An FPGA-based Accelerator for Cortical Object Classification

Mi Sun Park, Srinidhi Kestur, Jagdish Sabarad, Vijaykrishnan Narayanan, Mary Jane Irwin

Dept. of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802
{mup183, kesturvy, sabarad, vijay, mji}@cse.psu.edu

***Abstract*—Recently significant advances have been achieved in understanding the visual information processing in the human brain. The focus of this work is on the design of an architecture to support HMAX, a widely accepted model of the human visual pathway. The computationally intensive nature of HMAX and wide applicability in real-time visual analysis application makes the design of hardware accelerators a key necessity. In this work, we propose a configurable accelerator mapped efficiently on a FPGA to realize real-time feature extraction for vision-based classification algorithms. Our innovations include the efficient mapping of the proposed architecture on the FPGA as well as the design of an efficient memory structure. Our evaluation shows that the proposed approach is significantly faster than other contemporary solutions on different platforms.**

I. Introduction

The functioning of the brain has long fascinated researchers. Reverse engineering the brain is considered as one of the grand challenges of engineering. The ability to reverse engineer the brain promises computing systems that are capable of cognitive functions. While current systems excel in several applications, computer systems are still in infancy in cognitive activities. There has been recent progress in such systems that outperform humans in quiz competitions [1] and Chess [2]. While these represent significant strides forward for such cognitive systems, the size of systems deployed to support these applications and the narrow domain of end applications is a dampener for immediate wide-spread adoption.

In contrast, the ability to harness the significant advances in understanding the human visual cortex provides an opportunity to impact a variety of smart vision applications. There are various biologically inspired recognition algorithms that have been proposed including Convolutional Neural Networks (CNN), Deep Belief Networks, and HMAX [3]. These biologically-based models are robust to wide variations in scale, lighting, pose, background clutter and are more suitable for general purpose object classification with multiple classes than traditional computer vision approaches. In this work, we focus on the HMAX algorithm [4] which is a widely accepted model to abstract the behavior of the visual cortex. In the HMAX model, a hierarchy of features is extracted by the two stages of convolutional template matching and pooling operations. At the S1 stage, the image is convolved with a Gabor filter to extract information at different orientations at different scales. The first pooling layer, C1 is used to improve invariance to the input image through application of local maximum operators. The second template matching stage, S2 is the most computationally intensive portion of HMAX and applies learned patches to convolve with the features extracted from the C1 layer. The final pooling stage considers all scales and orientation to extract a feature vector C2 that is then provided to a classifier. HMAX has been shown to be more successful in classifying objects from a large number of classes with accuracies ranging around 70-80%.

978-3-9810801-8-6/DATE12/©2012 EDAA

The ability to classify objects in real-time has a variety of end applications including security surveillance, remote elder care monitoring and unmanned aerial vehicles. A key challenge is that most existing solutions for accelerating HMAX have been either performed on platforms that are not amenable for embedded systems either due to their low performance or high power consumption. These efforts include the implementation of a CNS framework on GPUs and accelerators for HMAX on FPGAs [5] [6] [7]. In this work, we choose a FPGA platform to implement our design due to its configurable nature as well as the ability to customize the underlying architecture to the accuracy needs of the application. The configurable nature of the FPGA is particularly useful due to the different variants of HMAX and refinements to the underlying HMAX model that are emerging in this active research area in neurosciences [8]. Further, one can tune the data bitwidth and the pipeline to trade-off the required performance, power and accuracy needs of the classification application.

This paper makes the following contributions towards accelerating HMAX algorithms.

- An adaptable processing element (PE) that can be configured either dynamically or statically is proposed. This feature enables efficient support for different variants of HMAX template matching (sparse and dense) as well as different stages (S1 and S2).
- The PE design has been tuned to exploit the underlying features of a Virtex-6 platform by mapping to an high-speed pipelined DSP48E1 slice, and these PEs are tiled through dedicated cascade chain.
- A 2D systolic array that can be dynamically configured to support convolutions of different kernel sizes is proposed (such as 4x4, 11x11, 16x16).
- A customized memory hierarchy is proposed to efficiently access the template patches and the input image and reduce the impact of costly memory accesses. We also propose scheduling techniques for mapping template patches when executing on multiple pipelines.

We have developed a prototype of the accelerator on a Xilinx Virtex-6 FPGA and experimental results shows 31X and 2X to 107X speedups over existing GPU and FPGA implementations respectively.

