The Pennsylvania State University
The Graduate School
College of Engineering

A RECONFIGURABLE ACCELERATOR FOR NEUROMORPHIC OBJECT RECOGNITION

A Thesis in
Computer Science and Engineering
by
Jagdish Sabarad

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2016

The thesis of Jagdish Sabarad was reviewed and approved* by the following:

*Signatures are on file in the Graduate School.

Abstract

A significant challenge in creating machines with artificial vision is designing systems which can process visual information as efficiently as the human brain. Recent advances in neuroscience have enabled researchers to develop computational models of auditory, visual and learning perceptions in the human brain. Among these models, the two widely accepted algorithms that model the process of attention and recognition in the mammalian visual pathway are - the Saliency based model for visual attention and HMAX model for object recognition. One of the major burdens of these biologically plausible models is their massive computational demands. Real time implementation of these biologically inspired vision algorithms, while challenging, can have a diverse and profound impact in applications like autonomous vehicle navigation, surveillance, robotics and face, text and gesture recognition. To mimic true biological systems, implementations of these algorithms must not only meet real-time performance goals, but also stringent power budgets and small form-factors.

Previous attempts to parallelize the HMAX model on multi-core processors have been unable to provide real-time performance due to limited parallelism and high computational complexity. Researchers have leveraged graphics processors due to their ease of programmability and high parallelism. However, their excessive power consumption hinders deployment in embedded or low-power systems.

The focus of this work is on the design and architecture of a reconfigurable hardware accelerator for the time consuming S2-C2 stage of the HMAX model. The accelerator leverages spatial parallelism, dedicated wide data buses with on-chip memories to provide an energy efficient solution to enable adoption into embedded systems. This work presents a systolic array-based architecture which includes a run-time reconfigurable convolution engine which can perform multiple variable-sized convolutions in parallel. An automation flow is described for this accelerator which can generate optimal hardware configurations for a given algorithmic specification and also perform run-time configuration and execution seamlessly. Experimental results on Virtex-6 FPGA platforms show 5X to 11X speedups and 14X to 33X higher performance-per-Watt over a CNS-based implementation on a Tesla GPU.

Table of Contents

List of Figures

List of Tables

Acknowledgments

Issac Newton, the great physicist and astronomer, said, "If I have seen farther, it is by standing on the shoulders of giants". And so it is with the success of this work, the success and final outcome of this thesis required a lot of guidance and assitance from many people and I am extremly fortunate to have had the opportunity to work alongside them.

First, I owe enormous debt of gratitude to my advisor Dr. Vijaykrishnan Narayanan, whose generous guidance, encouragement and support made it possible for me to work on this research topic. I express my appreciation to Dr. Jack Sampson for serving on my committee, his thoughful questions and comments were valued greatly.

I would also like to thank my fellow graduate students at Microsystems Design Lab (MDL) and the staff of Penn State Computer Science and Engineering Department for their support which made work exciting and a rewarding throughout this academic exploration.

I would like to thank my wife and my family for their unconditional love, support and patience in whatever I do. Finally, I would like to express my gratitude to all my teachers past and present who urge me to do better and stay ever-curious.

This work was partially supported by the Defense Advanced Research Projects Agency (government contract no. HR0011-10-C-0033) NeoVision2 program.

Chapter 1 Introduction

1.1 Motivation

For several decades computer scientists have been working towards creating machines capable of interpreting visual information as effectively as the brain. It is fascinating to see how humans are able to quickly scan a scene and easily identify objects of importance irrespective of the environments. Be it dull or bright, whether at an angle or far away, humans can effectively recognize a car as a car and a dog as a dog. Neuroscientists have only recently begun to understand and model how the brain handles cognitive functions making up its auditory, visual and learning perceptions. While the reasons for the superior performance of biological systems are still only partly understood, it is apparent that the architecture and the style of computation in nervous systems are fundamentally different from those applied in artificial information processing. It is envisioned that these brain-inspired, or neuromorphic, algorithms for computer vision will provide an unprecedented improvement in the way computers can analyze and interpret information. Applications of neuromorphic vision algorithms include autonomous vehicle navigation, smart camera systems, surveillance and augmented virtual reality. Real time implementations of these algorithms, while challenging, can have a diverse and profound impact in these applications.

To mimic true biological systems, hardware implementations of neuromorphic algorithms must not only meet real-time performance goals, but also stringent power budgets and small form factors. Due to limited parallelism and significant overheads of general purpose many-core processors, utilizing hardware accelerators for such algorithms can be a pragmatic choice, both in terms of performance and energy efficiency [5]. In addition, due to temporal dependencies and intricate control flow in these algorithms, they might not be a good fit for graphics processors.

Accelerators can be in the form of fixed-function hardware, programmable cores or dynamically programmable logic such as FPGAs [5]. Increasing amounts of dark silicon in next generation processors [8] have fueled interest in instruction- set extensions by adding fixed-function hardware [13] and by dynamically composing an accelerator datapath [11]. The two popular computational models of vision in the mammalian visual pathway are -

- The bottom-up Visual Attention model based on Saliency [14] [25].
- And, the multi-class recognition model based on HMAX (Hierarchical Models of recognition in the cortex) [26] [21].

The Saliency model mimics the way the brain determines fixations and object importance within an image, and forms the front end of the object recognition system. The model includes processing each image in 12 channels - 7 static channels namely - Intensity, Color (RG, BY color opponents) and Orientation (4 angles) , and 5 dynamic channels namely - Motion (4 directions) and Flicker [25]. This Saliency based attention model has been applied in object detection, gaze detection, eye movement tracking and video compression.

The brain performs exceedingly well not only at extracting salient regions within an image, but also at categorizing those objects. HMAX is a cortical model for classification which models the ventral visual pathway. It is fundamentally made up of hierarchical Simple (S) and Complex (C) layers which perform alternate template matching and max pooling to build complex features that are invariant to position and scale [26]. The HMAX model which follows the Visual Attention model forms the back end of the system and has applications in object, text and gesture recognition [22].

From a computational standpoint, this hierarchical organization of Simple (S) and Complex (C) computing layers results in a feed-forward architecture, where the amount of data, and the number of computations needed to process that data increases massively as we move up the layer hierarchy. The S2 stage in particular is the most compute intensive. This stage involves convolving each scale of the multi-scaled image pyramid with a large number of variable-sized convolution kernels. Where each convolution has a computation complexity of $O(w^2MN)$, $w \times w$ being the kernel size and $M \times N$ is the size of the input layer. This has been validated by empirical tests using the CNS-HMAX framework [20] running on a Nvidia Tesla C1060 GPU. The tests indicate that the algorithm spends almost 85% of the total execution time to compute the S2 features.

Previous attempts to parallelize the HMAX model on multi-core processors have been unable to provide real-time performance due to limited parallelism and high computational complexity. Researchers have leveraged graphics processors due to their ease of programmability and high parallelism. The CNS-HMAX framework [20] for GPU-based systems has shown to offer 97x speed-up over CPU-based implementations. However, their excessive power consumption hinders deployment in embedded or low-power systems

This work presents a reconfigurable accelerator for the S2 and C2 layers combined. The architecture, which is inspired by systolic arrays [30], is extremely parallel and offers the benefits of application-specific hardware while providing the flexibility to support several algorithmic variants. The accelerator includes -

- A run-time reconfigurable convolution engine (*CoRe16*) which can compute variable-size convolutions in parallel.
- Intelligent multi-ported memory architectures for low-latency and high bandwidth access of image data and kernel coefficients.
- An automation flow which includes a hardware optimizer for efficient partitioning of the patches on to multiple CoRe16s and a software API for run-time configuration and execution.

1.2 Related Work

There has been considerable interest in using hardware accelerators to speedup biologically inspired vision algorithms. Hardware acceleration of Convolutional Neural Networks (CNN), which is a prominent model for feature extraction, has been explored in both the FPGA [9] and the architecture domains [6]. An accelerator for Saliency-based visual attention was proposed and implemented on an FPGA [15]. Other Saliency algorithms such as AIM(Attention based on Information Maximization) algorithm have also been accelerated using FPGAs [4]. Some general computer vision based approaches for object recognition such as geometric-hashing [7] and ensemble classifier [23] have been accelerated on hardware. An FPGA implementation of a parts-based object detection system was proposed in [10] , and a prototype of a machine vision processor for V1- like algorithm was presented in [29]. While there has been a lot of interest in accelerating the HMAX-based recognition algorithms using FPGAs [17] [24], there was no prior work of on accelerating HMAX before this work was published [27]. In the computer vision context, [18] integrates face detection and face recognition to provide an end-to-end solution. However, this work as published in [16] is the first that discusses the mapping of end-to-end models of the biological visual stream on multi-FPGA platforms.

1.3 Outline

This thesis is organized as follows - Chapter-2 discusses the computation vision models of Saliency based Visual Attention and HMAX based object recognition.Chapter-3 discusses the design and architecture of the proposed S2-C2 hardware accelerator and finally, the experimental setup and results of a complete end-to-end system integrating both the attention and object recognition vision models are discussed in Chapter-4.

