A Framework for Accelerating Neuromorphic-Vision Algorithms on FPGAs

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

Dept. of CSE, The Pennsylvania State University
University Park, PA 16802, USA
{debole, maashri, mjc324, vijay}@cse.psu.edu

†School of ECEE, Arizona State University
Tempe, AZ 85287, USA
{chi-li.yu, chaitali}@asu.edu

***Abstract*—Implementations of neuromorphic algorithms are traditionally implemented on platforms which consume significant power, falling short of their biologically underpinnings. Recent improvements in FPGA technology have led to FPGAs becoming a platform in which these rapidly evolving algorithms can be implemented. Unfortunately, implementing designs on FPGAs still prove challenging for non-experts, limiting their use in the neuroscience domain. In this paper, a FPGA framework is presented which enables neuroscientists to compose multi-FPGA systems for a cortical object classification model. This is demonstrated by mapping this algorithm onto two distinct platforms providing speedups of up to ~28X over a reference CPU implementation.**

Keywords- ***Multi-FPGA partitioning; FPGA programming; Neuromorphic vision algorithms; FPGA application mapping***

I. INTRODUCTION

The algorithmic abstractions for the visual cortex are arguably the best understood portions of the human brain. In the last three decades, neuroscientists have made a number of breakthroughs in understanding the ventral and dorsal paths of the human's visual cortex. These advances have inspired a number of algorithms for computer vision – collectively referred to as "neuromorphic vision algorithms" – which have the potential to provide an unprecedented improvement in the way computers can analyze and interpret information. Recent improvements in FPGA technology, however, are now enabling these systems to be built while meeting performance, power, and size constraints, and maintaining the flexibility required for algorithm exploration. The goals of this research are two-fold: accelerate the design exploration of neuromorphic implementations and performance acceleration of the resulting designs.

Currently, four broad categories of tools exist for accelerating the FPGA design process [2]. The first consists of those that are aimed at improving the non-recurring engineering costs (NRE) incurred from core development. Impulse C [3], Catapult C [4], Cameron [5], and Calypto [6] are all examples of tools that attempt to raise the level of abstraction from HDL. However, these tools are limited in scope, as they fail to elevate the level of abstraction beyond the individual core. The second, such as Xilinx's Platform Studio [7], provide system design methodologies similar to ASICs. With these tools, designers are provided with peripheral, bus, and application IP, however the onus is on the designer to construct the system in an appropriate fashion. Finally, the last category contains tools that attempt to provide abstraction at the system-level. Examples include Xilinx System Generator (XSG) [8] and ShapeUp [2]. In both cases, IP modules are encapsulated in a higher-level language and module parameters are provided as a means for performing operations such as static type checking. These black-box modules can then be composed either programmatically, or graphically. However, these tools do not attempt to provide standardized interfaces for IP components, nor address the issue of inter-IP communication. The fourth category includes tools that are a hybrid of categories 1 and 2. For instance, Cong et. al. [9] describes the use of AutoPilot [10], a C-to-FPGA synthesis solution, which is coupled with platforms offered by Xilinx. The authors show that using the tool yields an 11-31% reduction in FPGA resource usage compared to hand-coded designs. However, the authors did not discuss the ability of the tool to map components to Multi-FPGA systems.

This paper focuses on automation tools for transforming HMAX [11] variants onto a multi-FPGA system as well as accelerating its' performance. Work towards providing a high-level tool, Cerebrum[1], is described which standardizes multi-FPGA system specifications and uses high-level meta-data to deliver an application level design experience to the user. To accomplish this an IP-based multi-FPGA mapping algorithm is also developed that automatically evaluates the placement of IP components according to resource use, connectivity, and I/O.

