Emulating Mammalian Vision on Reconfigurable Hardware

Srinidhi Kestur† Mi Sun Park† Jagdish Sabarad† Dharav Dantara† Vijaykrishnan Narayanan†
Yang Chen† Deepak Khosla†

† Dept. of Computer Science and Engineering
The Pennsylvania State University
University Park, PA 16802
{kesturvy, mup183, sabarad, djd300, vijay}@cse.psu.edu

‡ HRL Laboratories, LLC
3011 Malibu Canyon Road
Malibu, CA 90265
{ychen, dkhosla}@hrl.com

Abstract—A significant challenge in creating machines with artificial vision is designing systems which can process visual information as efficiently as the brain. To address this challenge, we identify key algorithms which model the process of attention and recognition in the visual cortex of mammals. This paper presents CoVER — an FPGA framework for generating systems which can potentially emulate the visual cortex. We have designed accelerators for models of attention and recognition in the cortex and integrated them to realize an end-to-end attention-recognition system. Evaluation of our system on a Dinigroup multi-FPGA platform shows high performance and accuracy for attention and recognition systems and speedups over existing CPU, GPU and FPGA implementations. Results show that our end-to-end system which emulates the cortex can achieve near real-time speeds for high resolution images. This system can be applied to many artificial vision applications such as augmented virtual reality and autonomous vehicle navigation.

Keywords—FPGA; accelerator; neuromorphic; vision; recognition; saliency; HMAX;

I. INTRODUCTION

Human brain is the most powerful processor and the most efficient. Engineers have forever marveled it’s abilities and dreamed of creating artificial machines capable of emulating the brain. One significant challenge in this quest is to mimic the visual cortex to process visual information as efficiently as the brain. This paper takes an important step in this direction.

The human visual system has the ability to focus attention on to specific regions in a visual scene instead of doing an exhaustive search in a scene. This is because, in the primate brain, anatomical and functional separation between localization and identification of objects is observed: cortical areas along the dorsal stream are concerned with spatially directing attention towards conspicuous image locations (where), while areas along the ventral stream are concerned with localized identification of attended objects (what) [1] [2]. The ventral stream (figure 1) processes the images captured by the retina in an unsupervised feature extraction stage (LGN, V1, V2, V4 layers) and classifies the objects in a supervised learning stage (infero-temporal cortex or IT) [2], using attention cues from the dorsal stream.

Advances in neuroscience have enabled researchers to model the dorsal and the ventral streams of the visual cortex of primates [1]. The outlier-based bottom-up Saliency model [3] provides a computational model for attention and gaze prediction and the HMAX-based model for multi-class object recognition [4]. These biologically plausible models of vision are extremely computationally intensive and researchers have looked towards making use of custom hardware to implement these algorithms. In order to mimic the neuro-biological process or in other words be neuromorphic, small form-factor and low-power hardware is essential. Recent improvements in FPGA technology, are now enabling these systems to be built while meeting performance, power, and size constraints, and maintaining the flexibility required for algorithm exploration. Further the reconfigurable property allows to design systems which can evolve and learn just like mammals.

We present CoVER - Cortical Vision Emulation on Reconfigurable hardware - a scalable FPGA-based framework for neuromorphic vision. While prior work in this area has focused on accelerating individual algorithms, this paper describes a hardware platform for integrating multiple accelerators and a system flow to support it. We then present accelerators for models of attention and recognition in the cortex and use our framework to integrate them to realize a multi-FPGA system emulating the cortex.

CoVER includes a model of designing such accelerators and a hardware platform which consists of an interconnection network to span multiple FPGAs, which comprises of routers, repeaters and inter-FPGA links. In addition, a software front-end allows system composition and runtime configuration of the hardware. We present a detailed evaluation of the system in terms of qualitative (accuracy) and quantitative (performance). We demonstrate speedups over existing CPU, GPU and FPGA implementations for similar algorithms.

The goal of this framework is to enable algorithm-architecture co-exploration for emerging machine vision applications such as augmented virtual reality, autonomous vehicle navigation, smart cameras and surveillance. Our multi-FPGA prototype which has been successfully demonstrated can potentially be deployed in such systems.