Chapter 2

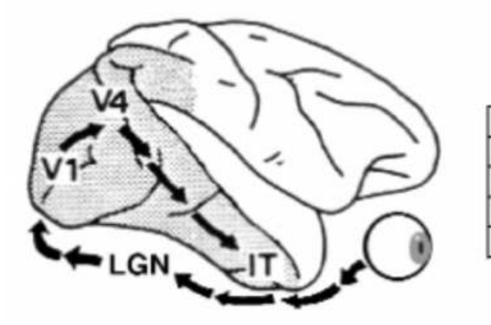
Models of Neuromorphic Vision

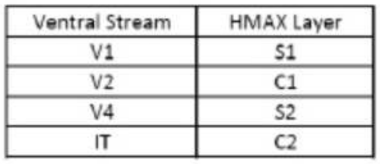
The human visual system has the ability to focus attention on to specific regions in a visual scene instead of doing an exhaustive search in a scene. This is because, in the primate brain, anatomical and functional separation between localization and identification of objects is observed: cortical areas 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) [26] [19] .

The ventral stream as shown in Figure 2.1 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) [26], using attention cues from the dorsal stream. In this chapter, two widely accepted biologically plausible models of the mammalian visual cortex are described.

2.1 Saliency Based Visual Attention System

Visual attention is the ability of the human vision system to detect salient or important sections in an image-scene, on which higher vision tasks, such as object recognition, can focus. The mammalian visual system has innate ability to identify objects in a natural scene through the selective attention mechanism. The human visual system can sequentially perceive not only a static input scene(image) but also dynamic scenes(video) with this selective attention mechanism. Real-time





Ventral Stream	HMAX Layer
V1	S1
V2	C1
V4	S2
IT	C2

Figure 2.1: Ventral stream ("what" pathway) of the human visual cortex and corresponding layers of HMAX model

analysis of such images or a motion video is critical in many surveillance applications.

A number of models of the human visual attention have been proposed. Among them, a popular model is the one proposed by L. Itti et al. [14] and this model takes into account primitive features such as intensity(I), orientation(O), color(C), motion(M) and flicker(F) which are computed independently. The work was further enhanced by Peters et al [25]. Peters et al, extended the model proposed by Itti which had only I,O,C to incorporate F and M. Intensity (I), Orientation (O) and Color(C) can work on independent images, however Flicker (F) and Motion(M) add spatio-temporal properties and are critical in a video-stream. Figure 2.2 as explained in [25] gives a detailed block diagram showing the sequence of the major computational steps. Figure 2.3 as depicted in [25] illustrates this sequence for a sample image by showing the intermediate

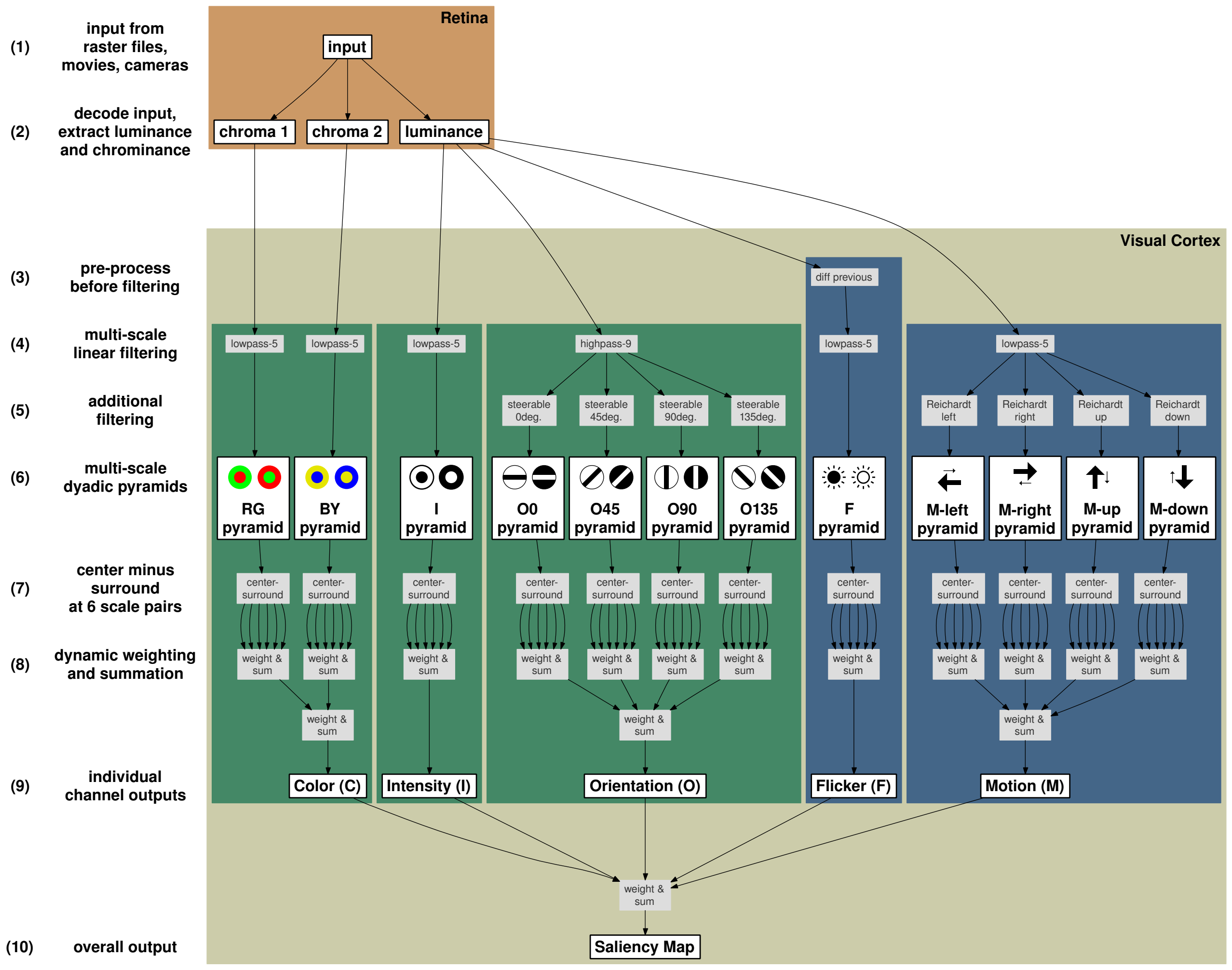


Figure 2.2: Saliency Algorithm [25]

maps generated by each computational step. This model processes the input through feature channels (color, intensity, orientation, flicker and motion) to generate feature maps at multiple scales. These help identify unusual feature values relative to their surroundings also considering the spatial aspect. An output normalization/weighting stage then follows. The conspicuity map of each feature channel is then summed and averaged to obtain the final Saliency map. As such, there are several stages of normalization and summation, first within features(say different orientations) and then across features(different feature channels),to give a single saliency map.

We refer to the Figure 2.2 for a more detailed breakdown of this process into following steps:

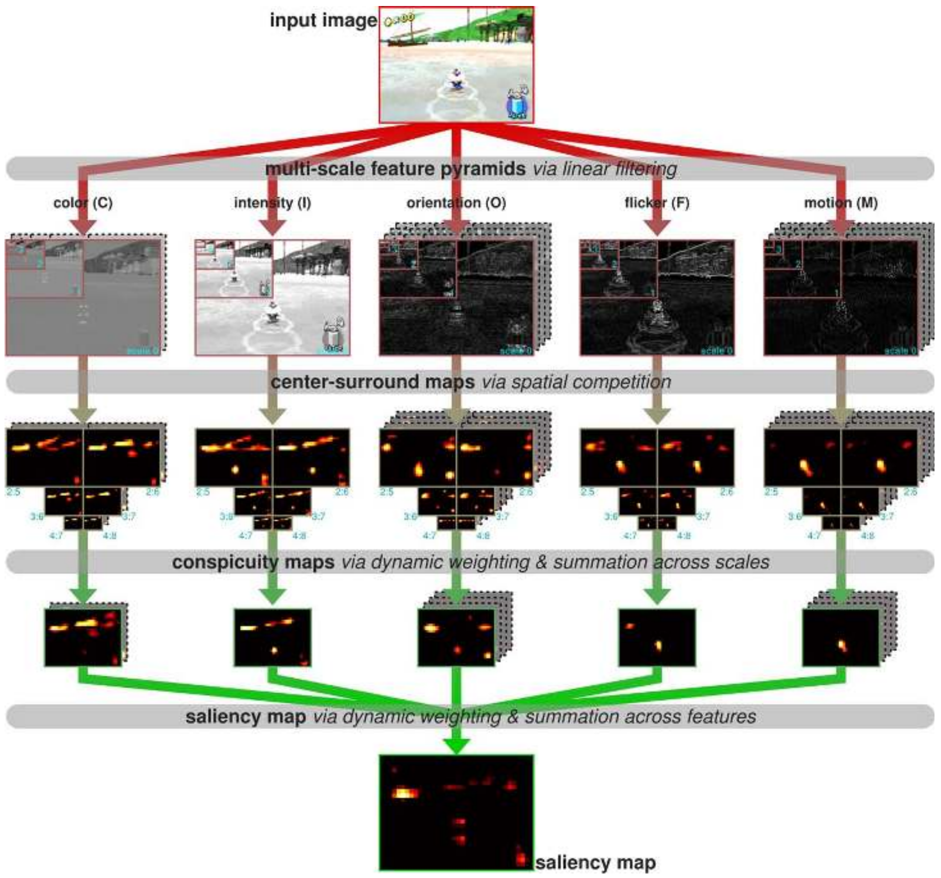

input image
multi-scale feature pyramids via linear filtering
color (C)
intensity (I)
orientation (O)
flicker (F)
motion (M)
center-surround maps via spatial competition
conspicuity maps via dynamic weighting & summation across scales
saliency map via dynamic weighting & summation across features
saliency map

Figure 2.3: Saliency with image [25]

(1) RGB input from some source (camera/frame grabber or movie files) feeds into the front end of the Visual Attention system called "Retina." This block is a preprocessing module.