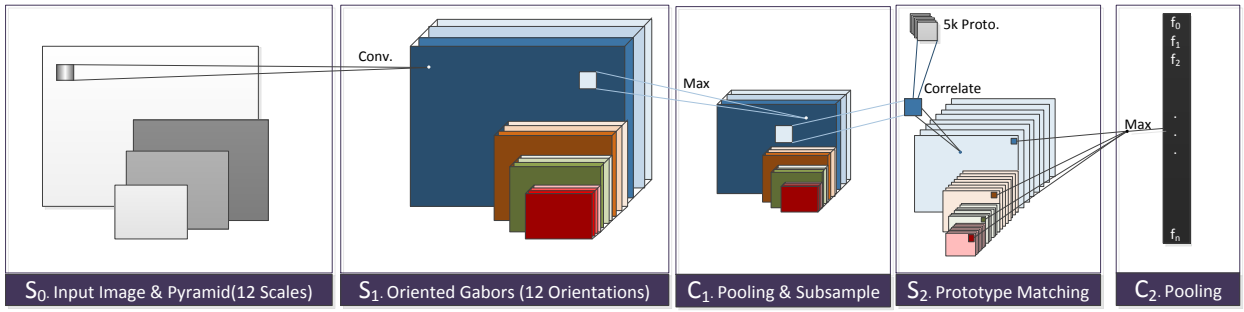
The remainder of the paper is organized as follows. Section 2 describes HMAX, a cortical model for object classification. Section 3 introduces Cerebrum, a tool for enabling users to compose systems and program dataflow through common multi-processor concepts and languages. In Section 4, a mapping algorithm capable of automatically mapping many-core designs across multi-FPGA platforms is described. Section 5 provides a case study for mapping HMAX onto two distinct FPGA platforms and demonstrates the achievable speedups. Finally, Section 6 concludes the paper.

II. HMAX

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

* This work is funded in part by the DARPA Neovision 2 program. A. Al. Maashri is sponsored by a scholarship from the Government of Oman

[1] The name Cerebrum was chosen to portray the tool as the center for all actions in creating FPGA systems, similar to the role of its biological counterpart.



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

Figure 2. HMAX Stages and Percentage of Execution (CPU)

space and scale invariant object recognition by building complex features from a set of simple features in a hierarchical fashion. The reader is encouraged to consult [11] for an in-depth treatment of the topic. The computational version used for this work was an extension developed from Mutch and Lowe [12].

A CPU reference of HMAX was implemented on an Intel server containing a quad-core 1.6 GHz Xeon processor and 24GB of system memory. The reference implementation consists of a pre-processing stage, S_0, and 4 cortical stages split into S_1, C_1, S_2, and C_2. The final output of HMAX is a set of features that can then be used for classification.

The percentage of execution time for each stage is shown in Figure 2. S_2 is perhaps the most complicated layer as it attempts to match a set of 4x4x*m*, 8x8x*m*, 12x12x*m*, and 16x16x*m* prototypes that make up a patch dictionary which can be used as fuzzy templates consisting of simple features that are position and scale invariant. 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 (in this paper, m = 12). 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) as given by the normalized dot product. Figure 1, shows the pseudo code for computing the S_2 response for a given input image.

HMAX contains several algorithmic parameters that can be tuned in order to achieve the best classification results for the application domain. A key difficulty is the effort required to integrate custom HDL IP cores in a way that creates a functionally valid system prone to rapidly changing system specifications. Therefore tools which can assist users mitigate the FPGAs high NRE costs are required.

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 1. Pseudo Code of S_2 Stage

III. CEREBRUM

One of the major contributing factors to FPGA system complexity is the dearth of tools that leverage scalable infrastructures (such as NoCs) and standardized components for multi-FPGA systems. The lack of abstraction mechanisms for FPGA accelerators makes determining and modifying dataflow a time-consuming process requiring a high-degree of expertise. Cerebrum is the result of an effort to standardize FPGA platform specifications, communication, and interfaces, as well as provide a layer of abstraction for specifying dataflow. The goal of Cerebrum is to allow neuroscientists and researchers to compose accelerators for various cortical vision algorithms with minimal engineering effort. Figure 3 describes the front end GUI and back-end EDA tool for composing FPGA systems.