II. Related Work

There has been a growing interest in designing systems for cortically inspired recognition algorithms. Hardware accelerators for CNN based visual recognition has been pro-

posed [3] [9]. Several recent approaches to speedup brain-inspired visual algorithms such as attention maximization algorithms [10], saliency extraction [11] and retinal image enhancement techniques [12] were proposed and implemented on an FPGA. Prior works on accelerating HMAX on FPGA platform and GPU platform [5] [7] [13] were also proposed. However the existing solutions for accelerating HMAX are not amenable for embedded systems either due to high power consumption or their low performance, therefore we propose an energy-efficient, high-performance FPGA-based accelerator for HMAX which outperforms the prior works.

III. HMAX: A Cortical Model for Object Classification

HMAX is a biologically inspired model that mimics the hierarchical feed-forward organization of the first few stages of the visual pathway in primates. In its simplest version, the model consists of four layers of alternating S (convolutional) and C (pooling) computational units, referred as S1, C1, S2 and C2. These stages extract a feature vector from a grayscale image that can then be passed to a classifier for object classification. With more understanding of the brain and differences in modeling them, variants of the HMAX model have emerged [8] [14]. The goal of this work is to design an architecture to accelerate the most computationally intensive stage, S2, of HMAX to achieve optimal performance. Further, we recognize that a merged S2C2 system can reduce memory access and propose an efficient architecture of the combined S2C2 system. While not elaborated in this work, our accelerator design also supports Gabor filter acceleration (S1 layer) as mentioned in Section IV. We provide an overview of the S2 and C2 layers that account for more than 90% of the total execution time of the HMAX model while running on CPU [5].

In the S2 and C2 stages, we can classify HMAX into two variants - *dense* and *sparse*. The *dense* version is the base model in [15] and the *sparse* version is the model which employs sparsification [15]. The S2 features are calculated by template matching every position and scale of the C1 pyramids (referred as X_s) with a dictionary of pre-computed patches (P_i), which is given by the following Gaussian Radial Basis Function (GRBF).

$$R(X_s, P_i) = exp(-\frac{\| X_s - P_i \|^2}{2\sigma^2\alpha}) \quad (1)$$

where $\alpha = (M_i/4)^2$, M_i is the size of the patches (aka prototypes) and σ is set to 1. The *dense* patches are of size M_i x M_i x N_θ, where $M_{i=\{1,..4075\}} = \{4, 8, 12, 16\}$ and N_θ is number of orientations (from 4 to 12), whereas the *sparse* patches are of size M_i x M_i and each coefficient in the M_i x M_i patch has a preferred orientation.

The C2 stage is a global max across scales for each of the 4075 pyramids, to obtain a single vector, each value of which is response to a given patch. The C2 module stage contributes to a massive reduction in the amount of data since it reduces an entire image pyramid after S2 to a single element in C2.

To implement this algorithm in hardware, the exponential computation can be moved after C2 to minimize the number of exponential operations. This would require doing an across-scales Global Min operation (instead of Global Max) in C2. The exponential is then applied on this reduced data to obtain the C2 feature vector.

IV. Architecture of HMAX Accelerator

The proposed architecture supports multi-level configurability to compute convolutions in both S1 and S2 layers, in addition, it also supports S2 computations for both sparse and dense patches.

At the lowest level of our architecture hierarchy is the processing element (PE) depicted in Fig 1. This PE can be dynamically configured at run-time to operate as a Gabor filter (for S1) or a sparse, or a dense GRBF (for S2) filter. This feature is supported by enabling or disabling the components inside the PE. For example, configuring the PE to operate as a Gabor filter, disables the n:1 multiplexer and the subtractor in the PE as shown in the Fig 1. In the case of sparse GRBF operation, all the components inside the PE are enabled, the n:1 multiplexer selects the input pixel depending on the precomputed dominant orientation. The subtractor is for computing the Euclidean-distance between the C1 input pixel and the patch as seen in equation 1. In addition, any component inside the PE can also be undefined at compile-time, which helps save resources.

