A HARDWARE ARCHITECTURE FOR ACCELERATING NEUROMORPHIC VISION ALGORITHMS

A. Al Maashri, M. DeBole, C.-L. Yu†, V. Narayanan, C. Chakrabarti†

Department of Computer Science and Engineering,
The Pennsylvania State University
University Park, PA 16802, USA
{maashri, debole, vijay}@cse.psu.edu

†School of Electrical, Computer and Energy Engg,
Arizona State University
Tempe, AZ 85287, USA
{chi-li.yu, chaitali}@asu.edu

ABSTRACT

Neuromorphic vision algorithms are biologically inspired algorithms that follow the processing that takes place in the visual cortex. These algorithms have proved to match classical computer vision algorithms in classification performance and even outperformed them in some instances. However, neuromorphic algorithms suffer from high complexity leading to poor execution times when running on general purpose processors, making them less attractive for real-time applications. FPGAs, on the other hand, have become true signal processing platforms due to their lightweight, low power consumption and massive parallel computational resources. This paper describes an FPGA-based hardware architecture that accelerates an object classification cortical model, HMAX. Compared to a CPU implementation, this hardware accelerator offers 23X (89X) speedup when mapped to a single-FPGA (multi-FPGA) platform, while maintaining a classification accuracy of 92.5%.

Index Terms: Neuromorphic vision algorithms, FPGA, Signal Processing, Hardware, Neuromorphic Hardware Architecture

1 INTRODUCTION

In the last three decades, neuroscientists have made a number of breakthroughs in understanding the ventral and dorsal pathways of the mammalian visual cortex. These advances have inspired a number of computer vision algorithms – collectively referred to as "neuromorphic vision algorithms". Neuromorphic algorithms are derived by reverse-engineering the mammalian brain to produce vision algorithms that are both efficient and robust. Additionally, neuromorphic vision algorithms are promising alternatives to classical computer vision approaches [1], where extracting objects of interest and classifying them are based on cortical models describing the brain [2,3]. The Riesenhuber & Poggio model, also referred to as HMAX [3], is an example of a biologically-inspired algorithm used for object recognition. Despite its highly accurate performance and robustness, HMAX exhibits a relatively slow run time when implemented on standard general purpose processors. This makes a real-time realization of the algorithm a challenging task, especially when targeting the algorithm for small-footprint embedded systems with limited computational resources and power constraints.

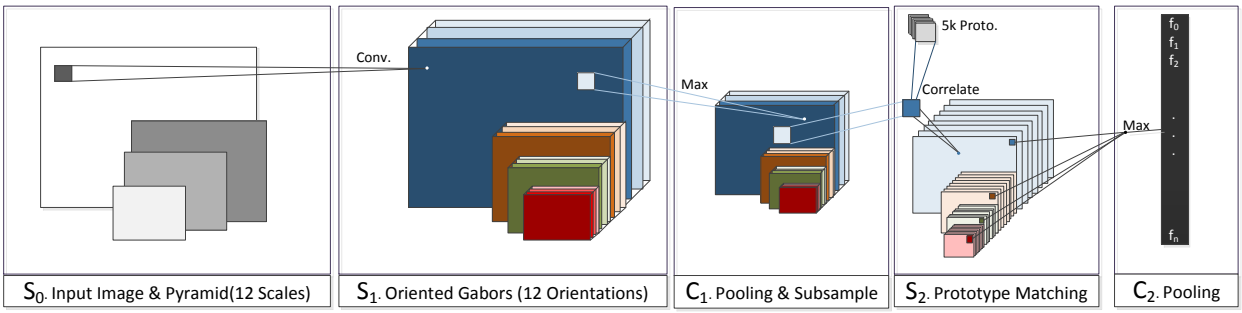
Today, Field-Programmable Gate Arrays' (FPGAs) are serious contenders to general purpose processors and are being used in a range of application domains including robotics, medicine, image processing, and video analytics. Recently announced FPGAs, now at the 28nm node, contain over 1,955,000 logic cells [4], providing FPGAs with unprecedented computational power while maintaining a minimal power footprint. With the increases in available FPGA resources, it is unsurprising that FPGAs are gaining popularity for accelerating neuromorphic vision algorithms [5,6].

