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

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 high computation time and 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.

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.

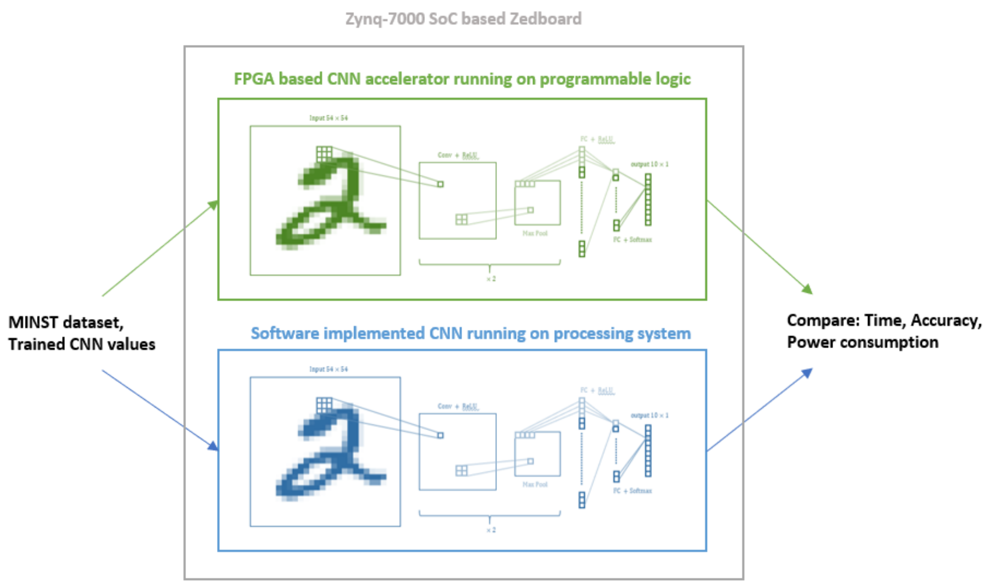

Zynq-7000 SoC based Zedboard
FPGA based CNN accelerator running on programmable logic
Software implemented CNN running on processing system
MINST dataset,
Trained CNN values
Compare: Time, Accuracy,
Power consumption

Figure 1: Project Idea ilustration

Contents

1 Background 4
1.1 Neural Networks 4
1.2 Convolutional Neural Networks 5
1.3 CNN Acceleration 7
1.4 Field-Programmable Arrays 7
2 Design 9
2.1 Fixed Point Calculations 9
2.2 Sliding Window 9
2.4 Data Flow 10
2.3 Parallelization 10
3 Implementation 10
3.1 High Level Synthesis 10
3.2 ZedBoard FPGA 11
3.2 Software 13
3.2 Simulation 14
4 Results 14
4.1 Time comparison 14
4.2 Precision comparison 15
4.3 Power comparison 15
5 Discussion 15
5.1 Conclusions 15
5.2 Future work 15
6 References 15

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 a need with 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 2D input data such as images. The idea behind the CNN is the locality of information in images, which means that important information in images can be captured from local neighborhood relations. 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 new 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 of 2D images into smaller data sets, that contain the most distilled data from within the image. 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 2D convolution. This operation is described by Eq. (1):

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

  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**.

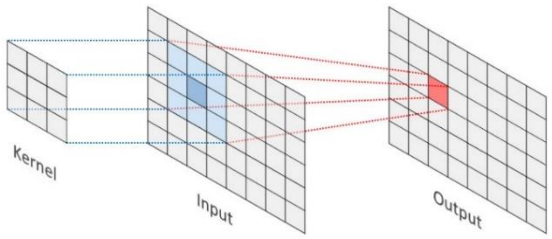


Figure 2: 2D convolution illustration

- **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 can be visualized in **Figure 2**.

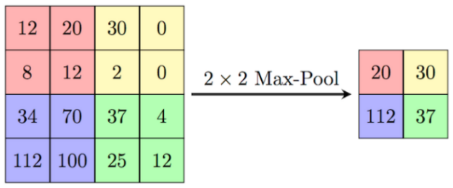


Figure 3: Max pooling illustration

- **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.

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

- **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.**

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

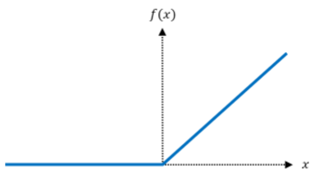


Figure 4: ReLU function

$$f(x)_\mathrm{i} = \frac{\mathrm{e}^{\mathrm{x}_\mathrm{i}}}{\sum_{\mathrm{j}=1}^{\mathrm{K}} \mathrm{e}^{\mathrm{x}_\mathrm{i}}} \qquad (4)$$

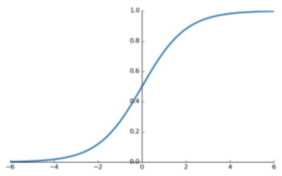


Figure 5: Softmax function

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 a CONV single operation can be done in a single clock cycle.

