VLSI Design of Systolic Array Based IC for Accelerated Image Processing

Eliza P Cohn*, Rebecca J Greene*
*Johns Hopkins University

Abstract—**Matrix convolution is a computational task that is central to the popular fields of image processing and convolutional neural networks (CNNs). Because convolution involves repetitive operations on each input pixel, it is possible to design specialized hardware that will perform this tasks more efficiently than the classical Von Neumann or Harvard CPU Architectures. In this paper we propose a VLSI design based on a systolic array to accelerate 8-bit image processing and CNN computation. The design is written in SystemVerilog and tested on a 64x64 pixel image using Modelsim. Cadence Virtuoso and Encounter toolchains are used to synthesize and layout the design, from which we acquire approximations of power, timing, and area resources. Based on these estimates The design is capable of performing approximately 210 GFOPS/W (fixed point operations per second per watt), versus an estimated 6.25MFOPS/W for the comparable 8-bit PIC12F629 [4].**

I. Introduction

Image processing is becoming an increasingly fundamental task in a wide variety of disciplines. The need to enhance or extract information from an image plays a vital role in applications such as robotics, remote sensing, medical diagnosis, industrial inspection, and entertainment [5]. However, the standard von Neumann architecture is not well-suited to for image processing tasks. The architecture consists of a single, shared memory for both programs and data, a data bus, an arithmetic unit, and a control unit. The processor operates by alternating between fetching and execution cycles. Because of this, when trying to use von Neumann architecture for computationally intensive tasks, a bottle neck will occur when reading and writing data from the processing element to the memory.

Because of these limitations, an interest in more specialized architectures has emerged. Due in part to hardware sizes and costs decreasing while processing requirements become more understood, it is becoming more prevalent to design architectures for specific tasks. The graphics processing unit (GPU) is a hallmark innovation representing this school of thought. The GPU was developed to rapidly manipulate and alter memory for use in computer graphics and image processing. One novel example was developed by IBM in 2014 called the TrueNorth chip [7]. TrueNorth is a brain-inspired design, a neuromorphic CMOS integrated circuit with 4096 cores. Each processor core implements a convolutional neural network, containing 256 programmable simulated neurons. Each neuron then has their own set of programmable ”synapses”. In total, this allows for an extremely powerful and energy-efficient architecture that allows for very complex and specific programming. Another neuromorphic chip developed by Intel, called the Loihi chip, has a similar architecture that emulates the neural structure of the brain [8].

These designs serve as inspiration for addressing our specific problem of performing image processing through the use of convolutional kernels. To do this, we use a systolic architecture to preform accelerated matrix multiplication, which is useful for implementing CNNs. Systolic systems are an ideal choice in this task-specific design, for data flows from the computer memory through many processing elements (PEs) before returning back to memory. This permits multiple computations for each memory access. Because of this, systolic architectures can also speed the execution of compute-bound problems without increasing I/O requirements [2].

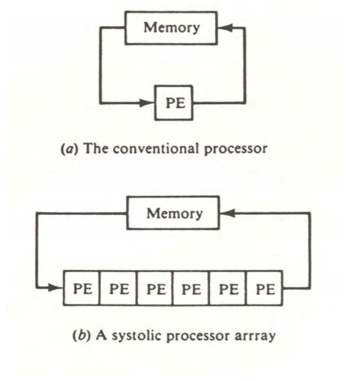

Memory
PE
(a) The conventional processor
Memory
PE
PE
PE
PE
PE
PE
(b) A systolic processor arrray

Fig. 1. A top-level comparison of Mano architecture versus systolic architecture [2]

Due to the comparative advantages of systolic systems, we seek to design an architecture that is able to apply convolutional filters to images. We base our design off of a proposal by Kiningham et. al from Stanford [3]. The basis of their design is to implement a systolic array based architecture that accelerates dense matrix multiplication operations. We preserve the fundamentals of their top level structure flow, starting first with a weight fetch, then pushing data through the array using an input queue, and catching the outputs of the array with an accumulator and passing that final output through a ReLU/Quantization step. However, the design of Kiningham et. al is centered on a 256x256 systolic array of multiply accumulate cells, which we propose to decrease to a 3x3 array to match the size of our convolutional kernel and decrease overall size. We also proposal a simplification of their ReLU/Quantization step, which will be discussed in more detail later in the paper.