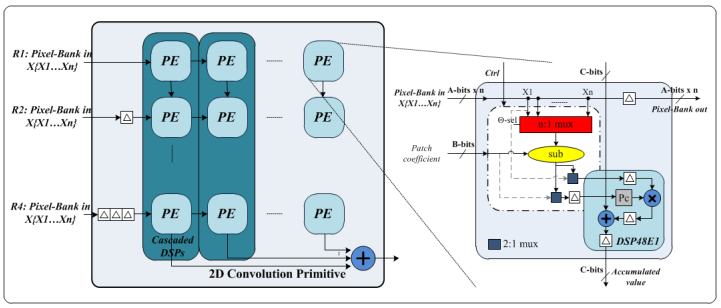

Fig. 2. The hardware (4x4) primitive for HMAX

At the next level, we cascade the PEs in two-dimension to form a 2D (4x4) systolic array (denoted as primitive, henceforth and depicted in Fig 2). This primitive serves to support convolution variants of different sizes required for S1 and S2 operations. In addition to the benefits of the regular systolic data flow, our design exploits the cascaded DSP48E1 slices on the FPGAs for enhanced performance of this primitive. Specifically, the multiply-accumulate part of the PE was optimally pipelined to match the DSP48E1 structure. Further, each column of the array was mapped to the column-wise cascaded DSP48E1 slices in an FPGA to enhance resource utilization efficiency and reducing wiring delays. Every clock cycle, the primitive receives 4 input pixels and performs 16 multiply-accumulate operations with a set of preloaded patch coefficients. In the next cycle, the input pixels are shifted to neighboring PEs to the right, while the multiplied values are passed to PEs in subsequent rows for accumulation. We can also tile PEs to form a 2D primitive of any size, however our choice of 4x4 was influenced by the smallest patch size used in HMAX.

At the next level, the 4x4 systolic primitives form a reconfigurable convolution engine (RCengine) shown in Fig 3. that can be configured either as 16 independent 4x4 convolutions, 4 8x8 convolutions, 1 12x12 convolution or 1 16x16 convolution. The RCengine has three components which can be configured at run-time to support these configurations. (1) It contains programmable delay elements at the output of each 4x4 array to introduce appropriate delays for accumulation at the adder tree when composing them to operate as a larger size primitive. (2) The number of inputs to the adder tree changes with the

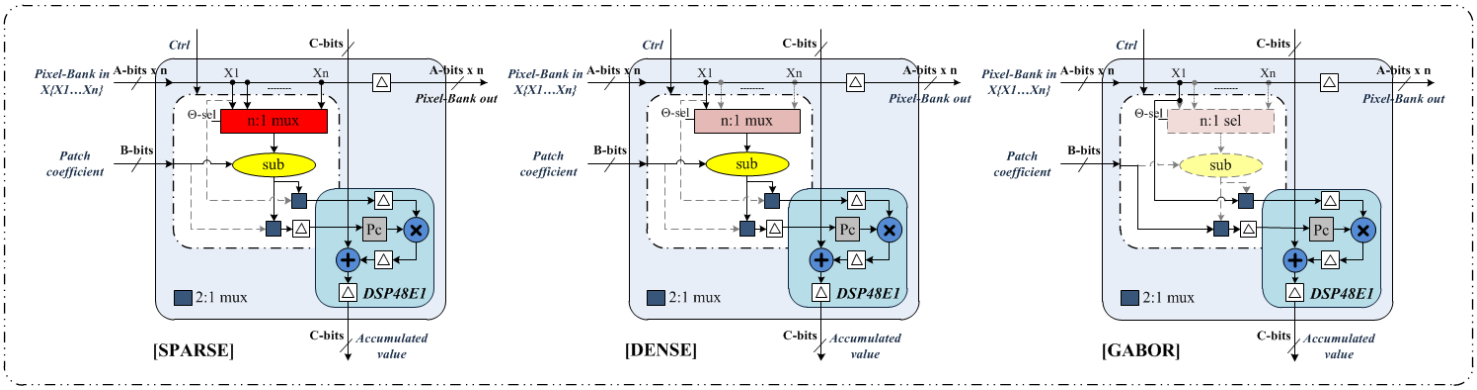

Fig. 1. Processing Element in HMAX primitive

mode - 16 for 16x16, 9 for 12x12, 4 for 8x8 and 0 for 4x4. (3) The configurable routing fabric (CR) which includes a bank of multiplexers selects the appropriate pixels to feed each 4x4 primitive. For example, in the 4x4 mode, all the primitives operate on rows i to i+3 and so CR broadcasts these values to all 4x4 arrays. In the 8x8 mode, the primitives P1, P2 operate on rows i to i+3, while primitives P5, P6 operate on rows i+4 to i+7. The RCengine supports run-time operations of all patch sizes up to 16 by computing with zero-padded patch coefficients for some patch sizes which are not multiple of 4. The output throughput is 16 pixels/cycle for the patch sizes of (1x1, 2x2, 3x3, 4x4), 4 pixels/cycle for (5x5, 6x6, 7x7, 8x8) and 1 pixel/cycle for patch sizes between 9x9 and 16x16. Further, the reconfiguration latencies of the RCengine are minimized to support iterative processing of different set of patches. The RCengine (including the 256 patches coefficients) can be completely reconfigured in about 18 cycles for a different mode.