A. *Cerebrum GUI*

The purpose of the Cerebrum front-end is to provide users with a graphical way to create systems and automate the back-end process for the user. During the design, the user is given access to a library of IP cores in which to create a design. Each core falls into one of two categories, stream and compute oriented computational modules. Shown below is an example XML IP core specification for a module consisting of two compute modules made up of three Xilinx IP cores:

```
<Software>
 <DesignDisplay>
  <Category Name="Nallatech Interfaces" />
   <Ports>
</Ports/></DesignDisplay></Software>
…
<Hardware>
 <Interface Type="SAP" PE="True"> nal_rx </>
 <PCores>
  <PCore Type="nal_rx" Version="1.01.a" … />
 </PCores>
…
<Clocks>
 <"100MHz Oscillator"… Frequency="100MHz"… />
</Clocks></Hardware>
```

The IP Core specification file specifies the interfaces and contents of the IP. The first section, <*Software*>, has several fields that determine how the core is exposed to the Cerebrum designer, with the most important being the port interfaces. The

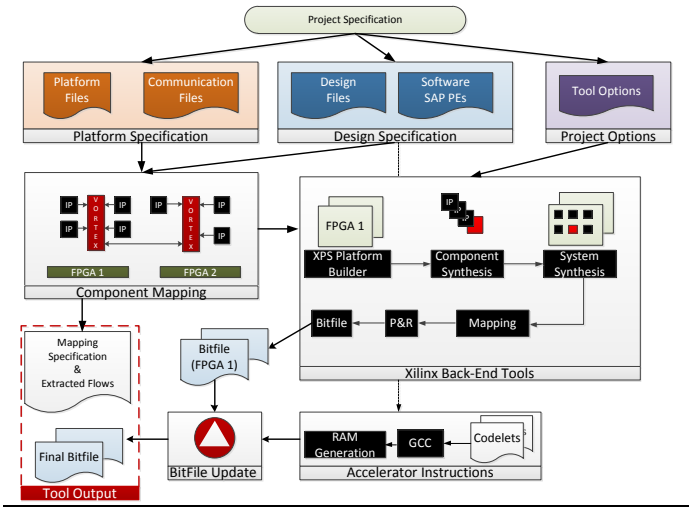


Figure 3. Cerebrum Front End GUI (Left) and EDA Flow (Right)

ports expose the cores interface, initiator/target (compute), input/output (streaming), and describe rules as to how components can be composed (for example, a target can only be connected to an initiator or output). The *<Hardware>* section on the other hand, details the internals of the IP core and is separated into two types, the *Interface Type* and *PCore* set. The *Interface Type* indicates to the back-end tools the network interface(s) that the core belongs to. The *PCore* set describes the underlying hardware components that make up the core. Lastly, there is an additional clock section that allows the components to expose clocks, or specify required clocks, *in addition to* the clock automatically being inserted by the framework.

Creating a system is then a matter of placing the cores onto the workspace and connecting them in a meaningful way. For example, compute based modules allow users to create transactions and be programmed using the C-language (e.g. codelets). Accelerator functions are provided through instruction set extension APIs that are specified along with the accelerator. Stream oriented modules are not programmable, but rather processes data as it streams between compute modules. Also, stream modules may be chained together allowing for applications to process data on-the-fly as would be desired for many real-time imaging applications. Reprogramming the data flow or which accelerator functionality is invoked is accomplished by re-writing codelets,

Table 1. Productivity Comparison Fixed-Hardware vs. Codelet

Platform (# of FPGAs)	Param	Synthesis (mins)	Codelet (mins)	Improvement (X's)
ML510 (1 FX 130T)	(1)(2) (3)	90-150	0.5-1.0	90-300
Nallatech FSB (4 SX240T)	(1) [1 FPGA]	180-360	0.5-1.0	180-720
	(1)(2) [2 FPGA]	360-720	0.5-1.0	360-1440
	(1)(2)(3) [4 FPGA]	720-1440	0.5-1.0	720-2880

(1) Number of orientations used in S_1
(2) Number of prototype patches used in S_2
(3) Number of scales used throughout S_1, C_1, S_2, C_2

which do not require synthesis. As demonstrated in Table 1, codelets substantially improve on the amount of time required to evaluate parameters when compared to a fixed implementation.