This paper describes a hardware architecture for accelerating the HMAX model by exploiting the massively parallel resources on FPGA devices. Compared to a CPU implementation of the HMAX model, these accelerators achieve 23X speedup when mapped to a single FPGA device and 89X speedup when mapped to a multi-FPGA platform. The accuracy of the FPGA implementation is maintained at 92.5%. Additionally, for applications where the accuracy can be traded off for speedup, this paper also describes a faster implementation with 64.5% classification accuracy.

The remainder of the paper is organized as follows. Section 2 describes HMAX, a cortical model for object classification. Section 3 discusses the architecture of the hardware accelerator and provides a high-level assessment of the accelerator. Section 4 discusses a number of proposed performance-boosting optimizations and an alternative design. Section 5 presents a discussion of the results. Finally, section 6 concludes the paper.

2 HMAX MODEL

HMAX ("Hierarchical Model and X") is a model of the ventral visual pathway from the visual cortex to the inferotemporal cortex, IT. This model attempts to provide space and scale invariant object recognition by building complex features from a set of simple features in a



Time (%)	.02	3.01	.16	**96.61**	.2

Figure 1. HMAX stages and percentage of execution time while running on CPU

hierarchical fashion. The reader is encouraged to consult [3,7] for an in-depth treatment of the topic. While several versions of HMAX exist, the version used for this work was an extension developed by Mutch & Lowe [1].

As illustrated in Figure 1, the HMAX model consists of a preprocessing stage S_0 and 4 cortical stages referred to as S_1, C_1, S_2 and C_2. The S_0 stage provides scale invariance by down-sampling the image to generate an initial image pyramid that is fed to the S_1 stage. Gabor filters are used to implement the S_1 stage since they have shown to "provide a good model of cortical simple cell receptive fields" [7]. The C_1 stage pools over the output of the S_1 stage and finds the local maximum value across adjacent scales over the same orientations. The output of the C_1 stage is pooled over by the S_2 stage, which performs correlation operation on the C_1 output against a stored dictionary of fuzzy prototypes gathered from natural images. The last stage in the HMAX model is C_2, which computes the global maxima over all scales and positions – producing a feature vector that is fed to a classifier to perform final object classification.

A CPU reference of HMAX was implemented on an Intel S7000FC4UR server containing a quad-core 3.2 GHz Xeon processor and 24GB of system memory. The reference implementation was originally developed in C++. The percentage of execution time for each stage is shown in Figure 1. S_2 is the most computationally complex layer as it attempts to match a set of 4x4x*m*, 8x8x*m*, 12x12x*m*, and 16x16x*m* prototypes, which have been randomly sampled from a set of natural images during training stage. The value of *m* represents the number of orientations extracted from the original image during the S_1 stage and typical values for *m* are between 4 and 12 (CPU reference uses 12). These prototypes make up a patch dictionary consisting of 5000 entries which are used as fuzzy templates consisting of simple features that are position and scale invariant. S_2 then computes the response of a patch, X, of C_1 units, to a particular S_2 feature prototype, P, of size *n x n x m* (*n* = 4, 8, 12, 16). This response is given by the normalized dot product:

$$R(X,P) = \frac{X \cdot P}{\|X\|^2 - (\sum x_i)^2}$$

Figure 2 shows the pseudo code for computing the S_2 response for a given input image. The outer loop is a result of the *m* image pyramids that are generated prior to S_1 and then pooled (across scales) during C_1. The inner loops are due to the 5000 prototype patches which consist of 12 orientations each.

The bottleneck of HMAX is the S_2 stage. The following section discusses a hardware architecture for accelerating the S_2 stage. Note that although the discussion is limited to S_2 accelerator only, all other stages were implemented and mapped to FPGA. However, due to space limitations we focus on S_2 accelerator since it contributed the most to the overall HMAX speedup.