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 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.

[3] link

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 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 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.

2.2 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 X**.

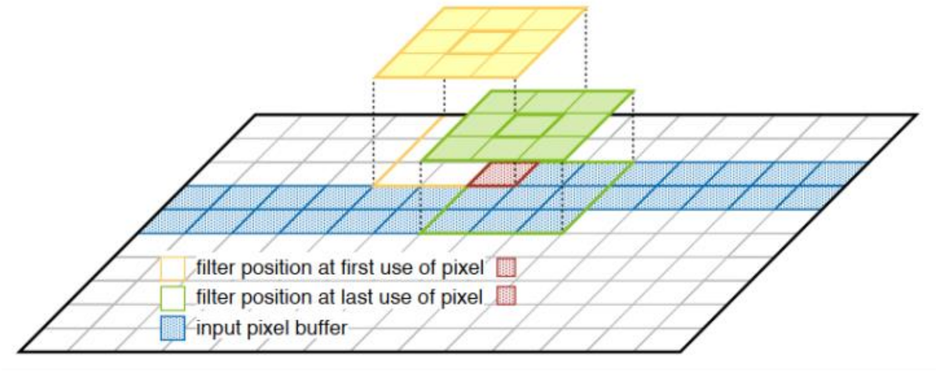

filter position at first use of pixel
filter position at last use of pixel
input pixel buffer

Figure 6:

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 affect and allows high data throughput in the CNN.

2.4 Data Flow

2.3 Parallelization

3 Implementation

3.1 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.

3.2 ZedBoard FPGA

The FPGA platform used for this project is the Xilinx Zynq-7000 SoC based ZedBoard, seen in **Figure X.** 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.

A Full diagram of the Zynq-7000 is shown in **Figure X.**

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.