(2) The retina splits the RGB image into one luminance component(Intensity) and two Intensity normalized color opponent chrominance components(Red-Green(RG) and Blue-Yellow(BY)). These retinal outputs further expand into 12 feature channels within the model "Visual Cortex" . The two chrominance components feeding two color-opponent channels ("C"; RG and BY), and the luminance component feeding the intensity, orienta- tion ("O" ; $0°, 45°, 90°, 135°$), flicker ("F") and motion ("M"; left, right, up down) channels. The flicker channel computes the absolute difference between the Intensity images of the current and previous frames.

(3) Each image stream is decomposed into a multi-scale dyadic feature pyramid by recursive application of a linear filter followed by subsampling across nine scales, from level 0 at the original scale of the image, to level 8 at a 256-fold reduction in width and height. The C, I, F and M channels efficiently compute low-pass pyramids L from their respective inputs, using a separable filter composed of 5-point low-pass kernels. The O channel computes a high-pass ("Laplacian") pyramid H [12] , essentially it is the difference between parallel full-band F and low-passed L pyramids.

(4) O and M channels use extra processing to extract the features of interest from the pyramid. In the orientation channels, an efficient steerable filter implementation is used to produce $0°, 45°, 90°, 135°$ orientation-tuned responses from the high-pass pyramid [12] In the motion channel, a simple Reichardt motion detector [31] is used to compute motion energy in the left, right, up and down directions. Details of these processing components is discussed in further chapters.

(5)There are now 12 pyramids, representing color, intensity, orientation, flicker and motion features.

(6) A center-surround operator is then applied to detect locations whose fea- ture values are different from those at surrounding locations and surrounding spatial scales. This is implemented by computing point-wise differences be- tween different pairs of pyramid scales, for combinations of three center scales (c = 2, 3, 4) and two center-surround scale differences ($s = c + \delta, \delta = 3, 4$); thus, six center-surround maps are computed for each of the 12 features, giving 72 maps at this stage.

(7)Center-surround operator followed by normalization and scaling helps enhance sparse peaks and supress the cluttered peaks. In this way, initially noisy feature maps can be reduced to sparse representations of only outlier locations which strongly stand out from their surroundings. The processed center-surround maps are then summed together to form a single output from each of the 12 individual feature channels.

(8) The overall output maps for each feature type are formed. For the I and F channels, the maps simply pass through from the previous step. Within the C, O, and M channels, each having multiple subchannels, the subchannel outputs are processed with the same dynamic iterative normalization procedure, and summed together to produce the channel output map.

(9) Finally, the channel output maps are processed once more with the iterative normalization procedure and summed together to form the final output saliency map.

2.2 Hierarchical Models of Object Recognition, HMAX

The processing in the HMAX computational model starts with a gray scale image. Next, using bicubic interpolation a image pyramid of 12 scales is created, where each scale is a factor $2^{1/4}$ smaller than the last. As shown in Fig 2.4, this image pyramid forms the input layer of the HMAX model. This image pyramid is successively feed-forward to obtain responses from the alternating higher S(simple) and C(complex) layers. The simple layers are convolutions with local filters to obtain higher order features, and the complex layers help increase the invariance by pooling units of same type from previous layers. The overall goal of the model is to reduce gray scale images to feature vectors, which can then be classified by an Support Vector Machine(SVM) classifier.

2.2.1 S1 and C1 computation layers

The S1 layer is computed by processing the image layer with 2D Gabor filters with different orientations at each possible position and scale [28] [21]. The S1 layer has twelve Gabor filter kernels representing twelve orientations, and each is of size 11x11. The output of S1 layer can be visualized as 12 pyramids (for 12 orientations), which feeds into the C1 layer [22]. The C1 layer convolves 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 to give position and scale invariance. This operation also subsamples the S1 pyramid (steps of 5 in position), resulting in C1 pyramids which are smaller in spatial extent, but have the same number of orientations as S1 pyramids; see Fig 2.4.

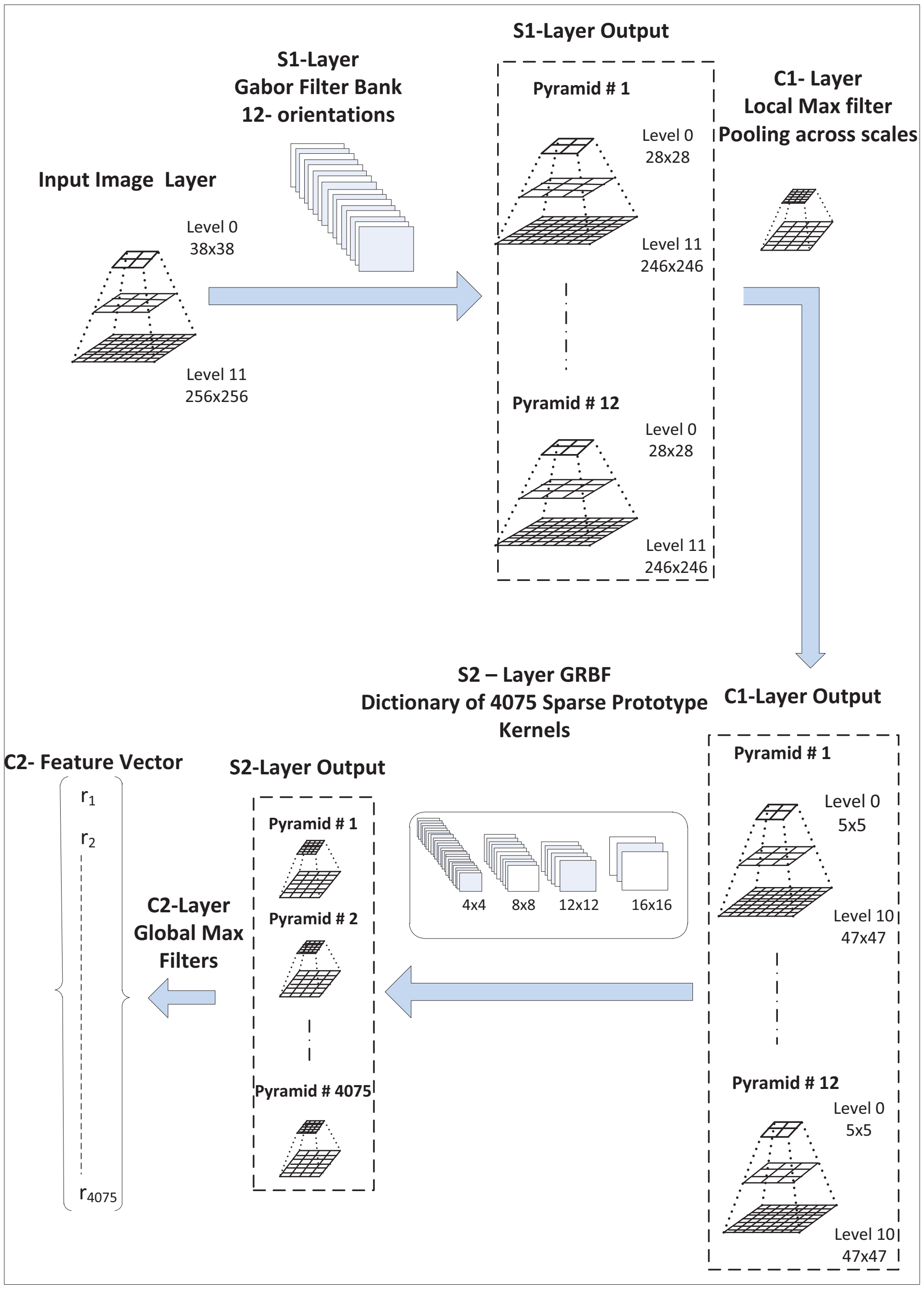


Figure 2.4: Feature vector computation using HMAX model [21]

2.2.2 S2 and C2 computation layers

The S2 layer computes the intermediate features of the HMAX model. These features are calculated by template-matching every position and scale of the C1 pyramids with a dictionary of

4075 "*sparsified*" or "*dense*" prototype kernels (aka *Patches*) [21]. The *sparse* prototype kernels are of size n x n, where $n \in \{4, 8, 12, 16\}$, and each coefficient in the kernel encodes information about it's preferred orientation of the C1 pyramid. Whereas, the *dense* patches are of size n x n x 4 (four orientations), where $n \in \{4, 8, 12, 16\}$. The response of the C1 pyramid to these prototype kernels is given by the following Gaussian Radial Basis Function (GRBF).

$$R(X_s, P_i) = exp(-\frac{\| X_s - P_i \|^2}{2\sigma^2\alpha}) \tag{2.1}$$

where $P_{i=\{1,..4075\}}$, represents the number of prototype kernels in the dictionary, and $X_{s=\{1,..11\}}$ represents the number of scales in the C1 pyramid. The standard deviation σ is set to 1, and the parameter $\alpha = (n/4)^2$ is the normalizing factor for different kernel size n. Depending on the kind of prototype kernels used, the resulting S2 output can be 4075 pyramids (sparse) or 4075 x 4 pyramids (dense).