S_2 Stage(Pseudo Code)

```
1   for each scale, s = 1:1:11
2     for each orientation, o = 1:1:m
3       for each window, w = {4x4, 8x8, 12x12, 16x16}
4         sum_norm(w)+= corr(S1[s,o], ones[w])/|w|
5         sum2_norm(w)+= corr(S1[s,o]^2, ones[w]/|w|)
6       end;
7     end;
8     for each prototype, p = 1:1:5000
9       for each orientation, o = 1:1:m
10        result_correlate+= corr(S1[s,o], proto[p,o])
11      end;
12      result_S2[s,p] = result_correlate./ (sum_norm − sum2_norm)
13    end;
14  end;
```

Figure 2. Pseudo Code of S_2 Stage

3 S_2 HARDWARE ACCELERATOR

The S_2 stage performs template-matching operation through correlation filtering. A stream-based correlation is proposed, in which the input image is streamed into a convolution network allowing the correlation filter to perform the MAC operations in parallel. This consists of a multi-tap, kernel-size-configurable correlation filter depicted in Figure 3, in order to support all four kernel-sizes; 4x4, 8x8, 12x12 and 16x16. The filter is composed of sixteen 1D filters as shown in Figure 3 (top). The user can configure the filter with the desired kernel size at runtime and choose the output from the corresponding tap (See bottom-left of Figure 3). The outputs of all 1D filters are accumulated using adder-tree architecture as illustrated in Figure 3 (bottom-right).

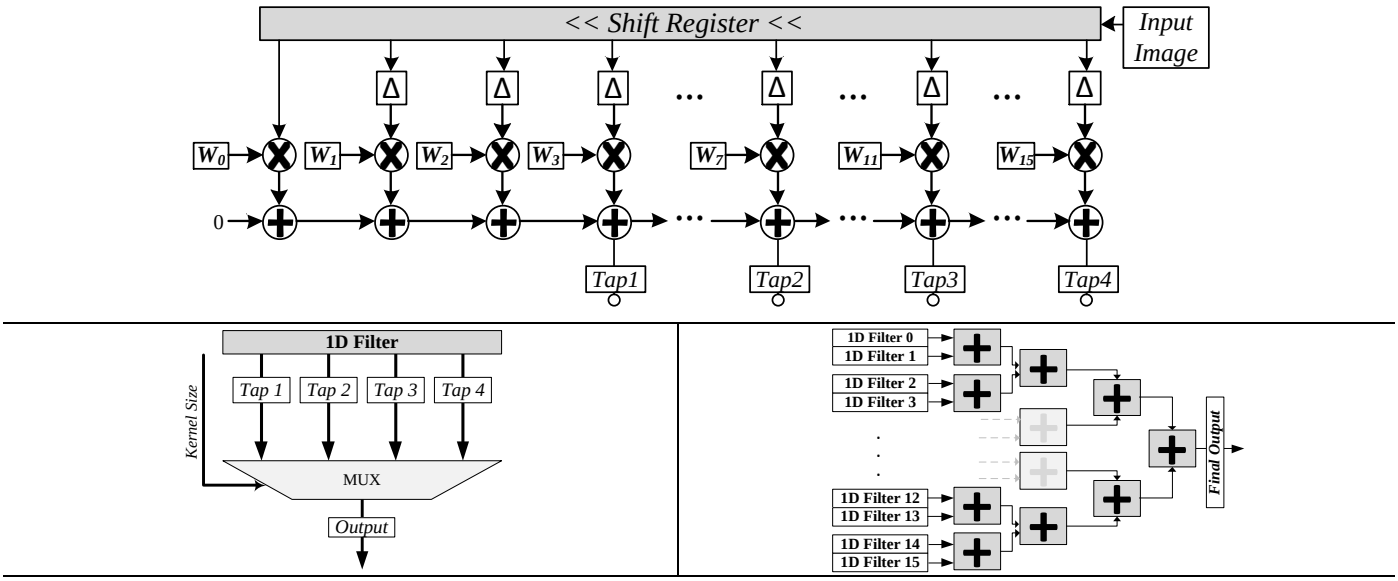


Figure 3. Architecture of kernel-configurable correlation filter: (*top*) Pipeline architecture of 1D filter. (*Left*) Depending on the kernel size, only one of the correlation outputs (taps) is selected using a multiplexer, (*Right*) The output from all 1D filters is summed using an adder tree