Figure 1. Ventral stream (“what” pathway) of the human visual cortex and corresponding layers of the HMAX model

<table>
<thead>
<tr>
<th>Ventral Stream</th>
<th>HMAX Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2</td>
<td>C1</td>
</tr>
<tr>
<td>V4</td>
<td>C2</td>
</tr>
<tr>
<td>IT</td>
<td>C2</td>
</tr>
</tbody>
</table>
II. RELATED WORK

A substantial body of work has investigated the use of hardware accelerators to speedup bio-inspired vision algorithms. In our previous work [5], we proposed a model called SHARC for implementing FPGA accelerators for streaming applications. SHARC includes a hierarchical accelerator structure and a run-time reconfiguration flow and was used to accelerate the basic bottom-up saliency model [6] on FPGA. Other Saliency algorithms such as AIM(Attention based on Information Maximization) algorithm have also been accelerated using FPGAs [7]. A framework for accelerating Convolutional Neural Networks for synthetic vision systems was presented in [8]. There has been a lot of interest in accelerating the HMAX-based recognition algorithms using FPGAs [9] [10] [11]. The focus of these papers is to create a run-time configurable convolution engine for the complex S2 stage of HMAX. An FPGA prototype of a machine vision processor for V1-like algorithm was presented in [12]. In the computer vision context, [13] integrates face detection and face recognition to provide an end-to-end solution. However, there is no existing paper which talks about actually mapping end-to-end models of the biological visual stream.

Most literature on multi-FPGA systems have been targeted towards ASIC or microprocessor emulation, and few frameworks exist to aid in integration of multiple accelerators to multi-FPGA platforms. Recently, we presented a multi-FPGA framework for nonuniform fourier transform [14]. This paper presents a multi-FPGA framework for mapping cortical vision algorithms. The streaming nature of these algorithms enable simplification of the underlying infrastructure which would allow more resources to be allocated to accelerating the application.

III. COMPUTATIONAL MODEL OF MAMMALIAN VISION

In this section, we provide a brief description of two widely accepted biologically plausible models for the mammalian visual cortex.

A. Bottom-up Saliency Model for Visual Attention

The bottom-up attention model for visual saliency was first proposed by Itti [6] for spatial saliency and extended by Waldher et al. [15]. We adopt the more recent "outlier-based" bottom-up saliency model by Peters et al. [3], which incorporates temporal features as well and can be applied to gaze prediction and eye-motion tracking.

Each input frame is first decomposed by the model Retina into luminance (I) and chrominance components (RG, BY). These then pass into the model Visual Cortex where the processing subsequently diverges into as many as 12 parallel feature channels grouped into five different feature types, including static features - color (C) (RG and BY subchannels), intensity (I), and orientation (O) (four subchannels for different orientations); and dynamic features - flicker (F), motion (F)(four subchannels for different directions).

Within each of these streams, the corresponding retinal output image is first decomposed into a multi-scale feature pyramid (8 scales) by an appropriate series of filters. For the I, C and F features, Gaussian filters are applied, for the O feature, Gabor filters tuned to the corresponding orientation are applied and for motion, a reichardt detector filter is applied.

Next, local spatial competition is used to emphasize feature contrast, by computing the difference maps between pairs of pyramid scales to form a set of center-surround maps (6 pairs). Finally these center-surround maps are normalized using a max-normalizer and an across-scale combination step computes a weighted sum across scales, forming a single conspicuity map for each feature channel. Another round of normalizations and summation across feature types leads to a single saliency map, representing the output of the model. Different versions of the model can be formed by excluding all but one or a few of the the possible channels; for example a model with only static features would be formed by the C, I, and O channels. Further details are available in [3].

B. HMAX-based Model for Multi-Class Object Recognition

The recognition model starts processing by creating an image pyramid of 12 scales from a grayscale image to form the input later. The HMAX model consists of four layers of S1 (Gabor filter), C1 (Local invariance), S2 (Intermediate feature) and C2 (Global invariance), and extracts a feature vector which is passed to a classifier for final object classification.