II. Image Processing With Convolutional Kernels

Both simple image processing tasks and convolutional neural networks make use of kernels (also called masks or convolutional matrices) to transform input images. The kernel is a smaller matrix (usually 3x3 - 7x7 pixels) that is convolved with the input to produce an output image or intermediate network layer. Figure 2 shows an example of the emboss filter being applied to an input image.

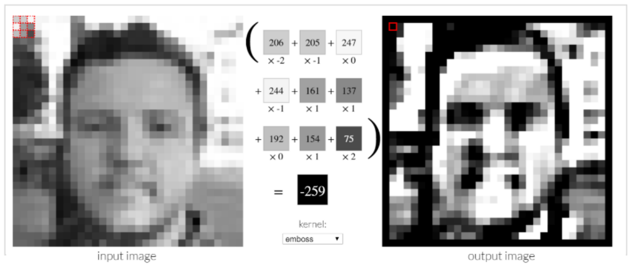


Fig. 2. The 3x3 emboss kernel is convolved with the 64x64 input image to produce a 62x62 output image [1]

In this example, the kernel is 3x3 pixel matrix with values

$$\begin{bmatrix} w_{00} & w_{10} & w_{20} \\ w_{11} & w_{11} & w_{21} \\ w_{22} & w_{12} & w_{22} \end{bmatrix} = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}$$

If we were to apply this to a simple 3x4 image with pixel values

$$\begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \\ c_0 & c_1 & c_2 & c_3 \end{bmatrix}$$

There would be two output pixels, with values

$$p_0 = \sum_{i=0}^{2} w_{i0}a_i + w_{i1}b_i + w_{i2}c_i$$

$$p_1 = \sum_{i=0}^{2} w_{i0}a_{i+1} + w_{i1}b_{i+1} + w_{i2}c_{i+1}$$

After multiplication and summation, the outputs are usually passed through an activation function that maps them to a limited set of allowable values. For CNN applications, this is typically implemented as a rectified linear unit (reLU) which aims to mimic the action potential of a cell through the transformation function

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

For a simple image processing task, the output values must be constrained to the allowable range for the image storage type. If the input image is represented as an 8-bit grayscale image then the output pixel values must be restricted to the set [0,255]. This problem can be solved with either normalization or saturation thresholding. The later technique simply imposes upper and lower bounds by setting

$$f(x) = min(max(x, 0), 255)$$

III. Convolution with Systolic Array

Any architecture consisting of multiple processing elements could nominally be labeled a systolic array. The PEs can be aligned across any number of dimensions, and information is passed between different elements. However, systems optimized for solving convolution problems usually take one of two forms [2], both shown in Figure 3.

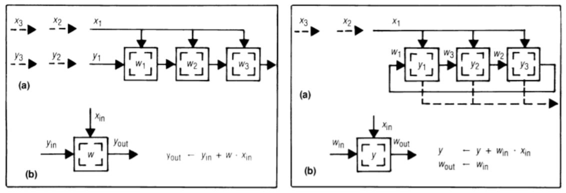


Fig. 3. The left image shows a systolic array where weights are fixed in local memory, x values are broadcast, and y values move systolically through the array. The right image show an array where the y values are stored, and weights are pushed through the array [2]

For the purpose of accelerating a convolutional kernel, latency is minimized by using an architecture that hold stores the kernel weights in local memory at each processing element, while the image pixels are passed systolically between the PEs. This way the array size is determined by the dimensions of the kernel, which will are typically 1+ orders of magnitude smaller than the input image dimensions. The array is heavily pipelined, so the data throughput will always be one pixel per clock cycle, independent of which value is fixed. However the latency is proportional to the perimeter of the array.

This leads to the array architecture shown in Figure 4, based off the Kiningham et. al proposal [3]. For each clock cycle, each processing element has two inputs – an image pixel (d_i), and the sum from all elements above it (Y_i). At the start of the next clock cycle, the PE will pass the input image pixel to the next element on the right (d_o), and will pass downwards the output Y_o where

$$Y_o = Y_i + wd_i$$

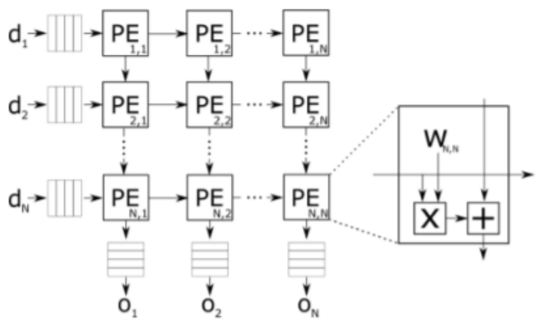


Fig. 4. Proposed 2-D Systolic Array [3]

After passing through the array, the output of all columns need to be delayed and summed together in an accumulation step to produce a value for the output pixel. This value is then quantized with an activation function, yielding the final pixel result.