The HMAX accelerators are ported to a packet-switched routing infrastructure developed to allow the accelerators to communicate and share data with one another. Preliminary results show that the accelerated HMAX achieves a speedup of 3.91X compared to a CPU implementation of the HMAX model. Based on these results, a high-level analysis of the S_2 accelerator is performed to identify the performance bottlenecks. Consequently, a number of optimizations are proposed to improve the performance.

3.1 An Assessment of the S_2 Accelerator

This subsection evaluates the S_2 accelerator architecture described above. An overview of the operations in the S_2 accelerator is described below:

- First, configurations such as kernel size and image size are transferred from the user to the S_2 accelerator on the FPGA. The time to transfer the configurations is denoted by T_{config}.
- Then, the prototype patch (i.e. correlation coefficients) is transferred to the S_2 accelerator. The time to transfer the patch is denoted by T_{coeff}.
- After that, the input image is transferred to the S_2 accelerator. The time to write the image to the accelerator is denoted by T_{image}.
- As the image is being streamed into the accelerator, the S_2 unit starts to perform the correlation operation. The time to compute the correlation is denoted by $T_{compute}$. Note that T_{image_delta} represents the time between the image being streamed into S_2 accelerator and the time S_2 starts computation. T_{image_delta} is important because it is the fraction of T_{image} that is not masked by $T_{compute}$.
- Finally, the results are transferred back to the user. The time to transfer the results is denoted by $T_{results}$.

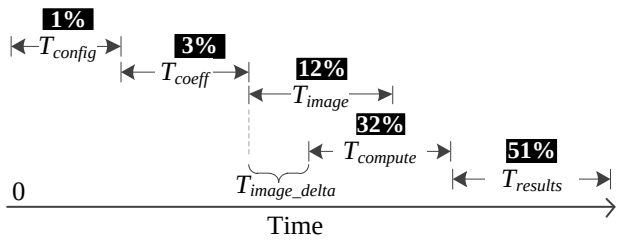


Figure 4. Operations of S_2 accelerator: Boxes on top of time slots indicate the execution time when processing the largest scale

Figure 4 illustrates the process described above. The figure shows the percentage of execution time for each stage when processing the largest scale.

It is evident that the three major contributors to S_2 total execution time are T_{image}, $T_{compute}$ and $T_{results}$. On the other hand, T_{config} can be ignored since its execution time is insignificant and it occurs infrequently (i.e. only when kernel size or image size changes). Finally, it is observed that T_{coeff} execution time is relatively insignificant; however, this transaction occurs frequently (~5000 times per image scale for each orientation). Based on these observations, the following section proposes a number of hardware optimizations that target reducing these critical paths.

4 S_2 ACCELERATOR HARDWARE OPTIMIZATIONS

Based on the assessment presented in the previous section, the following is a discussion of a number of proposed optimizations that reduce the overall execution time of the S_2 accelerator.

4.1 Optimization #1: Reducing $T_{results}$

In the initial design of the S_2 accelerator, the output is stored in a First-In-First-Out (FIFO) queue. When the results are

read from the accelerator, they are streamed out one pixel at a time (i.e. the rate at which the S_2 is capable of producing results). However, the underlying routing infrastructure can support 4-pixel transaction per clock cycle. This means that the output interface of the S_2 accelerator has degraded the transfer rate by 75%.

To resolve this issue, an "aspect-ratio" FIFO which packs every 4 pixels into a single data chunk, is added at the output interface of the S_2 accelerator. This optimization allows the S_2 output rate to match that of the underlying routing infrastructure.