The S1 layer is computed from the input image layer by applying Gabor filters at multiple orientations at each possible position and scale [4] [16]. The C1 layer computes position and scale invariance over local regions by convolving the S1 pyramids (of the same orientation) with a 3-D Max filter at every 10x10 units in position and 2 units deep in scale.

The S2 layer performs template matching at every position and scale of the C1 pyramids (4 pyramids of 11 scales) with a dictionary of up to 4075 patches to calculate the intermediate features. The patches are of size n x n, where n ∈ {4, 8, 12, 16}, and each coefficient in the patch has a preferred orientation of the C1 pyramids. The response of the C1 pyramids X to these patches P is given by the following Gaussian Radial Basis Function (GRBF),

\[ R(X, P) = \exp\left(-\frac{||X - P||^2}{2\sigma^2} \right) \]  (1)

where the standard deviation \( \sigma \) is set to 1 and the normalizing factor \( \alpha \) is \((n/4)^2\).

The last layer in the HMAX model is the C2 layer, which computes a global max across scales for each of the 4075 S2 pyramids to obtain a 4075 feature vector, each value of which is response to a given patch. Further details are available in [4].

C. Integrating Attention and Recognition

Recognition algorithms without detection involves an exhaustive search approach where a detection window is shifted sequentially over the entire image. This is computationally expensive. In comparison, the cortex uses attention cues to drive recognition. Hence, a biologically plausible model would use an attentional front-end to filter only the interesting areas of a visual scene to be processed by recognition [1].
IV. CoVER FRAMEWORK

In this section we describe our framework for mapping cortical vision algorithms to FPGAs and also present our accelerators for attention and recognition.

The emulation system flow is shown in figure 2. A high-resolution camera captures image frames. The system includes a Saliency accelerator for attention and an HMAX accelerator for recognition. The resulting feature vectors from HMAX are used by a classifier to classify the objects. Results from the classifier are displayed. While, this provides a baseline, models of learning and perception can be included in the future.

As a baseline approach, we can include only the recognition component of the system, to detect and recognize targets in a new image. However, for emulating the cortex, a combined attention-recognition system is essential. The intermediate step between attention and recognition is the focus of attention (FOA) step where only the most salient chips are forwarded to recognition.

We now describe our model for implementing FPGA accelerators. We incorporate the hierarchical structure for the accelerator and the run-time reconfiguration flow similar to SHARC [5], but add several enhancements. Neuromorphic vision algorithms usually involve repetitive computations over multiple scales or features and hence the hierarchy and run-time configuration aspects of the model fits well.

In addition to the above features, we introduce an accelerator programming and memory model. We observe that a typical cortical vision algorithm operates on a per-frame basis and continuously repeats the same computation for each frame. We assume that an accelerator has a basic pipeline which needs to be executed one or more times per frame, by modifying some aspect of the pipeline. In such a case, we introduce a custom instruction set for each accelerator. These instructions represent a certain variant of the pipeline to be executed by performing intra-frame reconfiguration. For each frame, some or all of the instructions are executed according to a pre-determined sequence.

Hardware support is provided for this by including a circular instruction queue which can be initialized before running the application. The instruction queue can be implemented either as a FIFO (with its output data port tied to its input) or a RAM which can be repeatedly read from in the desired sequence. The instruction fields can be defined as per application needs and further the instruction decoding needs to be handled within the accelerator. This allows application-specific customizations of the instruction set.

We also include an on-chip frame buffer for each accelerator to allow reuse of an image frame for multiple iterations. This avoids going to external memory or interfaces to fetch image data.

This way, the CoVER accelerator model handles 3 different flavors of run-time reconfiguration (RTR).

- run-time initialization - configuring a pipeline before every run. This is the most common form of RTR,
- intra-frame reconfiguration - Reconfiguring a pipeline within the execution of a frame. This is typically very common in neuromorphic vision.
- inter-frame reconfiguration - reconfiguring a pipeline after execution of a frame. This is seen in neuromorphic vision when there is feedback or a learning stage.