The C2 is a global max across scales(done for each of the 4075 pyramids) to obtain a single vector, each value of which is response to a given prototype patch. The C2 response can be represented as:

$$max\{R(X_s, P_i)\} \tag{2.2}$$

2.2.2.1 Hardware oriented S2 and C2 reformulation

Alternatively, we can re-formulate the S2 and C2 computations as:

$$R(X_s, P_i) = (\frac{\| X_s - P_i \|^2}{2\sigma^2\alpha}) \tag{2.3}$$

$$exp\{min\{R(X_s, P_i)\}\} \tag{2.4}$$

This reformulation allows us to move the exponential operation to after the C2 computation. As a result, the C2 operation is global min instead of global max and the exponential needs to be computed only on the min value rather than the entire array - i.e we need to perform the exponential operation directly on the C2 vector once. Similarly, we also factor out the constant denominator term (for a given kernel size) in the equation. This helps save computation resources (multipliers), which in turn can be used to speed up the convolution operations.

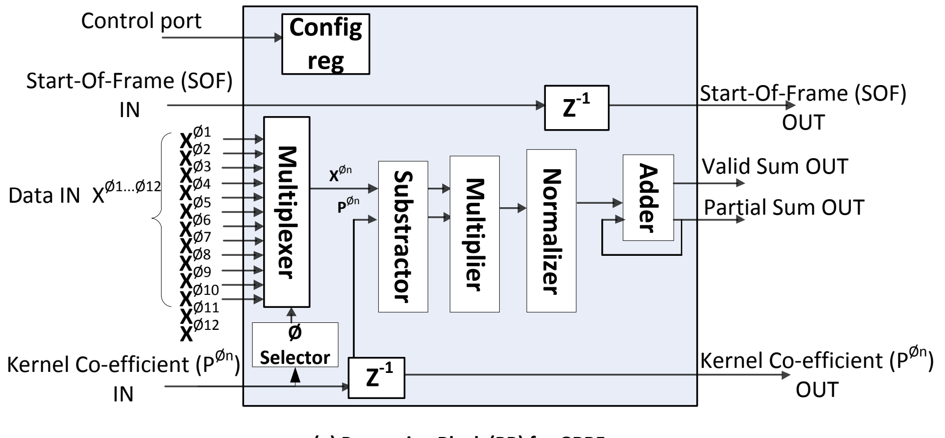
Chapter 3

S2-C2 Hardware Accelerator Architecture

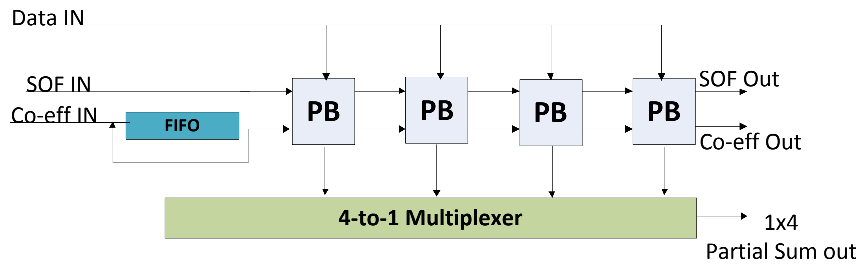
This chapter describes the architecture of the proposed hardware accelerator.

3.1 Reconfigurable Convolution engine (*CoRe16*)

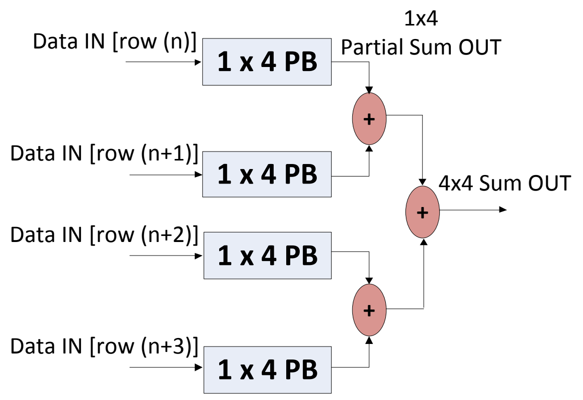
The proposed hardware accelerator for S2-stage is inspired by systolic arrays. Systolic arrays, due to their spatial and temporal regularity are highly modular and this makes it easy to parameterize the hardware design to scale to the computational needs of the application. The basic processing element of this accelerator module is the GRBF Processing-Block(PB) shown in Fig 3.1(a). Depending on the size of the smallest kernel in the S2 prototype dictionary, this PB can be composed into two dimensional arrays called Composed-Processing-Blocks(CPB), shown in Fig 3.1(c). And, depending on the size of the largest kernel in the S2 prototype dictionary, the CPB can be composed into a two-dimensional grid to form the Reconfigurable convolution engine, also referred to as CoRe16. Using CPB of size 4 x 4, 16 such CPBs can be tiled in a 2-D grid to form a 16x16 convolution engine shown in Fig 3.1(a). This CoRe16 is highly-flexible and can reconfigured at run-time to process kernels of all sizes in the S2-dictionary. The next few sections explains the hardware details of this accelerator module.



(a) Processing Block (PB) for GRBF



(b) 1x4 Processing Blocks



(c) 4x4 Processing Blocks

Figure 3.1: (a) Processing Block (PB) can be composed into 1D or 2D arrays to form (b)(c) Composed Processing Blocks (CPB).

3.1.1 Processing Block(PB)

The Processing block forms the lowest hardware primitive in this design hierarchy, and has been tailored for computing GRBF features in S2 layer. The simplified block diagram of this block is shown in Fig 3.1(a). The computational units inside the block are pipelined and have simple hand shaking signals between them. In addition, all these units are parameterized to handle the algorithm-specific parameters. The configuration information during run-time reconfiguration is passed to this block by using the control port and stored in internal registers. During operation, the inputs to this block are the C1-feature vector ($X^{\phi^1}, ...X^{\phi^{12}}$), which are features from same position of different C1 pyramids. The other input is the S2-prototype kernel coefficient (P^{ϕ^n}). The multiplexer unit at the input selects the input C1-feature depending on the orientation (ϕ^n) preference encoded in the kernel coefficient. The Subtractor followed by the multiplier in the pipeline computes the partial-product of the Euclidean-distance between the C1 feature and the kernel coefficient. This partial product is accumulated by the adder. The number of partial products accumulated depends on the the size of the S2 kernel being processed. The kernel size information is passed to the PB through the control port during the run-time reconfiguration phase. The output of PB is the partial sum along with a valid signal that indicates the validity of the partial sum. The second output of PB is the delayed (1 cycle) value of the kernel coefficient which is passed to the neighboring PB. The Start-Of-Frame(SOF) signals are frame delimiter signals to indicate the start and end of input frame.

3.1.2 Composed Processing Blocks(CPB)

Depending on application requirements, the Processing Block (PB) for GRBF feature computation can be composed into 2D arrays of various sizes. These 2D Composed Processing Blocks (CPB) form the next level of hardware primitives in this design hierarchy. The simple signaling interface of the PB makes it easy to connect them as 1D arrays of desired size(Fig 3.1(b)). The valid out signals of the PB in the 1D array are used as the select bits for the output multiplexer. The FIFO acts as a local buffer for the kernel coefficients. These 1D arrays can then be replicated to form the 2D array(Fig 3.1(c)). The adder tree at the output are used to accumulate the partial sums of the 1D arrays.