4.2 Optimization #2: Reducing occurrence of $T_{results}$

In the HMAX model, S_2 performs correlation filtering across all 12 orientations for each scale. In addition, the model accumulates the correlation results for all 12 orientations.

The initial S_2 architecture doesn't perform this accumulation internally. Therefore, every orientation streaming transaction is associated with a request to read the results. This can be alleviated by modifying the S_2 accelerator such that accumulation of the results is done internally, hence reducing $T_{results}$ to only 1 per 12 orientations. This is implemented in hardware by adding a temporary storage, referred to as *Accumulation Memory*, to store the output of each correlation. Then, as S_2 produces output of the next orientation, hardware logic reads the corresponding accumulated output of the previous correlation from memory, updates it with the current output and writes the result back to the *Accumulation Memory*. The logic was designed such that accumulation is done on the fly in order to sustain a throughput of one accumulation per cycle. At the end of the 12th orientation, the accumulated result is transferred to the user. Figure 5 illustrates the proposed architectural modifications including optimizations #1 & #2.

4.3 Optimization #3: Reducing occurence of T_{image}

Each time a correlation result needs to be computed, an image orientation must be streamed into the S_2 accelerator. Consequently, the same orientation must be streamed repeatedly each time a new prototype patch is loaded to the accelerator, leading to a total of 542172 image writes (Image orientations can be correlated with prototypes patches that are less or equal size). This enormous number of writes has a significant impact on the performance. Therefore, to avoid this unnecessary streaming of image orientations, a temporary storage, referred to as *Image Memory*, is introduced at the input interface of the S_2 accelerator. This temporary storage can accommodate all 12 orientations of the same scale. Additional control logic is added to manage streaming the orientations internally. This architectural modification reduced number of image writes to only 132 per image.

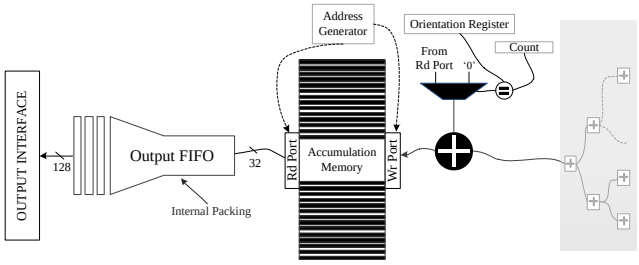


Figure 5. Adding aspect-ratio FIFO and accumulation memory

4.4 Optimization #4: Resource Duplication

By taking advantage of the available resources on the Multi-FPGA system hosted by the development platform, a total of 4 S_2 accelerators were mapped to a multi-FPGA system. This duplication of the accelerators allowed the correlation operations to be held in parallel.

4.5 An Alternative S_2 Design: Network-Configurable S_2 Architecture

This section describes an alternative architectural design of the S_2 accelerator. As described above, the S_2 accelerator needs to support four kernel sizes: 4x4, 8x8, 12x12 and 16x16. A configurable correlation filter (convolver) is proposed to implement these kernels that maximize the hardware utilization for three of the four kernel sizes.

The basic building block is a 4x4 convolver built with four 4-tap pipelined FIR filters as shown in Figure 6. The inputs to the FIR filters are fed from a serial-to-parallel FIFO; the outputs of the four filters are accumulated using an adder tree. The larger filters (8x8, 12x12 and 16x16) are built using the 4x4 filter.

Figure 7 shows a configurable filter capable of supporting kernel sizes of 4x4 and 8x8. When only 4x4 kernel size is required, the outputs of 4x4 filters A, B, C and D are directly sent to the adder tree. When 8x8 kernel size is required, the outputs of convolvers A and B are fed to convolvers C and D. The inputs of filters A and B are fed by the configurable FIFO array built with two cascaded 1-to-4 FIFOs. For the 8x8 correlation filter, the adder tree sums up the eight outputs from convolvers C and D and produces the final result. The multiplexers route the data according to the kernel size chosen by the user. Similarly, four 8x8 filters can be used to form a 16x16 filter, as depicted in Figure 8, and it can be configured as sixteen 4x4 or four 8x8 correlation filters. The 16x16 filter also supports 12x12 correlation operation, where only nine 4x4 filters are used while the other seven are disabled.