In contrast, the SHARC model [5] supports only 1 above.

We generalize the interfaces for each accelerator by specifying a common accelerator wrapper. This wrapper exposes a unified interface for system integration. The wrapper defines 4 memory spaces addressable from external interfaces.

- Configuration memory - The RTR data stream for each hardware module in the design is loaded into this memory space. It also includes the instruction sequence to initialize the instruction queue.
- Input memory - The input frames are loaded onto this memory space, this typically goes into the frame buffer.
- Output memory - The result data from the accelerator is buffered here before they are written out.
- Slave Registers - This is a set of registers which allows the interfaces to set DMA or packet parameters.

B. The FPGA Platform

We have chosen a COTS multi-FPGA platform - DNV6F6-PCIe, by Dinigroup [17] for prototyping. The DNV6F6-PCIe is populated with up to six Virtex6-SX475T
devices for compute purposes. An on-board Marvell CPU provides support for GbE, USB and PCIe.

However, the PCIe bandwidth utilization was observed to be extremely poor due to overheads in the Marvell CPU. In order to bypass the Marvell CPU, we utilize a high-speed connector, DNSEAM-PCIe [18] to obtain direct PCIe connection to one of the compute FPGAs via the GTX expansion headers. Currently, we have a 4-lane GEN2 PCIe connection from a host PC to FPGA C on the DNV6F6 board.

C. Multi-FPGA Interconnection Network

Cortical vision algorithms are very complex and use of multiple FPGAs is key for real-time performance. In order to decouple accelerator design, implementation and evaluation of different cortical algorithms, one option is to dedicate an FPGA for each complex algorithm.

Realizing multi-FPGA systems are challenging in terms of engineering effort and skill and often they become specific to the FPGA board under use. However, they provide scalability to large-scale systems. The CoVER framework includes an interconnection network spanning multiple FPGAs. The interconnection network uses inter-FPGA links for communicating between devices. These links are typically serial LVDS links and the bandwidth is platform-dependent. On the DNV6F6 the link bandwidth between a pair of FPGAs depends on the number of pin banks that we use, which is not the same between all pair of devices. We specify a common protocol for all physical interfaces called physical interface protocol (PIP) such as inter-FPGA links, PCIe and memory and provide interpreter modules to support this protocol. Asynchronous FIFOs are used to synchronize data between clock domains.

Table I

<table>
<thead>
<tr>
<th>Dev ID</th>
<th>Dest in FPGA(s)</th>
<th>Multicast</th>
<th># Hops</th>
<th># Destinations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C</td>
<td>No</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>No</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>No</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>E</td>
<td>No</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>B, F</td>
<td>Yes</td>
<td>1, 1</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>A, F</td>
<td>Yes</td>
<td>1, 1</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>A, B, F</td>
<td>Yes</td>
<td>1, 2</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>D, E</td>
<td>Yes</td>
<td>1, 2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>A, D</td>
<td>Yes</td>
<td>1, 2</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>B, E</td>
<td>Yes</td>
<td>1, 2</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>A, B, F</td>
<td>Yes</td>
<td>1, 1, 1, 2, 3</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>D, E</td>
<td>Yes</td>
<td>1, 2, 2, 3</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>A, D, E, F</td>
<td>Yes</td>
<td>1, 1, 1, 2, 3</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>A, B, D, E, F</td>
<td>Yes</td>
<td>1, 1, 1, 2, 3</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>A, B, D, E, F</td>
<td>Yes</td>
<td>1, 1, 1, 2, 3</td>
<td></td>
</tr>
</tbody>
</table>

1) Network Virtualization: In an ideal network of accelerators, the scheduler or controller which can be an on-chip or off-chip CPU or a neighboring FPGA, needs to be decoupled from the accelerators itself. Virtualization is a handy feature which masks the network complexity to the front-end, so that the system is not board dependent.