IV. Accelerator Design

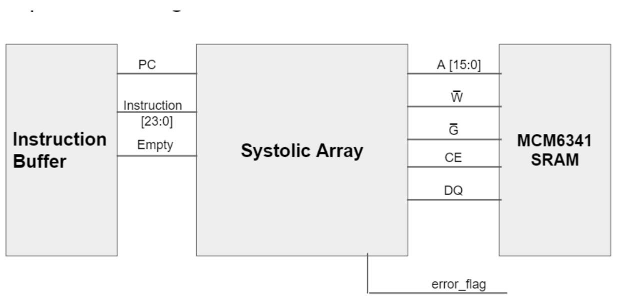


Fig. 5. The Proposed Image Processor is shown interacting with an external CPU via a dual-port ram and stores output in SRAM

The proposed chip was designed to have a functional model similar to a that of a digital signal processor (DSP) integrated circuit – it has been optimized for the matrix convolution task but is not turing complete. Thus it is necessary to connect the accelerator to some outside CPU to specify the incoming data. For optimal efficiency, these instructions should queued via a dual-port ram, and then the accelerator will store output layers in a SRAM, as shown in Figure 5.

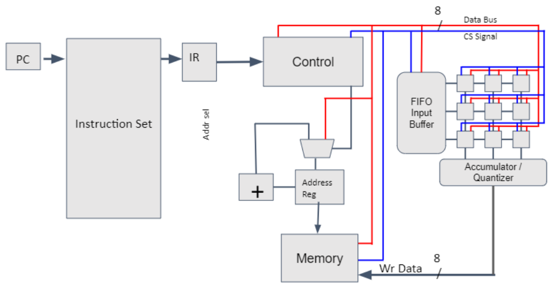
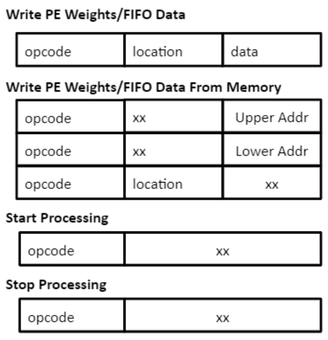


Fig. 6. Block Diagram of Proposed Marix Convolution Accelerator

Figure topblock shows a block diagram for the proposed design. The PE array and the accumulator/quantizer are the two fundamental components that implement the algorithms discussed in sections II and III, while the remaining units provide scaffolding to allow for communication and data flow. To acquire input, the program counter (PC) register increments at each clock cycle to store a new instruction from the dpram into the instruction register (IR). The instruction is split into three parts (see Figure 7), and the opcode becomes a key for a lookup table that controls the data bus and the chip select lines. Different instructions are used to load weights into the processing elements, load input image pixels into the array buffers (either from memory or the instruction set), start and stop the array from processing, and store the output values in memory. A small state machine tracks the order of opcodes to ensure that multi-step instructions are completed correctly, and throws an error flag if they are not, or if the array input buffers overflow.


Write PE Weights/FIFO Data

opcode	location	data

Write PE Weights/FIFO Data From Memory

opcode	xx	Upper Addr
opcode	xx	Lower Addr
opcode	location	xx

Start Processing

opcode	xx

Stop Processing

opcode	xx

Fig. 7. Sample Instructions

The data bus is designed to accommodate 8 bit pixel data, which constitutes the vast majority of the information being transmitted through the bus. One downside to this architecture is that two clock cycles are required to deliver a full 16bit memory address. Since the values being stored are all sequential image pixels, we structured the output address register to allow for automatic incrementation if specified. Then after specifying a starting address, output values will be stored at sequential address every clock cycle. This is especially valuable for large images, which need a very streamlined write process to achieve the highest possible throughput and latency.

A. The Processing Element

The PE block diagram is shown in Figure 8. Each processing element needs to perform an add multiply at each clock cycle. Since the nature of the design requires large space and power budgets, we selected multiplier and adder architectures that balanced the speed/area/power tradeoffs. With this goal in mind, the multiplier implemented was a 9-bit signed Baugh-Wooley multiplier, and the adder implemented was an 18-bit carry look ahead adder. When the PE array is processing, on each clock cycle an individual element witll multiply input data, ImDin, with a weight that is stored in local memory. Padding is done on the inputs before multiplication to ensure no overflow occurs. Once this multiplication is complete, the result is added together with the output of the previous vertical processing element, Yi. This final result, Yo, is a maximum of 18 bits and is fed into the PE below. ImDin is delayed one clock cycle and passed to the next PE on the right. Thus the data moves systolically across and down the array.