The hardware utilization of the S_2 accelerator is 100% for most of the kernel sizes. There is no idle 4x4 filter during computation, except for 12x12 2D correlation filter. Since it is a pipelined design, the S_2 unit's throughput is 1 pixel/cycle for 12x12/16x16 kernel, 4 pixels/cycle for 8x8 kernel, or 16 pixels/cycle for 4x4 kernel.

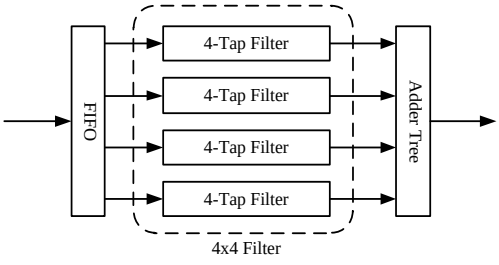


Figure 6. Building block: 4x4 correlation filter

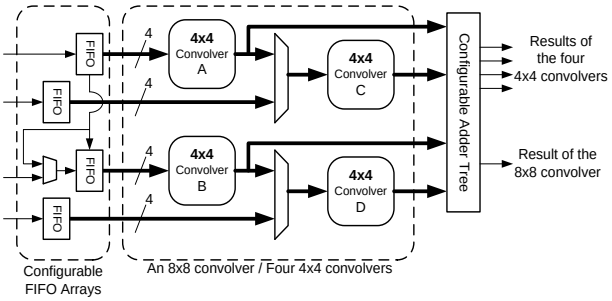


Figure 7. Configurable 4x4/8x8 correlation filter network

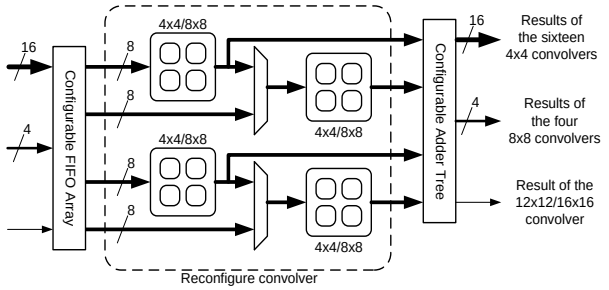


Figure 8. Overview of the Network-Configurable S_2 accelerator

5 RESULTS

5.1 Experiment Setup

An FPGA development system from Nallatech [8] is used to conduct the experiments. The system houses an Intel S7000FC4UR motherboard with a quad-core Xeon processor running at 3.2 GHz, with a total of 24 GB system memory. The motherboard uses Front Side Bus (FSB) to interface to a multi-FPGA acceleration module. FSB is a 64-bit bus running at a frequency of 1066 MHz allowing data transfer rates of up to 5.8 GB/s sustained read (i.e. System Memory to FPGA), and 2.8 GB/s sustained write (i.e. FPGA to System Memory). The multi-FPGA acceleration module contains four Virtex-5 SX-240T FPGAs [9]. All FPGAs operate at 80 MHz.

The accelerators were tested using Caltech 101 data set [10], using 200 images selected from 4 different categories: cannon, car side, pyramid and ketch. HMAX produces a feature vector that is used as an input to a regularized least-square classifier, which is trained with 15 images from each category.

5.2 Resource Utilization vs. Accuracy

The bit width of the input image determines the consumed resources on the FPGA. At the same time, increasing the bit width impacts the accuracy of the output results. Table 1 shows a tradeoff between consumed resources and accuracy of the results for different input bit width implementation of the S_2 accelerator. This analysis is useful when mapping the S_2 accelerator to different platforms. For instance, if S_2 accelerator is to be mapped to Xilinx Virtex 5 FX-130T device [9], which only has 320 DSP48 slices, then the table suggests that a maximum of 21-bit input image should be used. However, more freedom is given when mapping the accelerator to Xilinx Virtex 5 SX-240T device [9], which has 1056 DSP48 slices. Since 21-bit input gave sufficient accuracy, it was used throughout the experiments.