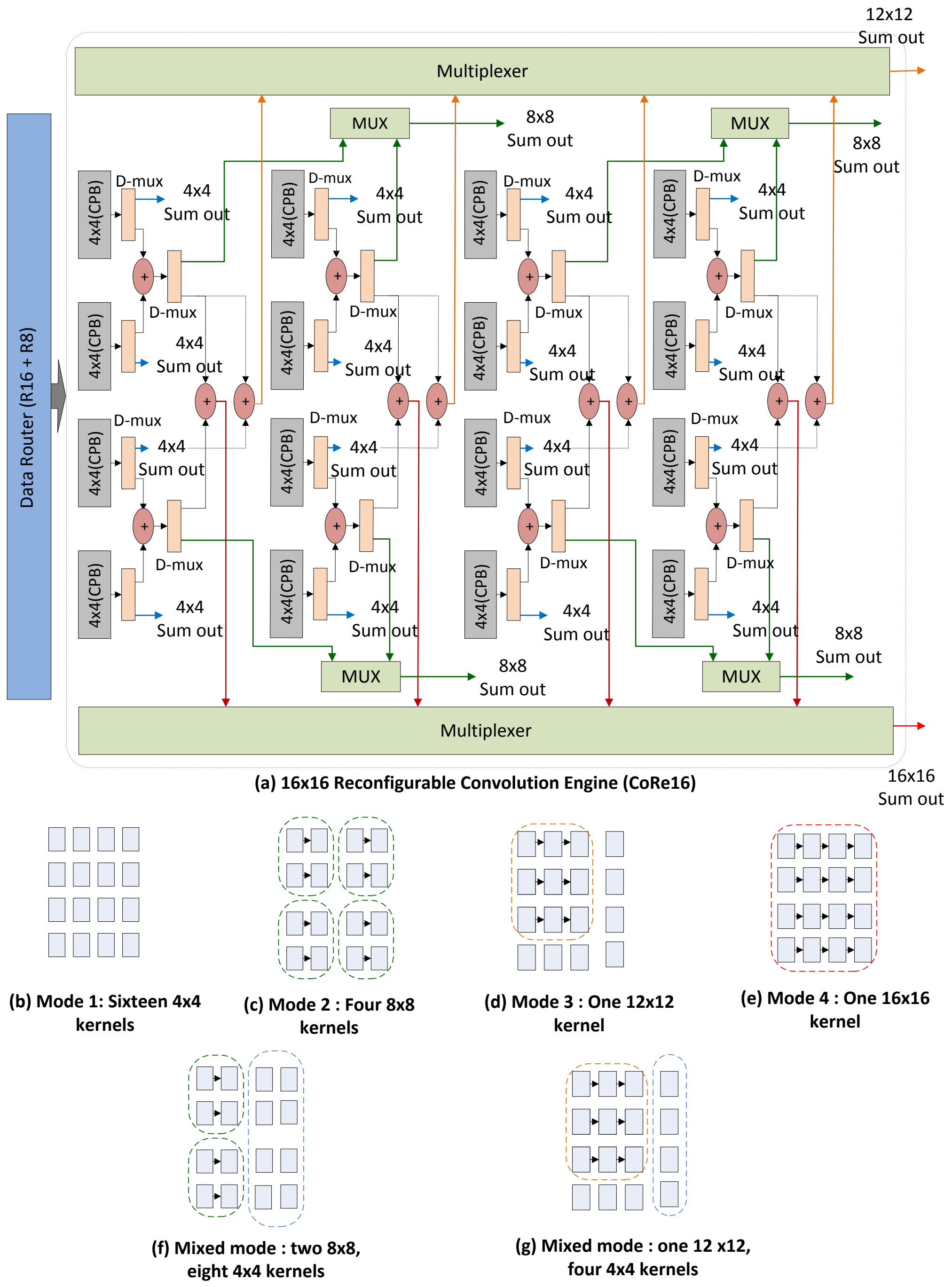

12x12
Sum out
Multiplexer
MUX
8x8
Sum out
Data Router (R16 + R8)
4x4(CPB)
D-mux
4x4
Sum out
MUX
Multiplexer
16x16
Sum out
(a) 16x16 Reconfigurable Convolution Engine (CoRe16)
(b) Mode 1: Sixteen 4x4 kernels
(c) Mode 2 : Four 8x8 kernels
(d) Mode 3 : One 12x12 kernel
(e) Mode 4 : One 16x16 kernel
(f) Mixed mode : two 8x8, eight 4x4 kernels
(g) Mixed mode : one 12 x12, four 4x4 kernels

Figure 3.2: (a) CoRe16 module (b)-(g) Operation modes

3.1.3 *CoRe16*

The Reconfigurable Convolution engine (CoRe16) can be composed by tiling the 2D CPBs as a 2D grid. This module forms the third level of hardware primitive in this design hierarchy. In this configuration, each CPB in the 2D grid is an independent GRBF computation engine. The CoRe16 which represents a reconfigurable 16x16 convolution as shown in Fig 3.2(a) is composed using sixteen CPB of size 4x4. A single instance of the CoRe16 can be reconfigured at run-time to operate in the following pre-defined modes:

- *Mode 1*: process up to *sixteen* 4x4 S2 kernels in parallel (Fig 3.2(b)).
- *Mode 2*: process up to *four* 8x8 kernels in parallel (Fig 3.2(c)).
- *Mode 3*: process *one* 12x12 kernel (Fig 3.2(d)).
- *Mode 4*: process *one* 16x16 kernel (Fig 3.2(e)).
- *Mixed-modes*: process [*one* 12x12, *four* 4x4], [*two* 8x8, *eight* 4x4] and [*three* 8x8, *four* 4x4], each case in parallel (Fig 3.2(f-g)).

The mixed-mode operation helps in optimal usage of hardware resources for an arbitrary distribution of the kernels in the dictionary. For example, for twenty 4x4, and seven 8x8 kernels in the dictionary, we could choose to sequence the operations as *{mode1, mode1, mode2, mode2}* or, we could sequence operations as *{mode1, mode2, mixed-mode (three 8x8, four 4x4)}*. In this example, the later sequence obviously has better resource utilization. Further, acceleration can be gained by having multiple of these CoRe16 modules operating in parallel. These multiple instances of the CoRe16 forms the top level hierarchy of our design, see Fig 3.3. In this configuration, a CoRe16 in the group can be independently operating in any of the above pre-defined modes. The data and the kernel co-efficient to this module are fed by the Multi-port memory and Patch Buffer modules. The details of these two modules are discussed next.

3.2 S2-C2 pipeline

3.2.1 Multi-port Memory and Routers

The CoRe16 operates on the C1 image pyramid, one level at a time. Since S2 computation needs to be repeated over a large number of kernels or patches for each C1 image (say 4075), there are repeated accesses for the same C1 data. Thus by storing the C1 image in a on-chip memory data reuse is utilized.

The (*CoRe16*) module needs access to up to 16 rows and 16 columns of the C1 image at a time depending on the mode it is operating. Example: If in Mode 1 (for 4x4 kernels), all the CPBs require rows i to i+3 and columns j to j+3, but in Mode 4 (for 16x16 kernel) each CPB requires a unique 4x4 block of the C1 image. In order to enable each CoRe16 to operate these various modes and utilize the parallelism within the CoRe16, a multi-port memory (MPM) is proposed along-with data routers (*R16* and *R8*) to feed the CoRe16 modules with appropriate image data.

The MPM utilizes bank-interleaving along the rows of the image to enable multiple reads per cycle. The number of banks is set to 16 to allow 16 reads per cycle from 16 contiguous rows, since 16 is the maximum size of patches. The bank-interleaving along the rows also takes care of the sliding window (16x16) required while processing the image.

Further, data routers R16 and R8 route the appropriate 4x4 pixels to each CPB in the CoRe16 module. The routers can be reconfigured at run-time to modify the selection of pixels based on the operating mode. The MPM can be reconfigured at run-time to change the C1 image size (which is different for each scale of the pyramid).

The C1 image is read repeatedly from the MPM to complete the processing of all the patches. When multiple CoRe16 modules are enabled, the output bus from MPM is broadcast to all CoRe16s with each CoRe16 processing a different set of patches on the image. The above process is further repeated for all scales of the pyramid to complete the S2-C2 computation for one image.

3.2.2 Patch Buffer

As mentioned before, the CoRe16 module operates in different modes and can process multiple patches and patches of different sizes. This requires an efficient mechanism to initialize the CoRe16 with patch coefficients. Since the patches are reused for every pyramid level for every image, all the patches are stored in on-chip BRAMs. If memory constraints are severe, then on-chip BRAMs can be used to cache a subset of patches and then remaining patches must be loaded from external memory.