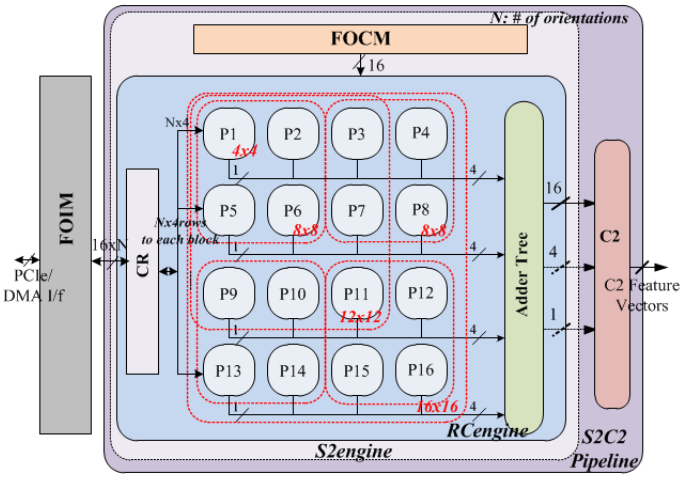

Fig. 3. S2C2 Accelerator

A. Memory Subsystems

Due to the large amount of data involved with HMAX computations, the data movement is a significant contributor to overall performance. In order to address this, we have designed customized on-chip memory architectures for both the image data and the prototypes by utilizing BRAMs on the FPGA.

1) Fast On-chip Image Memory (FOIM): In the S2C2 accelerator, the motivation for on-chip storage of image data are two-fold :

- The RCengine requires access to 1 pixel each from up to 16 consecutive rows of the C1 image every cycle depending on the mode it is operating. Moreover, each pixel can have up to 12 orientations. This makes the required bandwidth much greater than what the external interfaces (DDR memory/PCIe) can support.
- Since, the number of patches in the S2 stage is typically high, and a single RCengine can process at most 16 patches (4x4 case) in one pass, the S2C2 accelerator requires multiple passes to complete the GRBF computation on a single scale of the C1 image pyramid. This implies there are large number of repeated accesses to the same C1 image.

Our high-bandwidth and low-latency on-chip memory architecture addresses these bottlenecks. The FOIM utilizes bank-interleaving to allow multiple reads per cycle. The rows of the image are interleaved to up to 16 banks and each bank has independent read/write ports. The FOIM also has multiple columns where each column stores corresponding pixels of different orientations.

Each cycle all columns of all the banks are read to allow reading 16 pixels per cycle from 16 contiguous rows, (we use 16 since it is the maximum size of patches). The FOIM also handles the sliding window data access for convolutions by implementing an intelligent address generator to emulate raster scan order. The FOIM is reconfigured at run-time to change the image size to support the different scales of the C1 pyramid.

Each scale of the C1 pyramid is read repeatedly from the FOIM to complete the processing of all the patches. The above process is further repeated for all scales of the pyramid to complete the S2 computation for one image.

2) Fast On-chip Coefficient Memory (FOCM): The S2 stage involves processing a large number of patches (up to 5000), multiple times (up to 11) for a single image and this operation is repeated for each input image. This requires initializing the RCengine pipeline with a new set of 256 patches before loading the image into the pipeline every iteration. For optimal performance, we incorporate an on-chip memory architecture to store the patches.

The FOCM employs bank-interleaving (with 4 banks) and multiple columns (4 columns) to allow 16 reads per cycle. This enables the initialization of the 256 coefficients for the RCengine in just 16 cycles. Each iteration, the FOCM initializes the RCengine pipeline with a unique set of patches. This loading of patches into the FOCM is done once at configuration time.

The FOCM not only minimizes the initialization latency of the pipeline for each iteration, but also enables a high degree of flexibility and data reuse. Any change in the number of patches, distribution or the actual prototype coefficients can be accommodated without having to modify the hardware. The FOCM also generates a control header to the pipeline before every iteration, which includes information on the mode of operation, number of valid patches etc which are used to configure the RCengine.

B. Architecture for Sparse and Dense HMAX

The overall architecture for the S2C2 accelerator as shown in Fig 3, includes the FOIM as the high bandwidth image memory and an S2C2 pipeline. The S2C2 pipeline includes a RCengine module for the GRBF computation which has a CR module to route data from the FOIM into each (4x4) primitive, an FOCM to feed coefficients and handle per iteration control and a C2 module to compute the C2 vector.