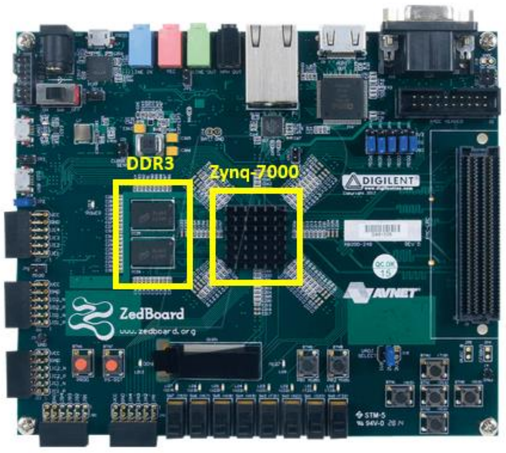


Figure 7: ZedBoard

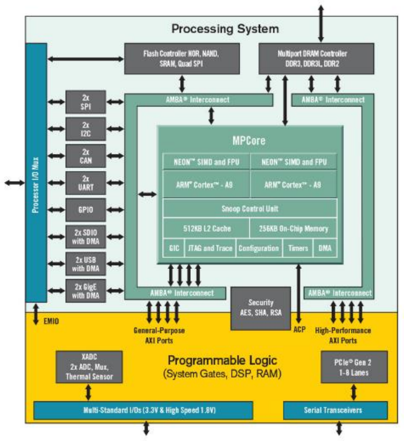
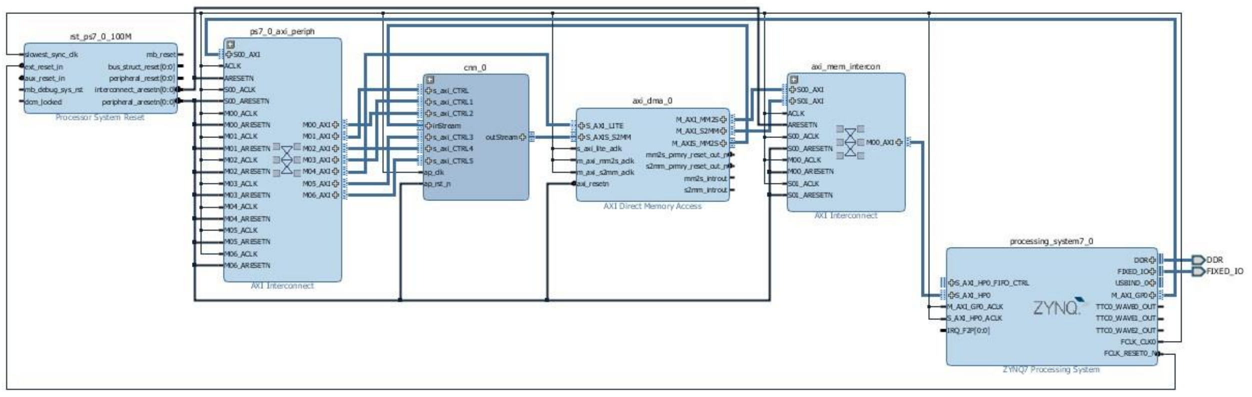
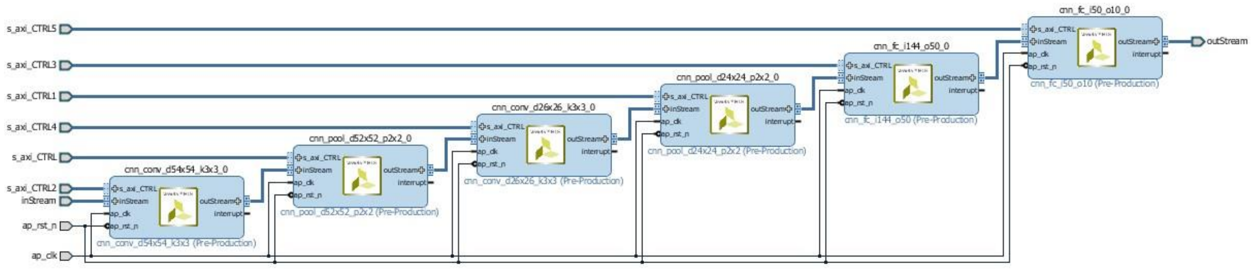
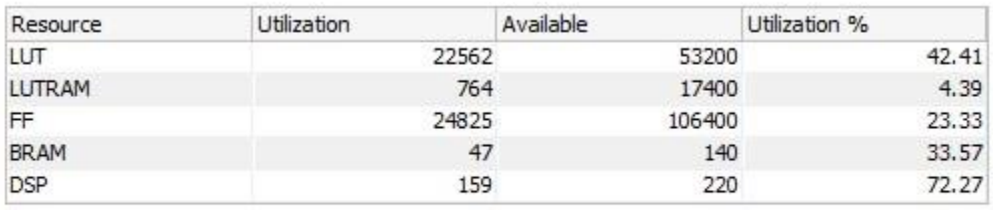