Our framework includes a simple router for the network which not only routes data but also virtualizes the network to the front-end. We utilize one of the devices on board as a network interface to the host, so that it does minimal compute. While this may be argued as a waste of resources, it is a practical choice since running all devices at full utilization might exceed the thermal limit of the board. We then provide dedicated/shared links from this network interface to all other devices which may be one or multiple hops. Multiple-hop connections are achieved by introducing repeater modules on the intermediate FPGAs.

The router is customized for vision applications and so operates in streaming mode. It scans the frame header which provides the device ID of the destination, the target memory space (config/input/output/regs) and the frame size. It then routes the entire frame without any further scan or packetization overheads. The front-end host sets the fields in the frame header and initiates the data transfer and is oblivious to the complexities of the network. Each FPGA in the network can have a router to handle routing data to external interfaces or the on-chip accelerator.

2) Clocking: One challenge in inter-FPGA communication is working with source-synchronous clocks. In our framework, each FPGA uses Xilinx Clock managers (MMC) to generate phase-aligned parallel clock for the accelerator and serial clock for inter-FPGA links. At the receiving end, the serial clock is extracted from the data on the link and used to generate the phase-aligned parallel clock. The repeaters act as receiver and transmitter and hence have to extract the clock from the source and generate clocks for the destination. On the DNV6F6, the serial clock is 500 MHz and the parallel clock is 100 MHz.

3) Multicast Support: The network is designed to support multicast - specific device ID’s assigned to target a specific set of FPGAs as the target. Each destination may be at one or multiple hops for multicast. Similarly, broadcast is also supported. A sample routing table for FPGA C is shown in Table I. The multicast feature of the network is extremely useful in cortical vision since, often multiple algorithm stages need to operate on the same image data. In such cases the critical network interface can be used only once by enabling multicast modes.

On the DNV6F6, we choose FPGA C as the network interface since the DNSEAM PCIe is connected to this device as shown in figure 3. FPGA C can do some optional pre-processing or post-processing steps which are not resource intensive. Each of the other FPGAs can be configured with an accelerator. In the current system, we have FPGAs B and A configured with the Saliency accelerator and FPGAs F and E configured with the HMAX S2C2 accelerator. FPGA D
is assigned for S1C1 stages of HMAX. While the system is customized to the DNv6F6PCIe platform, the framework is more generic and can be ported to other multi-FPGA platforms with relative ease.

D. Architecture of Saliency Accelerator

In the saliency model, there are up to 12 channels of computation among which the common blocks are identified as primitives. They include pyramid generator using successive 2D convolutions and down-sampling, center-surround difference (CSD), image re-scaling, the max-normalizer, the across-scale adder and an accumulator.

We propose a run-time configurable and highly parallel Saliency pipeline as shown in figure 4. The idea is to execute a single feed-forward pipeline multiple times, by configuring it to compute a different channel each time around. The pipeline hence includes not only the common elements, but also feature-specific elements such as the steerable filters for orientation (O) and the reichardt detector for motion (M). The accumulator in the final stage, accumulates the conspicuity maps across all channels to finally return a saliency map.

An important contribution is the inclusion of the Retina model in hardware, which allows us to speed-up pre-processing as well as to generate the I, RG and BY inputs to the pipeline locally in hardware. In addition, we include two independent frame buffers to cache the current chip and the previous chip. The previous chip is required for the temporal channels namely motion and flicker.

Key features of this design are that all the image scales/center-surround scales are computed in parallel. Further, within the primitives, there is complete parallelism. Example: In a separable 5x5 convolution, the 10 multiplications are done in parallel, while the image is continuously streaming. This highlights the hierarchical nature of the design owing to our CoVER accelerator model.

Further, the CoVER model also allows us to utilize a simple instruction set for Saliency - one for each channel (C, I, O, F, M). Each instruction encodes feature-specific control information. Example: The motion channel (M) instruction enables reads from both current and previous frame buffers and enables the reichardt filters in the pipeline, while bypassing the steerable filters. The M instruction is executed 4 times, with a different direction encoding each time to configure the reichardt filter. Similarly, the orientation channel (O) instruction enables reads only from the current frame buffer while bypassing the reichardt filters. The O instruction is also executed 4 times, each time configuring the steerable filters with a different direction. The Flicker (F) instruction enables reads from both current and previous buffers and enables a subtractor in the retina to compute the difference between the frames. Then F bypasses both steerable and reichardt filters. The pipeline is hence executed for 12 iterations to complete the computation of all Saliency channels.