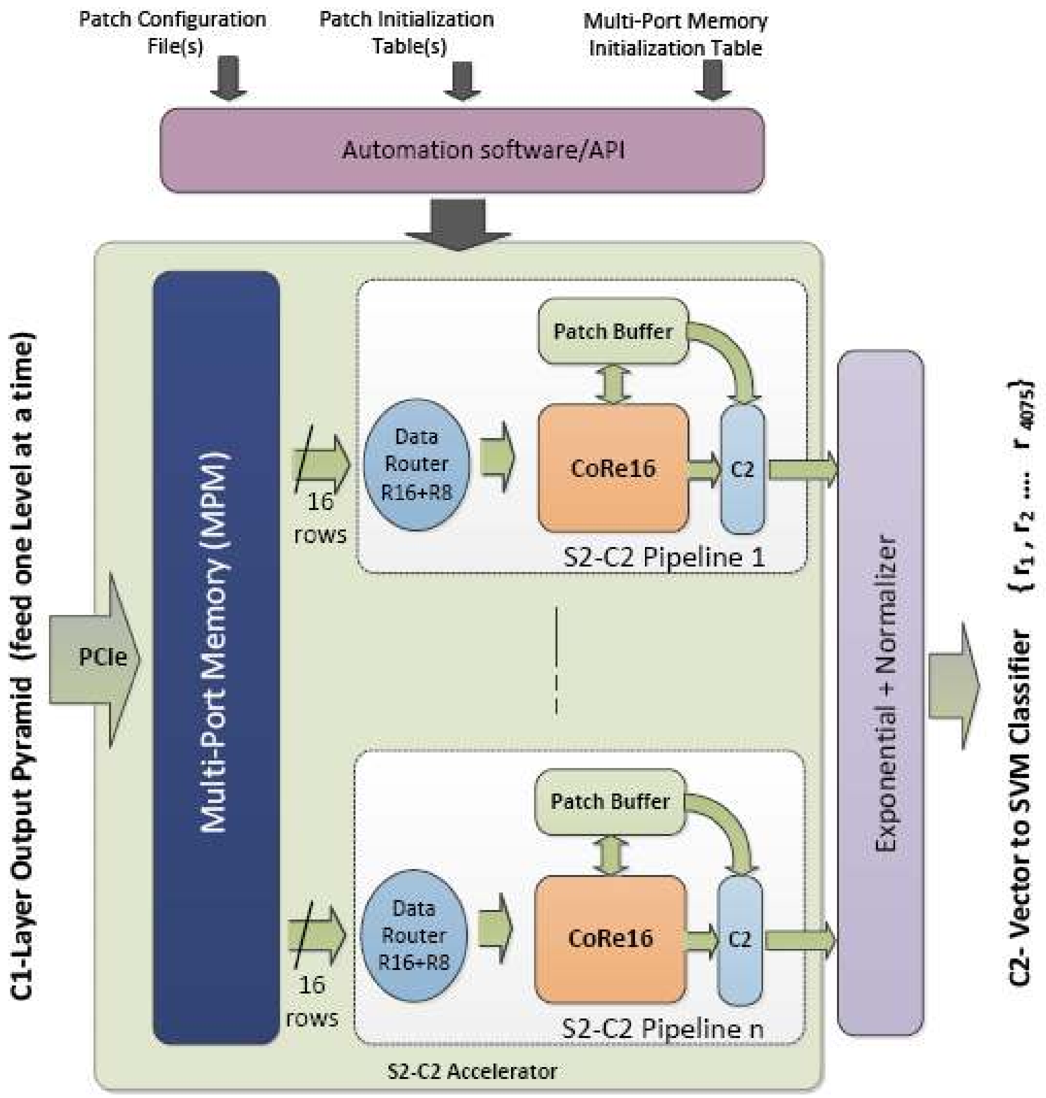


Figure 3.3: S2-C2 Pipeline

The patch buffer is a critical component which handles loading of appropriate patches into the CoRe16 module and control information to the C2 stage to handle data accumulation. To minimize load time, 16 coefficients are loaded per cycle which makes it 16 cycles to completely

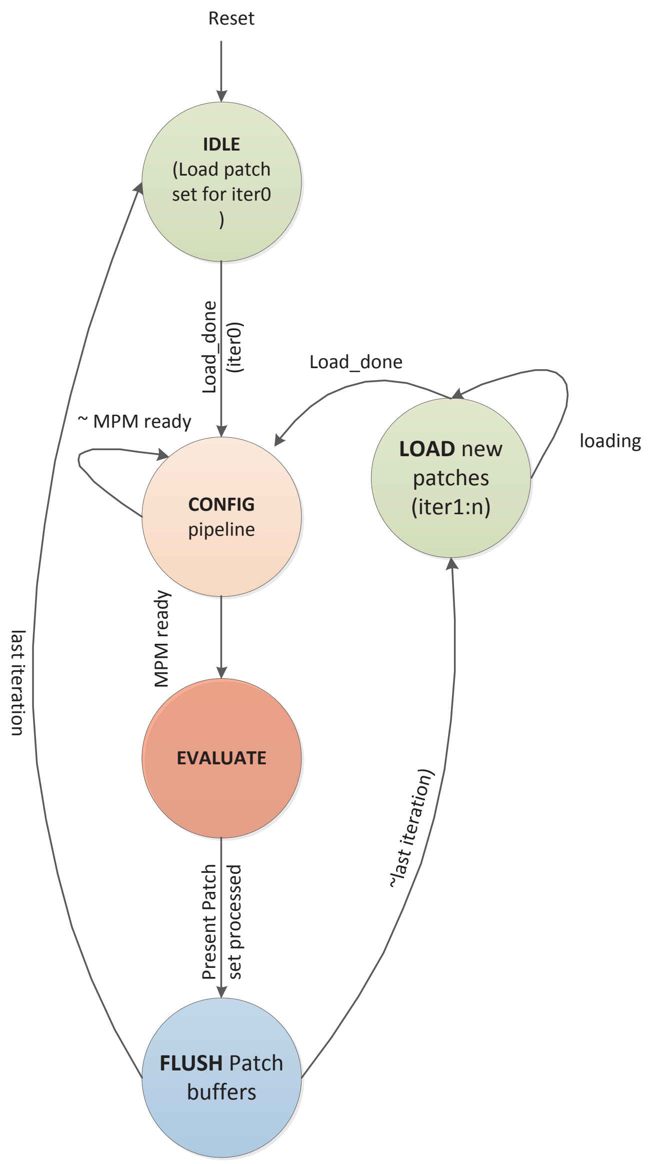


Figure 3.4: Patch Buffer Initialization

initialize a CoRe16 module (one iteration). This is enabled by a combination of bank-interleaving and a wide memory lines. Example: with 4 banks and a line size of 4, 16 coefficients can be read per cycle. This load operation must be repeated before each iteration to complete the processing for all patches for a C1 image. This process is then repeated over multiple levels.

The patch buffer also sends out a header to the CoRe16 and C2 modules before every iteration, which includes information on the mode of operation, number of valid patches etc which are used to configure the CoRe16 module and the data routers for that iteration.

The patch buffer needs to be initialized with the patches at run-time before computation begins. The patches are loaded from a patch configuration file at run-time in an automated way as shown in 3.3 and 3.4. Thus any change in the number of patches, patch distribution or the actual patch coefficients can be accommodated without having to modify the hardware.

3.2.3 C2 - Data Accumulation Unit

The C2 block is a global max (global min in this implementation) across scales. The input to C2 block is the S2 response to 4075 patches for each level of the pyramid. This can be actually visualized as 4075 pyramids as depicted earlier. The min operation helps perform the exponential operation external to the S2-C2 block. This mainly provides 2 benefits. First we can reuse the same exponential block across pipelines and second, we can also perform the exponential operation in software if required since we only need to perform this operation on the C2 vector(4075 values) instead of a per pixel basis.

The C2 block operates on the S2 responses of all the patches and computes the Global Minimum across all scales of the S2 pyramid(Fig 3.3). The operation is repeated iteratively for each patch across all C1 scales to obtain a C2 vector. A block level description of this hardware module is shown in Fig 3.5.

The global min is unique for each patch in the patch dictionary. The challenge is to keep track as to which of the patches are being processed during a given iteration and update the value accordingly. A control header generated by the patch buffer is used by the C2 module for this purpose. The header includes fields for the current pyramid level being processed, the current iteration count and the operating mode of the CoRe16. An on-chip RAM is used to buffer the

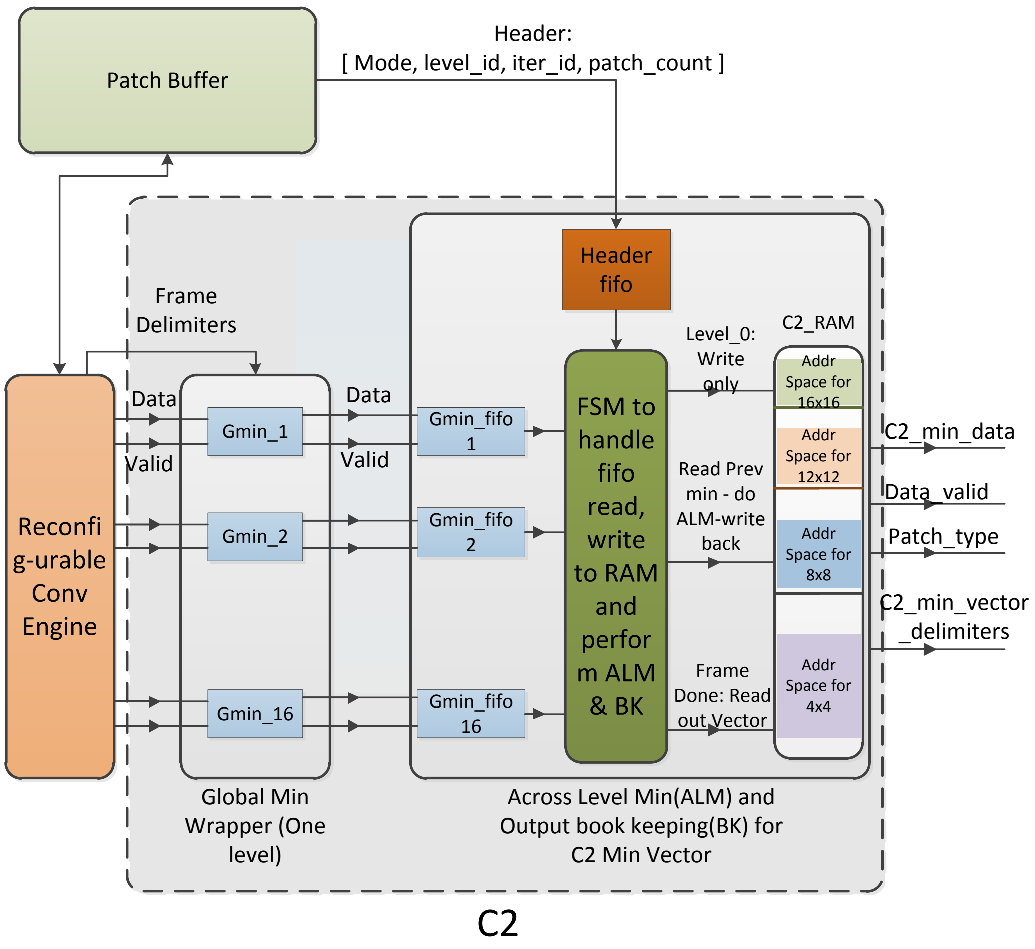

Patch Buffer
Header:
[Mode, level_id, iter_id, patch_count]
Header fifo
Frame Delimiters
C2_RAM
Level_0: Write only
Reconfi g-urable Conv Engine
Data
Valid
Gmin_1
Gmin_2
Gmin_16
Data
Valid
Gmin_fifo 1
Gmin_fifo 2
Gmin_fifo 16
FSM to handle fifo read, write to RAM and perform ALM & BK
Read Prev min - do ALM-write back
Frame Done: Read out Vector
Addr Space for 16x16
Addr Space for 12x12
Addr Space for 8x8
Addr Space for 4x4
C2_min_data
Data_valid
Patch_type
C2_min_vector _delimiters
Global Min Wrapper (One level)
Across Level Min(ALM) and Output book keeping(BK) for C2 Min Vector
C2

Figure 3.5: C2 Data Accumulation unit

min values for each patch and a particular value is updated by indexing into the RAM based on the fields in the header. Once all the pyramid levels are processed, the final C2 vector is read out of the RAM, exponential computed by a library core and then passed to a classifier software implemented on host computer for object classification.