This architecture is for the Sparse S2 which can complete processing all the orientations in one pass. However, in the Dense S2, each individual orientation of each patch needs to be applied independently on its corresponding orientation of the C1 image and the results combined before computing the C2 vector. So, the GRBF computation of img[θi] with prototype[θi], must be repeated for all orientations ($N\theta$).

Also, the dense patches are essentially 3-dimensional arrays and require much larger memory. In this case, we use the FOCM to store only the 4x4 and 8x8 patches and fetch the 12x12 and 16x16 patches from DDR memory. A portion of the FOCM memory space is used as a cache for these patches. This is because, the 12x12 and 16x16 patches are not used for the smaller pyramid levels (upper 4 scales). For the remaining pyramid levels, the image sizes are relatively larger and computation time is found to mask the DDR memory access time.

1) Custom Instruction-set for S2C2: The accelerator includes an on-chip instruction queue to store accelerator specific instructions. As shown in Fig 4, for each iteration, the S2C2 pipeline fetches an instruction from the queue, which is basically a control command to configure the pipeline for that particular iteration. It includes the mode of operation and number of valid patches for the RCengine, level information for C2 and additional bits for sparse/dense selection and orientation selection for dense. The RCengine further configures its internal primitive delay elements, adder tree and CR based on the mode of operation. The entire reconfiguration process for each iteration takes about 4 cycles.

The number of instructions in the queue is equal to the number of iterations. Once all instructions are executed, the next C1 pyramid level is fetched and the same set of instructions are re-executed. The instruction queue is hence implemented as a circular FIFO buffer. However, as we move up the pyramid, not all iterations need to be processed (for smaller images many patches are not executed) and this is controlled by configuration registers in the pipeline.

The instruction sequence is generated off-line in software and loaded into the queue through the host interface, once during configuration time. At run-time, there is no intervention from software/host and the accelerator can independently operate on streaming images.

V. Optimizations and System Integration

This section describes two architectural enhancements which improve performance of the accelerator and integration aspects of the system.

```
for scale (s = 0, s < 11, s + +)
  for iterations (i = 0; i < 1000; p + +)
    Load Instruction from Queue into S2C2 pipeline (4 cycles)
    Load patch coefficients into RCengine (16 coeffs per cycles)
    Read image from the FOIM into S2C2 pipeline (16 pixels per cycle)
  end
end
Compute Exponential and Normalize and to obtain C2 vector
```

Fig. 4. Pseudo-code for S2C2 Accelerator

A. Multiple Pipelines - Sparse v/s Dense

The performance can be improved by instantiating multiple parallel S2engines (aks pipelines) to perform the S2 computation as shown in Fig 5. Since all pipelines operate on the same image data, the output bus from FOIM is broadcast to all S2engines and so there is no resource overhead for image data access.

Since the image is broadcast to each pipeline, the patches operating on each pipeline must be unique. So, each S2engine must have a dedicated FOCM. The patches can be partitioned into multiple pipelines such that for each iteration, each FOCM initializes its corresponding pipeline with a unique set of patches. This partitioning and loading of patches into each FOCM is done offline in software once at configuration time.

For Sparse HMAX, each pipeline operates on all orientations of a prototype simultaneously and hence each pipeline is connected to its own C2 module and generates a C2 vector. However, for Dense HMAX, each pipeline operates on a different orientation of the same C1 image. Hence, each prototype must be partitioned such that individual orientations are loaded into FOCMs of different pipelines. Another complication for Dense version is that, a pipeline adder tree is required to combine the outputs of the multiple S2engines and a common C2 module is used on this combined S2 image to generate a single C2 vector.

A simple partitioning mechanism would be to balance the load on each pipeline and by operating each pipeline in the same mode for each iteration. If not, the pipeline that operates on smaller patch takes the longer time to complete the iteration and becomes the critical path. This partitioning can be done off-line as well.

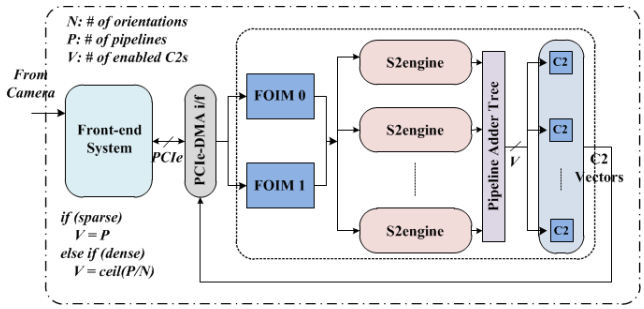

Fig. 5. Optimized Overall System

B. Memory Optimization

In order to mask the initialization/load time for the FOIMs, we duplicate the FOIM so as to implement a double-buffer architecture. This allows pre-fetching the next C1 image while the current one is being processed.