1) Down-sampling Optimization: We observe that pyramid level 0, which is the original image scale is never used by the model. So, one possible optimization is to down-sample the input 256x256 image to 128x128 right at the input, which will correspond to scale 1. If the down-sampling is done before loading the chip into the FPGA, it serves two purposes - improve the compute time by a factor of 4 and reducing the I/O overhead by a factor of 4. The 128x128 image is used as scale 1 directly so that the algorithm remains the same.

E. Architecture of HMAX Accelerator

Since S2C2 is known to be the critical stage in HMAX, we attempt to accelerate this computation. As shown in figure 5, the core of the S2C2 pipeline is a reconfigurable convolution primitive. Since the S2 stage requires large number of repetitive convolutions with 2D template sizes of 4, 8, 12 and 16, our primitive is sized to support a 16x16 convolution using building blocks of 4x4 convolutions. The basic PE is a multiply-add and an orientation selector multiplexer to select the dominant orientation to support sparsification in S2. The 16x16 primitive can be reconfigured to perform sixteen 4x4 convolutions, four 8x8 convolutions, one 12x12 or one 16x16 convolution. Our primitive exploits DSP48 cascading to minimize resource overheads in composing the pipeline.

1) Image and Coefficient RAMs: While the convolution primitive suggests high peak compute power, for sustained high performance the pipelines demand low-latency access to large amounts of image and template (patch) data. This is enabled by utilizing on-chip Block RAMs to buffer the image and patch coefficients. The convolution primitive is then reconfigured with a new set of patches by the coefficient RAM, which in this case is also the instruction memory. The S2C2 pipeline is programmed (intra-frame) multiple times until all the patches are computed.

On the other hand, the frame buffer is optimized to operate as a multi-ported memory to load multiple pixels per cycle into the pipeline. The image is reused multiple times after which the next C1 pyramid level is loaded. Sliding window access pattern is built into the address generator and hence no additional line buffers are needed to implement the 2D convolution. The size of the instruction queue can be scaled to accommodate larger patch dictionaries. The instruction sequence is generated in software and loaded into the queue through the host interface, during initialization.

The S2C2 accelerator performance scales linearly by increasing the number of pipelines. Each pipeline shares the image data but each pipeline operates with a private instruction queue. A C2 module performs a max operation over successive scales to generate the C2 vector.

F. System Integration

In addition to the multi-FPGA architecture, the CoVER framework includes a software subsystem which pre-
The software utilizes OpenCV libraries for image pre-processing and includes driver APIs for communication to the FPGA platform via PCIe, Ethernet and USB. The host software receives prerecorded video files or live video data from a camera and partitions it into multiple small chips (256x256) in each frame. Each chip is loaded sequentially to the Saliency accelerator and the resulting saliency map (16x16) is returned back to the host. The FOA algorithm in software accumulates the saliency map of all chips in a frame and then selects the most salient chips for recognition.

In the current system for HMAX, the input layer and S1 and C1 layers are implemented in software. Then the output image pyramids of S1C1 computation are transferred to the HMAX accelerator for S2C2 computation. The resulting C2 vectors from the hardware are transferred back to the host, where the software runs an SVM classifier trained using the same patches as HMAX.

V. RESULTS

In this section we present both qualitative and quantitative results from the CoVER framework. The qualitative results include the accuracy of object detection (attention) and recognition. The quantitative results include the throughput of individual accelerators for attention and recognition and the combined system throughput.

A. Attention Results

The result from the attention accelerator is shown in Figure 6. Each 512x512 frame is divided into 4 chips of size 256x256. The saliency maps from the chips are concatenated to obtain the full-saliency map of the frame. The number of objects that can be detected is equal to the product of number of chips and number of salient locations per chip. In order to minimize false alarms, the number of salient locations per chip is chosen to be 1.