Before the array can start processing, it is necessary to specify the kernel that is being used for convolution. To do this, the weight for a given processing element is loaded in using the data bus and the chip select line for that PE. These are both specified by the control logic LUT and a demultiplexer to ensure that the correct weight is being read into each processing element.

B. The Quantizer/Accumulator

The purpose of the accumulator was to implement the final summation to generate the output pixels, as discussed in sections II and III. To ensure that the correct summands

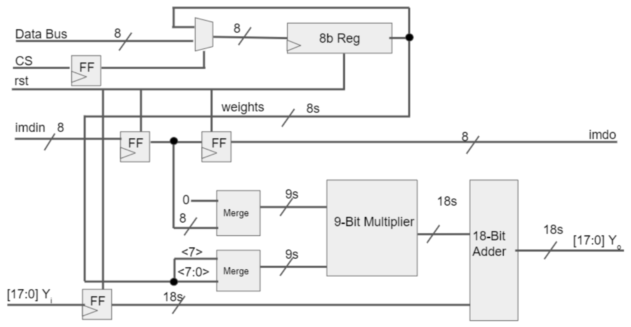


Fig. 8. Block Diagram of Proposed Processing Element

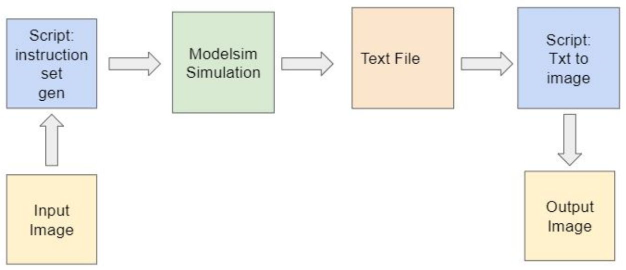


Fig. 10. Block Diagram of the implemented testing flow

were used, a series of delays were implemented to buffer the incoming data using registers. The inputs to the accumulator are Y0, Y1, and Y2, which represent the output of the three columns from the processing array. The inputs to the 20-bit carry look ahead adder was padded to 20-bits to ensure no overflow occurred. The logic is described in the following block diagram.

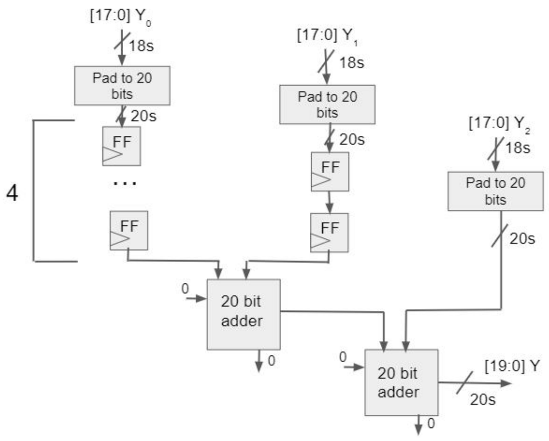


Fig. 9. Block Diagram of Proposed Accumulator

The 20-bit output of the accumulator was fed directly into the quantizer, which implemented the saturation thresholding technique as described in the introduction. The output of the quantizer is the final value representing the resulting pixel, and was written to memory.

V. Test Setup

To test our design we used the accelerator to transform a 64x64 grayscale image using a 3x3 emboss kernel. The test setup involved multiple test scripts, as shown in Figure 10. We used a matlab script to emulate the governing CPU, whose task is to read the image from some initial memory location, and fill the IC input dpram with instructions to process it correctly. The instruction set produced by our script first wrote kernel weights to the processing elements, then loaded data into the array input buffers. Once the data was loaded, it started the array, and periodically started and stopped automatic writes to output memory as valid data came out of the array, using NOP instructions to create the correct timing. Once all the necessary data had passed though, the stop processing instruction was given. It should be noted that it is necessary to pause memory writes between output rows for a length of time equal to the perimeter of the array. This script generated a txt file that was then loaded into the dpram used in the top-level SystemVerilog simulation file. For more efficient processing, instructions to load input image data should be inserted between the write start/stop commands so that the array can run continuously.