VI. Experimental Results

We use the Dinigroup DNDualV6PCIe FPGA platform [16] for prototyping our accelerator. The board has two Virtex-6 SX475T FPGAs which can be used as compute devices. The operating frequency of 100MHz for the FPGAs and 256x256 grayscale images are used in our experiments. We have validated our S2C2 accelerator running on both the FPGAs. In addition, the board power is measured with P440 Kill A Watt power meter. The static power consumption is 26.5 Watt, while the dynamic power of our S2C2 accelerator running on both the FPGAs is $26.5 \sim 27.5$ Watt.

Our software system takes prerecorded video files (.avi) or live video from a camera and generates video streams to communicate with the hardware via USB, Ethernet or PCIe. The software also preconfigures algorithmic and architectural parameters such as image size, # of FPGAs or types of operations (sparse/dense or Gabor) and generates the configuration data and instruction sequence required by the accelerator. The software optimally partitions the patches to the FOCMs in multiple pipelines to improve performance. In order to support arbitrary-sized convolutions, the software also appropriately zero-pads the coefficients for those cases where patch sizes are not multiple of 4.

A. Performance

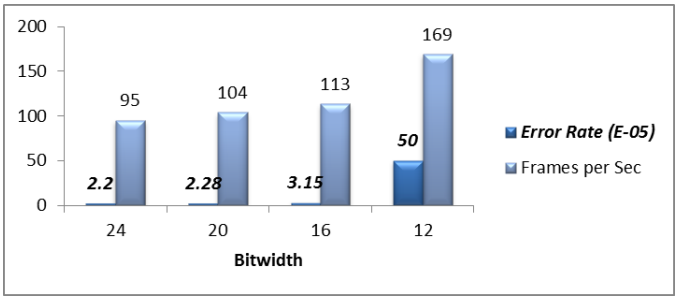

Fig. 6. Performance for various Bitwidth

Fig 6 shows the trade-off between performance and accuracy for various input bitwidth of the fixed-point implementation. Since a Xilinx DSP48E1 slice has an in-built 25x18 multiplier, for the GRBF computation we pass only 18-bits of the subtracted value into one of the slice inputs for 24-bit and 20-bit cases to use minimum # of DSP48E1s. Each RCengine uses 256 DSP48E1 slices for all cases. The error rate of C2 feature vectors is compared to a floating-point MATLAB implementation and the performance based on the maximum frequency and the # of pipelines is shown. The performance is higher for smaller bitwidths due to lower resource utilization which allows more parallel pipelines and higher frequency. We use 24-bit input images to maintain a high classification accuracy in all other experiments. This results the classification accuracy about 97% on 56 images set.

B. FPGA Resources

The performance of the accelerator depends heavily on the number of pipelines, which further depends on the algorithm parameters. We consider four separate algorithmic variants - sparse or dense and different number of orientations (N_θ). The table I shows the resource utilization for each variant. The number of pipelines that can fit is primarily controlled by the BRAMs and DSPs in Dense, whereas by LUTs and BRAMS in Sparse. For Sparse, due to the significant use of the n:1 multiplexer in each PE, LUTs resources rather than DSPs are the major restriction on the number of pipelines we can apply in a single FPGA.

For Dense, we undefined the multiplexers at compile-time to improve the efficiency of hardware resources. This allows up to 6 pipelines to be fit on a single FPGA. When N_θ is 4, we use 4 pipelines so that we can compute all orientations of a patch in one iteration. But, when N_θ is 12, we use 6 pipelines and need 2 iterations to compute all orientations of a patch.

TABLE I
Resource Utilization on Virtex6 SX475T

	Slice Regs	Slice LUTs	BRAMs	DSP48E1	# S2engines
Dense 4Θ	173,704 (29%)	134,700 (45%)	535 (50%)	1,024 (51%)	4
Dense 12Θ	256,672 (43%)	197,252 (66%)	803 (75%)	1,536 (76%)	6
Sparse 4Θ	275,336 (46%)	238,860 (80%)	723 (68%)	1,024 (51%)	4
Sparse 12Θ	279,968 (47%)	212,908 (72%)	746 (70%)	512 (25%)	2

1) Scalability: Fig 7 shows that performance scales well with using multiple FPGAs, due to the optimized local memory applied in our architecture. The accelerator architecture on each FPGA is identical and only the patches are partitioned across pipelines on multiple FPGAs. This chart shows that the design is scalable and more resources imply higher degree of parallelism from the accelerator. The Sparse versions are faster than Dense due to reduced amount of computations and smaller number of orientations implies better performance.