Table 1. Resource vs. Accuracy tradeoff. The accuracy of computed results are compared to a floating-point implementation running on CPU

Input Bit Width	Consumed Resources				Average Discrepancy
	Reg. Slices	LUT Slices	BRAM	DSP	
21	46944	42172	24	259	2.53E-04
25	56640	43772	24	515	9.98E-06
32	69504	47404	24	1027	8.11E-06

5.3 Accelerated HMAX Classification Accuracy

A total of 200 images were used for testing three different platforms. The first platform represents a CPU implementation of the HMAX model [1]. The second platform is an accelerated HMAX running with all scales, all orientations and a total of 5000 S_2 prototype patches. The third platform is an accelerated "reduced" HMAX. The reduced HMAX uses only 7 scales, 12 orientations and 3439 prototype patches. Also, the reduced HMAX supports kernel sizes of 4x4 and 8x8 only. Both second and third platforms are mapped to a Xilinx Virtex SX-240T FPGA device on the Nallatech development system.

Table 2 shows the classification accuracy of the three platforms described above, along with the execution time per image. The first and second platforms scored a classification accuracy of 92.5%. This indicates that accelerated HMAX running on FPGA maintained the accuracy of CPU implementation. On the other hand, the "reduced" version of the accelerated HMAX exhibits degradation in accuracy down to 64.5%. However, the execution time of the third platform was reduced to 2 seconds. Therefore, the reduced version of HMAX can be used for applications where more speedup is required on the expense of a reasonable accuracy loss. Moreover, this "reduced" version of accelerated HMAX can be mapped to small-footprint, low power FPGA devices such the Virtex-5 FX-70T [9] allowing the realization of HMAX on actual embedded systems. Also, it is observed that increasing the input bit width didn't improve the accuracy of the "reduced" HMAX.

5.4 HMAX Accelerator Speedup Performance

Figure 9 shows the speedup in execution time gained by each implementation of S_2 accelerator. All speedups shown in the

Table 2. Object classification accuracy for the three platforms.

Platform	# scales	# orient.	# prototypes	Accuracy (%)	Exec. time/ image (sec)
CPU	12	12	5000	92.5	222
FPGA	12	12	5000	92.5	11
Reduced FPGA	7	12	3439	64.5	2

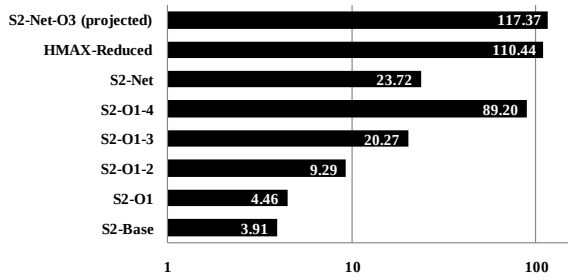


Figure 9. Speedup comparison between different S_2 implementations, normalized to CPU implementation of HMAX

figure are normalized to the CPU implementation of the HMAX model. The initial design of the S_2 acceleration, denoted by S2-Base, gave a speed up of 3.91X. S2-O1 represents the implementation of optimization #1, which yielded 4.46X speedup. S2-O1-2 which represents the combined implementation of optimizations #1 and #2, gave 9.29X speedup. Similarly, S2-O1-3 which represents the combined implementation of optimizations #1, #2 and #3, delivered 20.27X speedup. Moreover, S2-O1-4, which implements all four optimizations, yielded 89.20X speedup.