This dpram was used to run a ModelSim simulation of our design. The top level SystemVerilog script included a module designed to emulate an output memory, but which wrote the output pixel values to a txt file instead of storing them. A final script was use to convert this text file back into an image format.

Figure 11 displays two images. The image on the left is the original image after resizing and gray-scaling. The second image is the result of our architecture applying the emboss filter and our reconstruction script processing the output as a gray-scaled image.

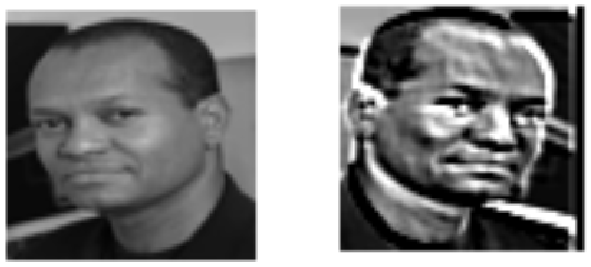


Fig. 11. Image Processing Example: Applying the emboss filter

Code for the verilog model and test scripts is available at https://bitbucket.org/regreener/520419fp/branch/modelsim

VI. Results and Figure of Merit

Using the Cadence Toolchain, we took the top-level verilog code that modeled our architecture to run both synthesis and place and route on our design. Through this, we were able to simulate our architecture in order to compute a realistic measure of the power, area, and timing.

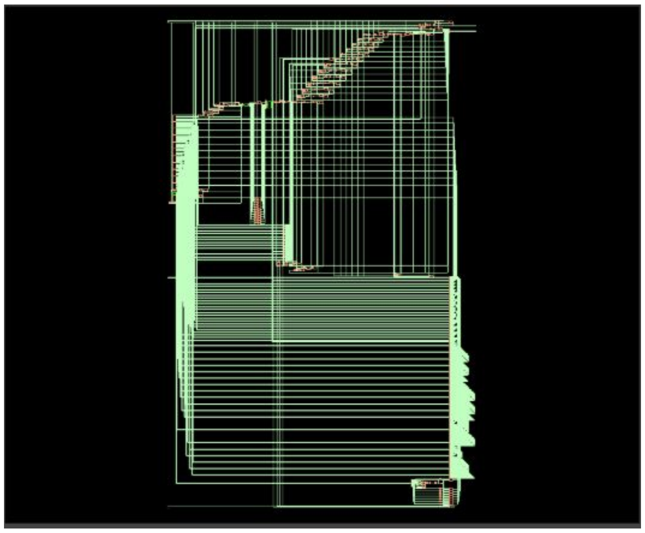


Fig. 12. Synthesis of the Systolic Architecture

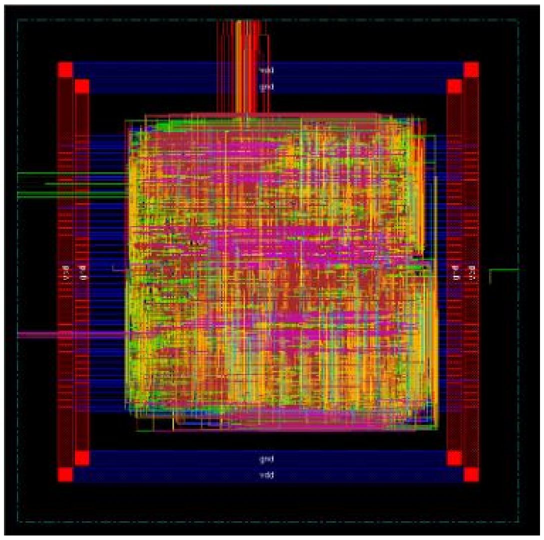


Fig. 13. Place and Route of the Systolic Architecture

Assuming an optimal instruction set, the design can perform approximately 20 operations per clock cycle while processing an image. Based on the Cadence Encounter timing reports, we run the system with a 10ns clock, or 20 GFOPS. Combining with the power report this leaves us with a figure of merit of 210 GFOPS/W.

For comparison, an 8-bit PIC12 designed with a Harvard CPU Architecture can be clocked to run 5MIPS. If we generously assume that each of those instructions is running an ALU operation, the PIC12 could run a maximum of 5MFOPs. With a power consumption of 800mW, the PIC12 can only perform 6.25 MFOPS/W, although it does have the advantage of being a standalone, turing complete CPU. [4]

In table 1, we report the specifics of the timing report calculated post optimization of the place and route.

In table 2, we report the other relevant figures of merit that describe the power, speed, area, and density of design.