B. *Cerebrum EDA*

Cerebrum consists of three categories of projects specifications:

Platform Specification: XML files which defines the I/O, resources, interconnections, and required interfaces.

Design Specification: An XML file which describes the IP cores, their interconnections, and any design parameters.

Project Options: FPGA back-end tool options

From these sets of files, the back-end flow implements the accelerator mapping, runs the FPGAs proprietary back-end flows, compiles codelets, and merges the executables with the bitfiles. In order to standardize the communication between IP blocks, the back-end flow relies on a communication network and set of network interface modules (NIFs) to act as the underlying infrastructure. In addition, the back-end uses a multi-FPGA accelerator-mapping algorithm in order to automatically place IPs onto the FPGAs and generate the communication network.

IV. Multi-FPGA Accelerator Mapping

The multi-FPGA accelerator mapping acts to automatically insert and place each component onto a corresponding FPGA platform consisting of one or more FPGAs. Let $G_c = (V_c, E_c)$ be a graph defining the IP resource and connectivity, with $V_c$and $E_c$defined as follows:

*V_c = $\{v_j | v_j$ represents a component in the design which consumes a set of resources, R$\}$.

*E_c= $\{e_j | e_j$ is an edge from v_j to v_{j+1} and represents a communication between each node$\}$.

The physical resources of the FPGAs and their connectivity are represented with another directed-graph, $G_F = (V_F, E_F)$, with the set of vertices, V_F, and edges, E_F, defined as follows:

*V_F = $\{v_m | v_m$ represents an FPGA present in the design which provides a set of resources, R$\}$.

*E_F= $\{e_m | e_m$ is an edge from v_m to v_{m+1} and represents the interconnectivity between each FPGA$\}$. The direction of each edge defines the link direction, uni- or bi-directional.

Mapping Problem Formulation: Given the physical FPGA resource and connectivity graph $G_F(V_F, E_F)$ and the component graph $G_c(V_c, E_c)$ representing a design, find a mapping of the components, G_c, to the FPGAs, G_F, that does not exceed the resources available on any one FPGA.

Feasibility Check: A first-pass feasibility check is performed prior to mapping. During this pass it is guaranteed that enough resources exist across all FPGAs. If satisfied, the mapping proceeds through four phases: (I) Component Grouping, (II) I/O Distance Calculation, (III) Pre-mapped Allocation, and (IV) Un-mapped Allocation.

I. Component Grouping: For all G_c vertices determine if there have been any that must be placed within the same FPGA. If

Table 2. HMAX Results for two FPGA based platforms

Platform (# FPGAs)	HMAX Layer			Tot. Exec. Time
	S1/C1	S2	C2	
CPU	2.937	222.9	0.458	226.295
ML510 1 FX130	252.78 (0.012)	10.98 (20.31)	8.04 (0.057)	271.79 (0.833)
Nallatech FSB 4 Sx240T	0.922 (3.185)	8.73 (25.53)	0.143 (3.203)	8.23 (27.49)

so, annotate each group with the FPGA they are to be mapped onto.

II. I/O Distance: These distances will be used to map those components directly interfacing with IO close to the source (sink).

III. Pre-Map Allocation Place components that have been pre-mapped to FPGA's.

IV. Un-mapped Allocation A greedy approach is used to iterate through the groups and assigned to an FPGA vertex based on the available resources, and the I/O distance of each the vertices, v_m. As each vertex is visited, a check is performed to see if there are sufficient resources available. The FPGA that provides enough resources and has the lowest combined I/O distance is chosen.