On the other hand, the alternative design of the S_2 accelerator, denoted by S2-Net, results in a speedup of 23.72X. Note that S2-Net incorporates the first optimization only (i.e. aspect-ratio output FIFO). In addition, the performance gained by incorporating the first three optimizations to S2-Net (denoted by S2-Net-O3) is estimated to be 117.37X. Finally, the accelerated "reduced" HMAX gave an overall speedup of 110.44X, outperforming the S2-O1-4, although it is mapped to a single FPGA only.

In addition, a GPU implementation of a sparse version of HMAX from [1] is compared to the FPGA implementation. This version has only 4075 prototype patches and all 12 orientations for every scale are condensed into a single orientation. It takes 0.291 seconds per image to execute on Nvidia GTX 295 GPGPU. In contrast, when mapped to a 4-FPGA platform, an accelerated sparse version of the HMAX executes in 0.277 seconds, giving a speedup of 5.1% over the GPU implementation. However, power measurements show that the FPGAs running at 24 Watts has a clear advantage when compared to the GTX 295 running at TDP of 289 Watts [11].

6 CONCLUSION

This paper presented a hardware architecture for accelerating HMAX, a cortical model for object classification. Although the focus was on S_2 acceleration, which is the bottleneck, all other stages in the HMAX model were also mapped to the FPGA device.

The S_2 accelerator was evaluated, and a number of optimizations were proposed to improve its execution time. Moreover, an alternative design to the S_2 accelerator was discussed. For small size patches (i.e. 4x4 & 8x8), this design allows streaming a single image while computing the correlation against multiple prototype patches, hence parallelizing the operations of the S_2 stage.

The classification results were presented and it was shown that the accuracy of the accelerated HMAX matches that of the CPU implementation. Furthermore, a "reduced" version of the HMAX model was implemented and found to have a much faster processing rate at the expense of reduced accuracy. These tradeoffs between speedup, power and resources on one side and accuracy on another can be utilized for embedded systems with limited resources.

For future work, further optimizations to the S_2 accelerator will take place. For example, although the T_{coeff} contributes by only 3% of the total execution time, it occurs with high frequency. This latency can be hidden by double buffering the prototype patches.

Acknowledgment

This work is funded in part by DARPA's NeoVision 2 program. Ahmed Al Maashri is sponsored by a scholarship from the Government of Oman.

7 REFERENCES

[1] J. Mutch and D. G. Lowe, "Object class recognition and localization using sparse features with limited receptive fields," *International Journal of Computer Vision (IJCV)*, vol. 80, no. 1, pp. 45-57, October 2008.

[2] L. Itti, C. Koch, and E. Niebur, "A model of saliency-based visual attention for rapid scene analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 11, pp. 1254-1259, November 1998.

[3] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in cortex," *Nature Neuroscience*, vol. 2, no. 11, pp. 1019-1025, November 1999.

[4] Xilinx, "7 Series FPGAs Overview," DS180(v1.5) 2011.

[5] S. Kestur, D. Dantara, and V. Narayanan, "A Streaming Model for FPGA Accelerators and its Application to Saliency," in *Design Automation and Test in Europe (DATE)*, France, 2011.

[6] C. Farabet, C. Poulet, J. Y. Han, and Y. LeCun, "CNP: An FPGA-based processor for Convolution Networks," in *Field Programmable Logic and Applications*, 2009, pp. 32-37.

[7] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust Object Recognition with Cortex-Like Mechanisms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 3, pp. 411-426, March 2007.

[8] Nallatech Inc. (2011) [Online]. http://www.nallatech.com/Intel-Xeon-FSB-Socket-Fillers/fsb-development-systems.html

[9] Xilinx, "Virtex-5 Family Overview," DS100(v5.0) 2009.

[10] L. Fei-Fei, R. Fergus, and P. Perona, "Learning generative visual models from few training examples: an incremental Bayesian approach tested on 101 object categories," in *IEEE Workshop on Generative-Model Based Vision, CVPR*, 2004.

[11] Nvidia. (2011) GeForce GTX 295. [Online]. http://www.nvidia.com/object/product_geforce_gtx_295_us.html