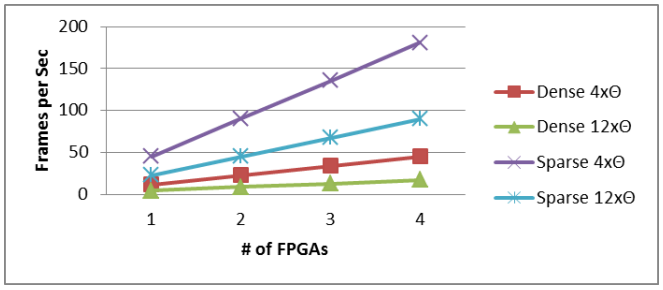

Fig. 7. Performance Scaling with number of FPGAs

C. Influence of Double-Buffering

The performance improvement from introducing the double-buffer architecture for FOIMs is listed in Table II. As shown, the impact of the improvement on load latency is more in Sparse versions due to reduced computations and higher for larger N_θ due to increase in sample size. In addition, the number of patches also matters, since the load latency can occupy a significant portion of the total time when the # of patches is small.

TABLE II
Performance Improvement with dual-FOIMs

# Patch	Dense 4Θ	Dense 12Θ	Sparse 4Θ	Sparse 12Θ
4075	0.2%	0.6%	0.8%	1.2%
1024	3.7%	4.2%	3.7%	10.3%

D. Comparison with Related Efforts

In Table III, we compare the performance of our HMAX S2C2 Accelerator with existing implementations. For the Sparse case, we use the CNS framework [6] on a GPU for comparison. The CNS (rev.372) numbers were obtained using a single NVIDIA Tesla C1060 GPU running CUDA 3.0 at 1.3 GHz. This was hosted on a Linux machine with dual quad-core Intel(R) Xeon(R) CPU running at 2.27Ghz and with 12GB RAM. We used the same set of patches on both the

TABLE III
COMPARISON OF HMAX IMPLEMENTATIONS

SPARSE 4Θ with 4075 prototypes (Frames Per Sec)				DENSE 12Θ with 5000 prototypes (Frames Per Sec)		
Telsla C1060 GPU [6]	4 x Virtex5 SX240T [5]	2 x Virtex6 SX475T	2 x Virtex6 SX475T [7]	Quad-core Xeon CPU [5]	4 x Virtex5 SX240T [5]	2 x Virtex6 SX475T
2.8800	3.6100	90.3880	45.84	0.0045	0.0909	8.4996

GPU and the FPGA platforms. Our accelerator running on two Virtex-6 FPGAs provides 31X speedup over the CNS implementation on GPU for the Sparse 4Θ case. Further, the power measurement shows that our FPGAs running at 27.5 Watt consumes around 85% less power compared to the GPU running at TDP of 187.8 Watt [17].

We also compare our performance with two recent FPGA accelerators [5] [7]. Our two-FPGA system outperforms their Nallatech four-FPGA system and the similar work on the Virtex6 SX475T by 25X and 2X respectively. Comparison with [5] is reasonable since we use only 2 FPGAs compared to their 4 FPGA system to offset the 2X more resources on our Virtex6 device. In addition, our speedup over [7] is 2X, although we used the same # of patches, the same frequency and the same Virtex6 FPGA platform.

The significant performance improvements compared to GPU and FPGA [5] implementations is due to the resource efficient compute elements, high degree of coarse- and fine-grain parallelism and efficient on-chip memory architectures. While the acceleration approach in [7] is similar to ours, our system benefits from custom instruction sets along-with minimized load latencies. Further, our 2D systolic primitive which utilizes cascaded DSP48E1 slices instead of logic slices to realize the convolution engine, allows more parallel pipelines on a single FPGA.

For the Dense case, we compare our results with CPU and FPGA numbers from [5]. We configure our accelerator with the same number of patches and 12 orientations. The table III shows that our system provides a speedup of 2164X and 107X respectively over the 3.2 GHz quad-core Xeon CPU and the four-FPGA Nallatech system. This huge speedup over [5] for dense S2C2 is specifically because their system relies heavily on access to DDR memories and also due to lack of reuse. In contrast, our system can operate 6 orientations simultaneously and maximize reuse of the image data and patches from on-chip memories.

VII. CONCLUSION

Neuromorphic vision algorithms can provide significant improvements in accuracy and robustness of visual perception tasks such as attention, classification and recognition. Owing to the huge computational demands imposed by these algorithms, customized hardware accelerators are a necessity to provide sustained performance at low power budgets. In this paper, we have presented an energy-efficient, high-performance FPGA-based accelerator for HMAX model, which is widely accepted and applied in multiclass object classification and recognition.