We provide a performance comparison of our system with existing Saliency implementations on a Xeon CPU [3], GeForce GPU [19] and Virtex6 FPGA [5] in Table II. The GPU and FPGA implementations are the closest to our implementation since they follow the bottom-up saliency model by Itti. The CIO channels correspond to

<table>
<thead>
<tr>
<th>Performance Comparison in FPS for Saliency Impl. (640x480)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xeon 2.8 GHz</td>
</tr>
<tr>
<td>19.48</td>
</tr>
<tr>
<td>19.37</td>
</tr>
</tbody>
</table>
spatial Saliency and CIOFM is the spatio-temporal Saliency. It can be seen that our FPGA implementation running at 100 MHz provides speedup factors of 8.7X over the CPU, 1.8X over the GPU and 1.89X over the FPGA. There is no existing hardware implementation for the spatio-temporal Saliency. In comparison to the software version on CPU, we achieve a speedup of 6.6X. Further improvements in absolute performance can be achieved by simple timing optimizations to improve clock frequency and is not the focus of this paper.

B. Recognition Results

The result from the HMAX system is shown in figure 8. The C2 vector from the HMAX accelerator is used by a Classifier to classify objects of interest into 6 classes. We have validated the recognition system on a database of 53 images and the classification accuracy is shown in table III. It is important to note that, while the size and nature of the patch dictionary directly affects the accuracy, the hardware remains the same. The patches are only used for run-time configuration of the accelerator. Related efforts on HMAX acceleration [9] [10] [11] do not report qualitative results.

Table III

<table>
<thead>
<tr>
<th># of patches</th>
<th># correct images</th>
<th>% Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>11/6 (only 4x4)</td>
<td>25</td>
<td>60.04</td>
</tr>
<tr>
<td>2000 (mixed)</td>
<td>51</td>
<td>96.22</td>
</tr>
</tbody>
</table>

Table IV

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Tesla C1060</td>
<td>4xVirtex5 SX2401</td>
<td>Virtex6 SX475T</td>
<td>Virtex6 SX475T</td>
<td>Our FPGA</td>
</tr>
<tr>
<td>V1</td>
<td>2.88</td>
<td>3.38</td>
<td>22.92</td>
<td>43.199</td>
</tr>
</tbody>
</table>

Table IV provides a performance comparison between our HMAX accelerator implemented on a single Virtex6 SX475T FPGA with existing GPU [20] and FPGA implementations [9] [10]. We obtain speedups of 15X over the GPU, 12.5X over the Virtex5-based multi-FPGA platform and 1.9X over a similar Virtex6 FPGA. While the architecture is similar in [10] and [9], our performance is higher because of tight resource optimizations, such as DSP48 cascading in the primitive, which enable higher number of pipelines on a device. Further, the CoVER model supports customized instruction sets, optimally reuses hardware components by reconfiguration.

C. Full-system Results

The results for the CoVER emulation system are given in table V. The system operates on 2352 × 1724 frames captured from a high-resolution Basler camera. The system includes 3 separate configurations. In only attention mode, only the Saliency accelerator is executed and the result is just the Saliency maps of the frame. This can be used if only object detection is required. We alternate the chips into the multiple Saliency FPGAs. The key is to overlap the PCIe transfer time with computation time and this strategy works ideally for 2 Saliency FPGAs, for which the throughput is doubled when compared to a single FPGA system.

In the only recognition mode, Saliency is bypassed and an exhaustive search is applied by running the HMAX accelerator on all the chips of a frame. Since, this is computationally expensive, the performance of the system drops considerably by close to 9X. However, the two-FPGA S2C2 system almost doubles the throughput since the patches are distributed equally on to the two instances, while the input images are multicast to them.

