	64	0	11	30	0	0
cnn_fc_i144_o50_0 (des...	9341	11429	353	126	26	75	0	0
cnn_pool_d24x24_p2x2...	757	760	2	0	2	0	0	0
cnn_pool_d52x52_p2x2...	781	767	2	0	2	0	0	0
processing_system7_0 (...	112	0	0	0	0	0	0	1
ps7_0_axi_periph (desig...	690	735	5	0	0	0	0	0
rst_ps7_0_100M (design...	19	40	0	0	0	0	0	0

5. Time summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	2.277 ns	Worst Hold Slack (WHS):	0.045 ns	Worst Pulse Width Slack (WPWS):	3.750 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	69947	Total Number of Endpoints:	69947	Total Number of Endpoints:	25859

All user specified timing constraints are met.