Our accelerator introduces an adaptable processing element to support different stages of HMAX (S1 and S2) and variants (sparse and dense). It includes a hardware primitive which has a 2D systolic structure and can be dynamically configured for Gabor or GRBF of varied sizes. A significant effort has been put into designing efficient multiported on-chip memories to provide low-latency and high bandwidth for access to both image data and patches. Finally, the architecture is scalable to multiple pipelines and devices to extract higher parallelism.

A detailed evaluation of the accelerator shows several performance trade-offs with respect to bitwidths and FPGA resources. Comparisons with existing works show that our system provides large speedups over GPU (31X) and FPGA implementations (2X to 107X). This accelerator provides real-time throughput for most algorithmic variants and is the first amenable solution for deployment in an embedded system.

ACKNOWLEDGMENT

This work was supported in part by the DARPA Neovision2 program and NSF awards 1147388, 0916887 and 0903432. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the Defense Advanced Research Projects Agency (DARPA) of the U.S. Government.

REFERENCES

[1] "Ibm-jeopardy-dac2011-keynote." [Online]. Available: http://www-03.ibm.com/innovation/us/watson/

[2] "Ibm deepblue vs gary kasprov." [Online]. Available: http://www.research.ibm.com/deepblue/watch/html/c.shtml

[3] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," in *Proc. of the 37th annual International Symposium on Computer Architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 247–257.

[4] M. Riesenhuber and T. Poggio, "Hierarchical models of object recognition in cortex," *Nature Neuroscience*, vol. 2, pp. 1019 – 1025, Nov 1999.

[5] A. Al Maashri, M. DeBole, C.-L. Yu, V. Narayanan, and C. Chakrabarti, "A hardware architecture for accelerating neuromorphic vision algorithms," in *IEEE Workshop on Signal Processing Systems (SIPS)*, Oct 2011.

[6] J. Mutch, U. Knoblich, and T. Poggio, "CNS: a GPU-based framework for simulating cortically-organized networks," Massachusetts Institute of Technology, Cambridge, MA, Tech. Rep. MIT-CSAIL-TR-2010-013 / CBCL-286, February 2010.

[7] J. Sabarad, S. Kestur, M. Park, D. Dantara, and V. Narayanan, "A reconfigurable accelerator for neuromorphic object recognition," in *Proc. of Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan 2012.

[8] J. Mutch and D. G. Lowe, "Object class recognition and localization using sparse features with limited receptive fields," *Intl. J. Comput. Vision*, vol. 80, pp. 45–57, October 2008.

[9] C. Farabet, C. Poulet, J. Han, and Y. LeCun, "CNP: An FPGA-based processor for convolutional networks," in *Field Programmable Logic and Applications, 2009. FPL 2009. Intl Conf on*, sept 2009, pp. 32 –37.

[10] S. Bae, Y. C. P. Cho, S. Park, K. M. Irick, Y. Jin, and V. Narayanan, "An fpga implementation of information theoretic visual-saliency system and its optimization," *Field-Programmable Custom Computing Machines, Annual IEEE Symposium on*, vol. 0, pp. 41–48, 2011.

[11] S. Kestur, D. Dantara, and V. Narayanan, "SHARC: A streaming model for FPGA accelerators and its application to saliency," in *Proc. of Design Automation and Test in Europe Conference (DATE)*, March 2011.

[12] S. Park, S. Kestur, K. Irick, and V. Narayanan, "Invited paper: Accelerating neuromorphic vision on fpgas," 2011.

[13] A. Nere, A. Hashmi, and M. H. Lipasti, "Profiling heterogeneous multi-gpu systems to accelerate cortically inspired learning algorithms," in *IPDPS*, 2011, pp. 906–920.

[14] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust object recognition with cortex-like mechanisms," *Pattern Analysis and Machine Intelligence, IEEE Tran on*, vol. 29, no. 3, pp. 411 –426, march 2007.

[15] J. Mutch and D. Lowe, "Multiclass object recognition with sparse, localized features," in *Computer Vision and Pattern Recognition, 2006 IEEE Comp Soc Conference on*, vol. 1, june 2006, pp. 11 – 18.

[16] "Dinigroup DNDualV6-PCIE4 documentation." [Online]. Available: http://www.dinigroup.com/new/DN-DualV6-PCIe-4.html

[17] "Nvidia tesla C1060 documentation." [Online]. Available: http://www.nvidia.com/docs/IO/43395/BD-04111-001_v06.pdf