V. HMAX Case Study

The Cerebrum Tool was then used to explore the implementation of HMAX onto two FPGA platforms, The Xilinx ML510 platform and Nallatech FPGA accelerator platform (Table 2). In the HMAX Ml510 platform a single FPGA was used to perform acceleration of S_2. The S_2 accelerator was able to achieve a 20X speedup over the CPU S_2 stage. Within the Nallatech platform 4 FPGAs were used were used to accelerate all stages of the algorithm. The first FPGA mapped S1, C1, and C2, while the remaining three FPGAs contains S2 modules. Using this platform the CPU reference implementation was accelerated by 27.9X over the reference CPU. A regularized least squares classifier was used to classify a dataset containing 16 different categories. The FPGA-based implementation of HMAX was used to extract features of 1382 images used in a test set, the results are shown in Table 3. The overall accuracy of the classifier is 76.63%, significantly better than random (6.25%).

Table 3. Classification Accuracy using HMAX

Class	Class	Number	Correct	Accuracy
Background	0	668	665	99.55
Boats	1	18	4	22.22
Buses	2	28	14	50.00
Cars	3	176	155	88.07
Heavy Vehicles	4	24	4	16.67
F-1 Cars	5	33	22	66.67
Helicopters	6	44	31	70.45
Military Vehicles	7	65	28	43.08
Monster Trucks	8	30	7	23.33
Pickups	9	47	13	27.66
Plane	10	61	43	70.49
Semi-Trailers	11	51	29	56.86
Tanks	12	45	13	28.89
Trains	13	32	12	37.50
UFOs	14	36	15	41.67
Vans	15	24	4	16.67
Total		**1382**	**1059**	**76.63**

VI. Conclusion

This paper presented a multi-FPGA framework for assisting neuroscientists implement algorithms on FPGAs. This framework abstracts FPGA specific details leaving neuroscientists to simply compose IP blocks to achieve the desired functionality. The framework was used to evaluate HMAX, achieving up to 28X speedup with 76% accuracy across 16 classes.

References

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

[2] Christopher Neely, Gordon Brebner, and weijia Shang, "ShapeUp: A High-Level Design Approach to Simplify Module Interconnection on FPGAs," in *18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, 2010, pp. 141-148.

[3] (2007) Impulse C. [Online]. http://www.impulsec.com,

[4] (2007) The Mentor Graphics Web Page. [Online]. http://www.mentor.com/ products/esl/high level synthesis/catapult synthesis/

[5] (2002) Cameron: Compiling high-level programs to fpga configurations. [Online]. http://www.cs.colostate.edu/cameron/

[6] (2008) Calypto's sequential analysis technology. [Online]. http://www.calypto.com/

[7] Xilinx. Xilinx Platform Studio. [Online]. www.xilinx.com

[8] Xilinx. Xilinx System Generator. [Online]. www.xilinx.com

[9] J. Cong et al., "High-Level Synthesis for FPGAs: From Prototyping to Deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 4, pp. 473-491, April 2011.

[10] Z. Zhang et al., "AutoPilot: A platform-based ESL synthesis system," in *High-Level Synthesis: From Algorithm to Digital Circuit*, P. Morawiec and A. Coussy, Eds. Heidelberg, Germany: Springer, 2008, ch. 6, pp. 99-112.

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

[12] Jim Mutch and David 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.

[13] Don Anderson and Tom Shanley, *PCI Express System Architecture*.: Addison-Wesley Professional, 2003.

[14] Xilinx, "MicroBlaze Processor Referenc Guide Embedded Development Kit (EDK 11.1)," UG081 2009.

[15] Kevin Irick et al., "A Scalable Multi-FPGA Framework for Real-Time Digital Signal Processing," in *In Proceedings of SPIE*, vol. 7444, 2009.

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