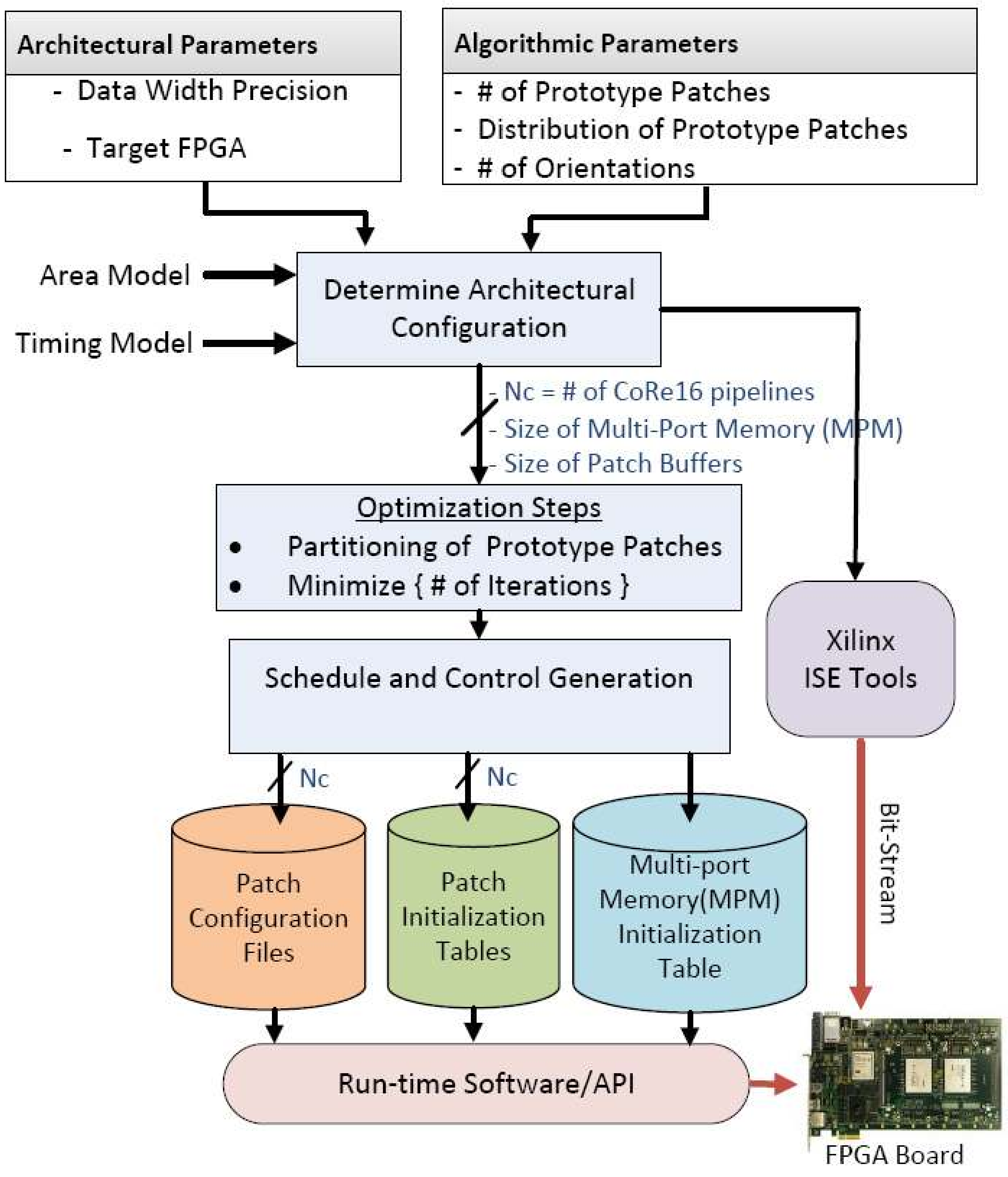
3.3 Automation Flow and Scheduling

Parallelism, reconfigurability and scalability are key factors that determine the efficiency of a hardware accelerator. This hardware accelerator has a high degree of coarse-grain parallelism - multiple CoRe16 pipelines (Fig 3.3) and fine-grain parallelism - within a CoRe16 all multiplications are in parallel (Fig 3.2). It is completely parameterized and supports both compile-time and run-time reconfiguration. The architecture is independent of image size or number of patches or number of pyramid levels or resources available and can scale to any desired algorithmic specification or power/area constraints.

In order to leverage these features of the accelerator, an automation flow (Fig 3.6) was developed. This automted flow can seamlessly generate optimal hardware configuration for a given algorithmic specification and perform run-time configuration and execution. The algorithmic and architectural parameters and performance requirements can be specified as a parameter file. The automation flow performs resource estimation based on area models for each hardware component and performance estimation based on timing models and generates the optimal hardware configuration under the given constraints. This is followed by an optimization step which attempts to maximize utilization of hardware resources by partitioning the computation effectively on to multiple pipelines. For a given hardware configuration and a given patch distribution (# of patches of 4x4, 8x8, 12x12, 16x16 sizes), the optimization step partitions the computations on to the pipelines such that the total execution time is minimized. The scheduling step includes generation of the initialization tables for the MPM and the patch buffer based on the above partitioning and generating the patch configuration file for each pipeline. The initialization tables include all parameters such as image sizes and number of iterations for each C1 pyramid level, which are required by the hardware to iteratively run the CoRe16 pipelines to compute the C2 vector for a given C1 pyramid.

The software/API handles write/read of data to/from the hardware. Based on the number of pipelines specified by the optimization step, the API determines the number of FPGAs required.

The C1 pyramid is then iteratively loaded one level at a time into each FPGA and the resulting C2 vector is fed into an SVM classifier implemented on the host computer.



Architectural Parameters
- Data Width Precision
- Target FPGA
Algorithmic Parameters
- # of Prototype Patches
- Distribution of Prototype Patches
- # of Orientations
Area Model
Timing Model
Determine Architectural Configuration
- Nc = # of CoRe16 pipelines
- Size of Multi-Port Memory (MPM)
- Size of Patch Buffers
Optimization Steps
• Partitioning of Prototype Patches
• Minimize { # of Iterations }
Schedule and Control Generation
Xilinx ISE Tools
Nc
Nc
Patch Configuration Files
Patch Initialization Tables
Multi-port Memory(MPM) Initialization Table
Bit-Stream
Run-time Software/API
FPGA Board

Figure 3.6: Automation and Scheduling

Chapter 4

Experimental Setup and Results

The Dinigroup DNDualV6-PCIe4 board [2] with two Xilinx Virtex6 SX475T FPGAs was used as a test platform. The Verilog implementation of this accelerator is in fixed-point and Xilinx ISE tools were used to generate the hardware. Direct PCIe connection between one of the compute FPGAs and the host-machine is established via the DNSEAM-PCIe [1]. A set of software APIs running on the host-machine was developed to configure the accelerators and for DMA read/write and polling registers. The software also handles scheduling by partitioning each incoming 2352 x 1724 video frame captured using a high resolution Basler camera into smaller chips of size 256 x 256. A double buffer architecture is followed on the multi-port memory to overlap I/O with computation.

In Table 4.1, we compare the performance of S2-C2 implemented using the CNS framework on a GPU with our FPGA accelerator. 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. The same set of patches were utilized on both the GPU and the FPGA platforms to get realistic comparison. The GPU numbers were averaged over 100 runs to remove any overheads. It should be noted that the CNS framework on GPU itself provides close to 97X speedups over a CPU implementation [20].

The test cases for comparison are -

- case 1k4x4 - 1024 patches, all of size 4x4.

- case 1k8x8 - 1024 patches, all of size 8x8.
- case 1k12x12 - 1024 patches, all of size 12x12.
- case 1k16x16 - 1024 patches, all of size 16x16.
- case 4kmixed - 4075 patches with 1521 patches of size 4x4, 1145 of 8x8, 802 of 12x12 and 607 of 16x16.

The operating frequency of the FPGA is 100 MHz. The performance metric is the frame rate for the entire S2-C2 processing stage which decides the recognition throughput. Table 4.1 shows that the implemented accelerator running at 100 MHz on a single Virtex-6 FPGA provides 5X to 11X speedups over the CNS implementation on GPU for cases ranging from 1k4x4 to 4kmixed when compared to the GPU. The performance numbers for the FPGA are for 2 CoRe16 pipelines and 12 orientations.