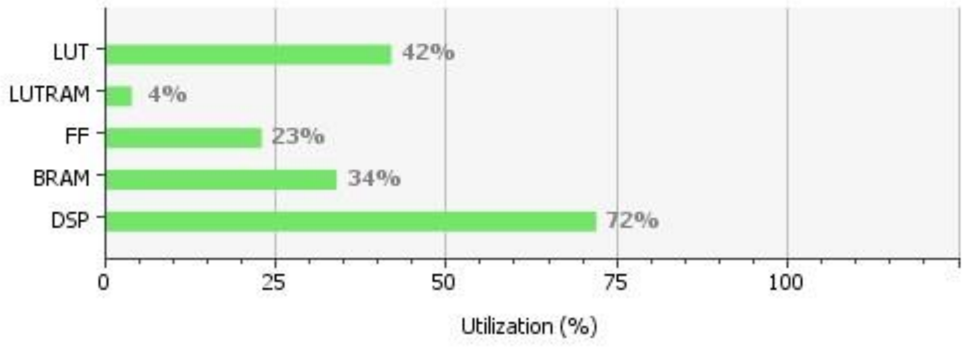
Figure 8: Zynq-7000 SoC diagram

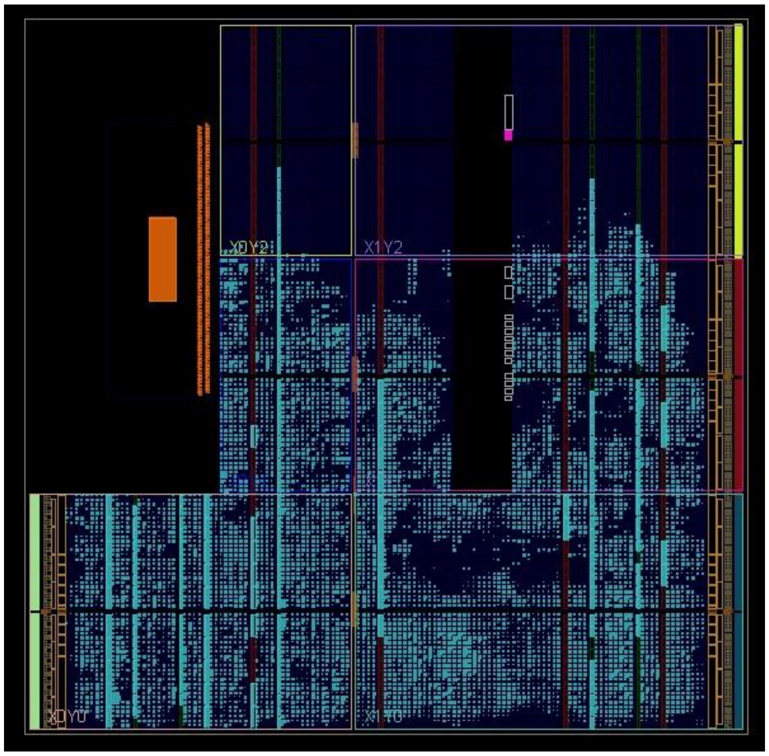






Resource	Utilization	Available	Utilization %
LUT	22562	53200	42.41
LUTRAM	764	17400	4.39
FF	24825	106400	23.33
BRAM	47	140	33.57
DSP	159	220	72.27





3.2 Software

3.2 Simulation

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

The CNN model was compiled in python using the Keras library and was trained with the data set to achieve an accuracy of 96.08%. To be able to use the MNIST data set for the software and hardware implementations, a python script extracted the images into csv files, that can be easily parsed using C code.

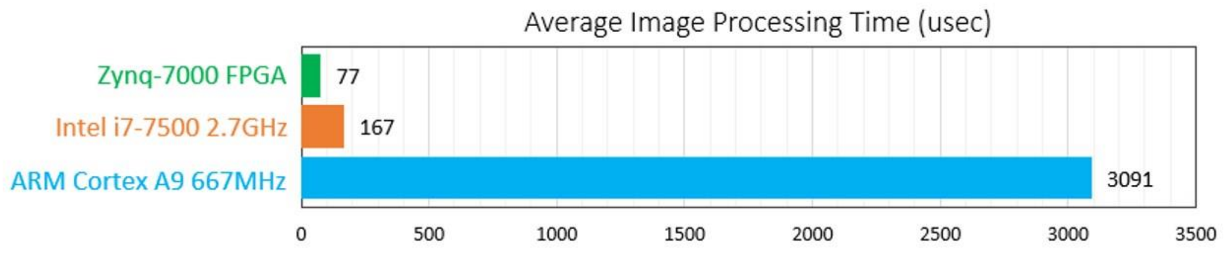
The labeled csv files were loaded into an SD card. Using the ZedBoard integrated SD card reader and the FatFS free module, the SD was accessed from within the processing system.

After loading each csv file, it was passed once to the hardware accelerator, and once to the software CNN module, and time till the CNN output a result was measured, as well as other statistics, such as was the image classified correctly, and at what certainty was the classification done.

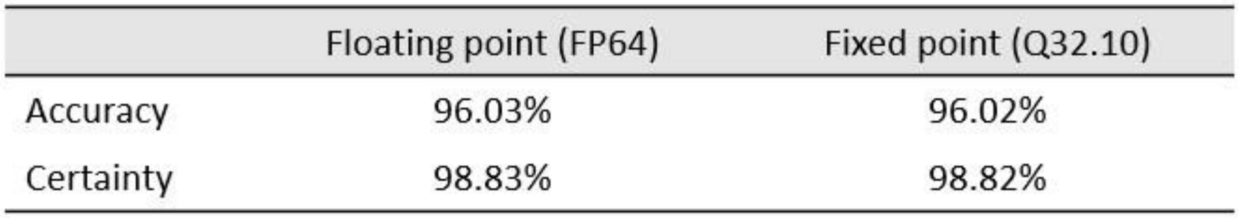
All the statistics were accumulated over 10,000 test images, and averages were calculated and evaluated.

4 Results

4.1 Time comparison

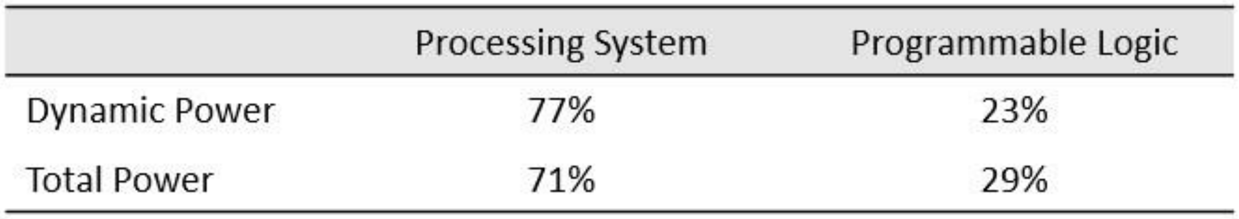


4.2 Precision comparison



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

4.3 Power comparison



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

5 Discussion

5.1 Conclusions

5.2 Future work

6 References