In the attention-recognition mode, the end-to-end system flow is executed, where we assume 10 salient chips out of the 54 chips per frame are used for recognition. In the current system, the FOA and S1C1 stages of HMAX which are implemented in software dominates the run-time. Integration of an S1C1 hardware accelerator into the system is in progress and projected results show that the full-system throughput is very close to the only attention throughput. This is because the back-end recognition is decoupled from the host and does not need any input from the critical PCIe interface. The input frames need to be buffered in DRAM until the FOA algorithm forwards salient chips into the HMAX FPGAs.

Table V

<table>
<thead>
<tr>
<th>System Flow</th>
<th>Number of FPGAs</th>
<th>FPS</th>
<th>Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only Attention</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Only Recognition</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Attention-Recognition</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Table VI

<table>
<thead>
<tr>
<th>Resource Utilization on Virtex6 SX475T FPGA</th>
<th>Slices Regs</th>
<th>Slice LUTs</th>
<th>BRAMs</th>
<th>DSP48s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Saliency</td>
<td>107,246 (26%)</td>
<td>184,374 (10%)</td>
<td>894 (2%)</td>
<td>785 (10%)</td>
</tr>
<tr>
<td>HMAX S2C2</td>
<td>275,536 (46%)</td>
<td>238,360 (10%)</td>
<td>723 (88%)</td>
<td>1,024 (51%)</td>
</tr>
</tbody>
</table>

Table VI

The resource utilization for both the Saliency and HMAX accelerators are provided in table VI. The Saliency design is for the baseline 256x256 chip size. The design is BRAM intensive because both the current and previous chips are buffered on-chip. The design can easily scale to multiple pipelines so that each pipeline executes a subset of the channels. This would scale only the LUTs and DSPs, since the pipelines share the same buffer. The HMAX resource numbers are for 4 pipelines and there is still room for more optimizations.

VI. DISCUSSION

The focus of this work is to design an artificial vision system which emulates the visual cortex. Since attention and recognition are the two main components of the visual process, we focus on these steps. Modeling cortical vision is in itself an active research area and as such there exists several variants of the models that we have used. This work does not attempt to prove the biological plausibility of the models chosen, but rather uses them as a baseline to emulate the visual cortex.

A limitation of our framework are the overheads due to software. Currently the FOA algorithm and the S1C1 stages of HMAX are implemented in software. This requires the
saliency map to be returned to the host CPU, so that the software then identifies the most salient chips per frame, compute SIC1 and then transfers the C1 images to the HMAX FPGA. In addition to increased run-times, the PCIe link now becomes the bottleneck due to increased I/O.

Another important feature that needs to be added is self-learning to mimic the brain. In one way, the ability to train the recognition model to new object classes is already in place in HMAX. However, a careful modeling of the biological learning process is needed and we need to introduce feedback paths in the emulation system to support this process.

Reconfigurable hardware serve as the ideal fabric for emulating the brain. Parallels can be drawn between logic slices in an FPGA fabric and neurons in a neural network. However, biological neural networks are far more complex. The reconfigurable nature of the hardware is critical to allow the artificial vision system to evolve, as the models evolve. Hardware reuse is the key, since FPGA resources and power are costly and hardware cannot infinitely scale to mimic biology. Further, self-adaptation and learning are essential components that can be realized efficiently in programmable hardware.

VII. CONCLUSION
Reverse engineering the brain is a grand-standing problem and significant advances have been made in this direction specially towards modeling the visual cortex of mammals. We present CoVER - Cortical Vision Emulation on Reconfigurable hardware, a framework for mapping complex computational models of cortical vision to FPGAs. The framework includes a model for developing hardware accelerators, a multi-FPGA platform architecture for scalable system design and accelerator designs for attention and recognition in the cortex. Results show stable performance and high recognition accuracy, while providing speedups over existing CPU, GPU and FPGA implementations. This is the first step towards emulating the mammalian visual pathway and is a promising step to realize artificial vision systems.

ACKNOWLEDGMENT
This work was partially supported by the Defense Advanced Research Projects Agency (government contract no. HR0011-10-C-0033) NeoVision2 program and NSF grants 0829607, 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 of the U.S. Government.

REFERENCES