Setup mode	all	reg2reg	in2reg	reg2out
$WNS(ns)$	1.213	1.415	1.213	6.426

Hold mode	all	reg2reg	in2reg	reg2out
$WNS(ns)$	0.022	0.022	0.096	1.295

TABLE I
ARCHITECTURE TIMING

$Area$	42,901 µm
$Power$	9.5 mW
$Speed$	100 MHz
$ModuleDensity$	93.5%

TABLE II
FIGURES OF MERIT

VII. Engineering Ethics

The IEEE Code of Ethics requires members to make value public and environmental safety, avoid conflicts of interest, make honest and realistic claims, reject bribery, improve understanding, maintain technical competence, seek critique, treat others fairly, avoid injuring others, and assist colleagues in their development [6]. Not all of these codes are directly applicable to this project, for as this is a course assignment it is difficult to create conflicts of interest or scenarios for public endangerment. The authors guarantee that no professors or teaching assistants were bribed for our results or final grade, with the exception of one coffee offered to Khaled in exchange for advice on how to resolve timing issues. Through this project both authors have dramatically burgeoned their understanding of the VLSI design process, and increased their technical competence, although they have done little to assist the general public. Through regular progress updates we have sought the critique of the teaching staff, and done our best to implement design changes as advised. When possible we have tried to assist other classmates through their journeys with cadence, and in turn received plenty of help from others. Some wrist tendons may have been injured over the course of the project, but this is a acceptable risk of computer aided design.

VIII. Global Impacts of IC Design

Concerns about power consumption made up a significant portion of the 2015 ITRS. The desire fore computing power in individual devices has increased in the past decade with the advent of the 'Internet of Everything', and this is tied to an increased energy demand. One of the main goals for this project has been to create a piece of specialized hardware that can rapidly and efficiently perform the commonly used matrix convolution operation, which would theoretically simultaneously increase compute performance and decrease energy demands for these devices running neural networks.

The collection of electronics nearing the end of their useful life that are thrown away, recycled, or donated is called "E-waste". The EPA and governments worldwide see a potential in this "E-Waste" to be reused, refurbished or recycled into useful materials. This would limit the amount of waste that would end up in landfills or improperly polluted from old electronic materials. This type of waste is the one of the fastest growing waste stream in the world and growing year by year. A surplus of waste is a problem not only for the generating country, but for countries all over the world. Though processing plants and an established recycling system are established can help handle and minimize the waste, it is still important to keep this issue in mind. The issue is magnified for countries who lack the necessary resources to successfully handle this waste. In the US, the environmentalist community along with the electronics community are pushing a National Strategy for Electronics Stewardship within the government that focuses on increasing greener design policies for electronics, allowing the US to be an example for the world by lowering "E-waste" internally and in our exports. This is just one example of an incentive for greener electronics policy, and other nations are creating similar plans of action to help make the electronic community more environmentally aware and reduce their footprint on the environment.

IX. Acknowledgement

We would like to thank Dr Ralph Etienne-Cummings for his instruction in the CAD for VLSI course and for providing the inspiration for this project, along with Jonah Sengupta and Khaled Abourmerhi for their role as teaching assistants and their support with the Cadence Toolchain. Funding has been provided by the Department of Electrical and Computer Engineering at Johns Hopkins University.

References

[1] V. Powell, "Image Kernels Explained Visually," Setosa . [Online]. Available: http://setosa.io/. [Accessed: 16-Dec-2019]..

[2] H. T. Kung, "Why systolic architectures?" IEEE computer 15.1 (1982): 37-46.

[3] K. Kiningham, M. Graczyk, A. Ramkumar, "Design and Analysis of a Hardware CNN Accelerator" 2017

[4] Microchip. "8-Pin FLASH-based 8-but CMOS Microcontrollers" PIC12F629/675 Data Sheet. 2003

[5] R. Wang, "Image Processing and Related Fields" HMC . [Online]. Available: http://fourier.eng.hmc.edu/e161/ [Accessed: 18-Dec-2019]

[6] Institute of Electrical and Electronics Engineers, Inc.. (2006). Code of Ethics IEEE, http://www.ieee.org/. Retrieved at July 28, 2009,

[7] P. Merolla, J. Arther. "A million spiking-neuron integrated circuit with a scalable communication network and interface", Science. 345 (6197): 668.

[8] Loihi. "Neuromorphic Computing." Intel. [Online]. Available: https://www.intel.com/content/www/us/en/research/neuromorphic-computing.html [Accessed: 18-Dec-2019]