Table 4.1. Comparison of S2-C2 (sparsified) for 256 x 256 images

Test Case	Tesla C1060 GPU		Virtex-6 SX475T FPGA	
	FPS	FPS per Watt	FPS	FPS per Watt
case 1k4x4	37.03	0.197	432.52	6.654
case 1k8x8	13.69	0.0729	131.13	2.017
case 1k12x12	7.69	0.0409	39.78	0.612
case 1k16x16	5.26	0.028	48.42	0.7449
case 4kmixed	2.88	0.0153	22.92	0.3526

Also measured is an approximate performance-per-Watt comparison using the peak power of the GPU and FPGA devices. The peak power of the Tesla C1060 GPU is taken as the Thermal Design Power (TDP), which is 187.8 Watt [3] and the peak power of the Virtex6 SX475T on the DN-DualV6-PCIe board [2] is 65 Watt. As shown in Table 4.1, this accelerator running on the FPGA provides 14X to 33X higher performance-per-Watt for cases ranging from 1k4x4 to 4kmixed when compared to the GPU. This shows that the proposed FPGA accelerator is an energy efficient solution for accelerating HMAX.

4.1 Performance vs Accuracy trade-off

The object recognition accuracy of the algorithm has a strong dependence on the number of patches and the patch distribution [22], i.e the number of patches of 4x4, 8x8, 12x12 and 16x16. It is obvious that the 16x16 and 12x12 (BIG) kernels involve much more computations than 4x4 and 8x8 (SMALL) kernels. Higher the number of BIG patches, better the recognition accuracy, but lower the performance. Lower the number of BIG patches, lower the recognition accuracy, but higher the performance. A trade-off analysis was performed by comparing the performance of the accelerator for different patch distributions, while the total number of patches is fixed. Figure 4.1 shows a plot of the throughput as a function of the total number of iterations, where total iterations $= [(\#(4 \times 4)/16) + (\#(8 \times 8)/4) + \#(12 \times 12) + \#(16 \times 16)]$. In the plot, as we move from left to right the number of SMALL patches are increasing and the number of BIG patches are decreasing. It shows that the performance scales almost linearly as the number of iterations. The same behavior is observed when multiple pipelines are present, which means that the optimizer partitions the computations almost equally to all pipelines. The plot also shows that the performance of the accelerator scales almost linearly with increase in number of pipelines. These results are for 4 orientations where we can fit 3 pipelines on a single FPGA.

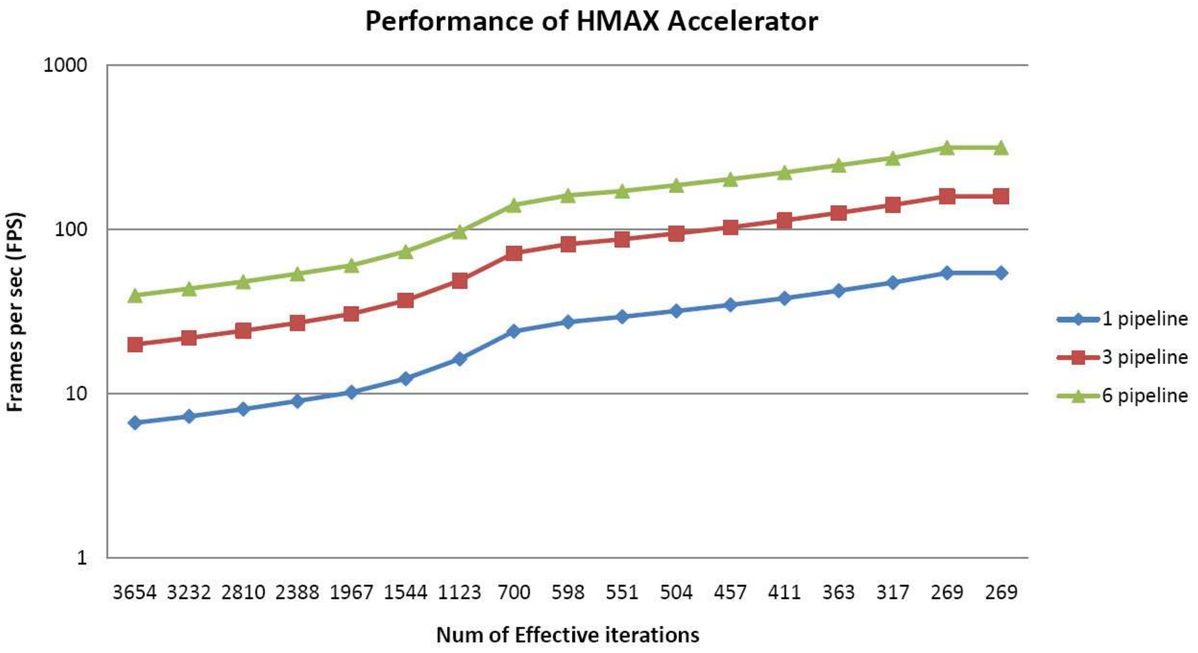


Figure 4.1: Performance scaling with degree of parallelism

Another parameter is the number of orientations, a higher value of which improves the resolution of the sparsification and hence improves recognition accuracy [21]. Table 4.2 provides the FPGA resource utilization for our S2-C2 accelerator for 4 and 12 orientations. For higher number of orientations, more resources are used by the on-chip memories and routing logic which reduces

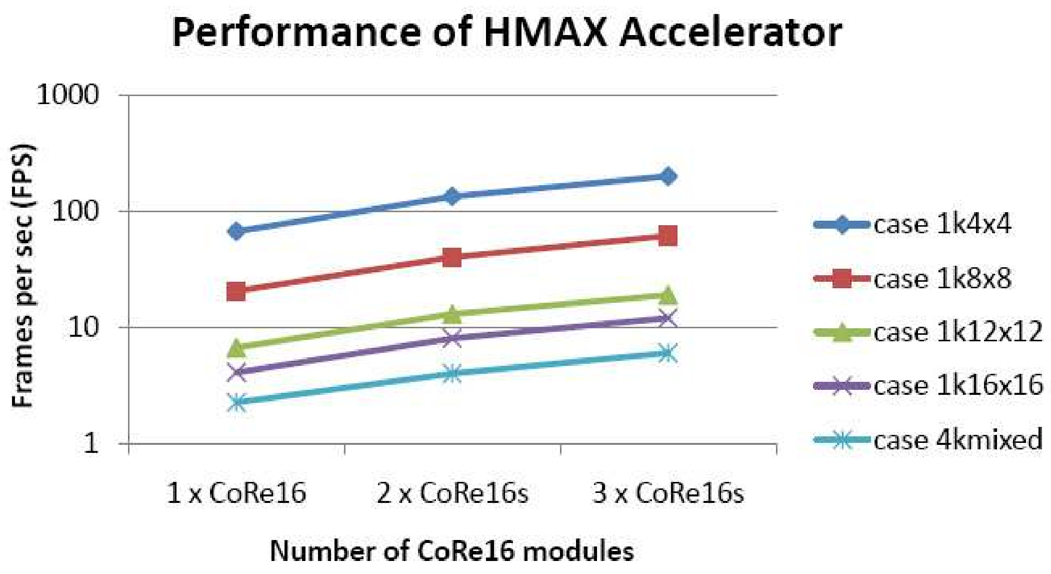


Figure 4.2: Performance scaling with number of CoRe16s

the number of pipelines that can be fit on a single FPGA, hence reducing performance.Further, the accuracy also depends on the precision of the hardware implementation. Higher the bit-width, better the accuracy. However, given a resource/area constraint, if the bit-width of the pipeline is high, resource consumption is high and the lesser number of parallel pipelines and lower the performance. For this implementation, a 24-bit width was utilized to guarantee the best classification accuracy.

Table 4.2. Resource Utilization on Virtex-6 SX475

Architecture	Slice Regs	Slice LUT's	BRAM's	DSP48
3 Pipelines, 4xθ	253313(42%)	209942(70%)	500(46%)	1536(75%)
2 Pipelines, 12xθ	224730(37%)	212448(71%)	558(52%)	1024(50%)

Figure 4.2 shows that the performance of the FPGA accelerator scales almost linearly with increase in number of pipelines. This is because, each CoRe16 can compute the S2 kernel independently on different set of patches, while reusing the same C1 data from the multi-port memory as shown in Fig 3.3. This improves the performance of the application while introducing minimal overhead. Further performance gains can be obtained by mapping CoRe16s onto multiple FPGAs, and partioning the S2-patches across pipelines on multiple FPGAs.

Table 4.3. Performance Comparison in FPS for HMAX Implementation

GPU [20]	FPGA [17]	FPGA [27]
Tesla C1060	4 x Virtex5 SX240T	Virtex6 SX475T
2.88	3.61	22.92

Table 4.3 provides a performance comparison between this HMAX accelerator [27] implemented on a single Virtex6 SX475T FPGA with existing GPU [20] and FPGA implementations [17]. As seen this implementation obtains speedups of 15X over the GPU [20], 12.5X over the Virtex5-based multi-FPGA platform [17].

4.2 Full-System Implementation and Results

As implemented in another work by Srinidhi et al [16], this section discusses the integration of the proposed HMAX S2C2 architecture into an end-to-end attention recognition system. The accuracy of the object recognition system are also discussed.

Biologically Inspired vision algorithms are both computation and data intensive, so acceleration on FPGAs requires a large number of logic slices, RAMs and DSPs. In order to decouple

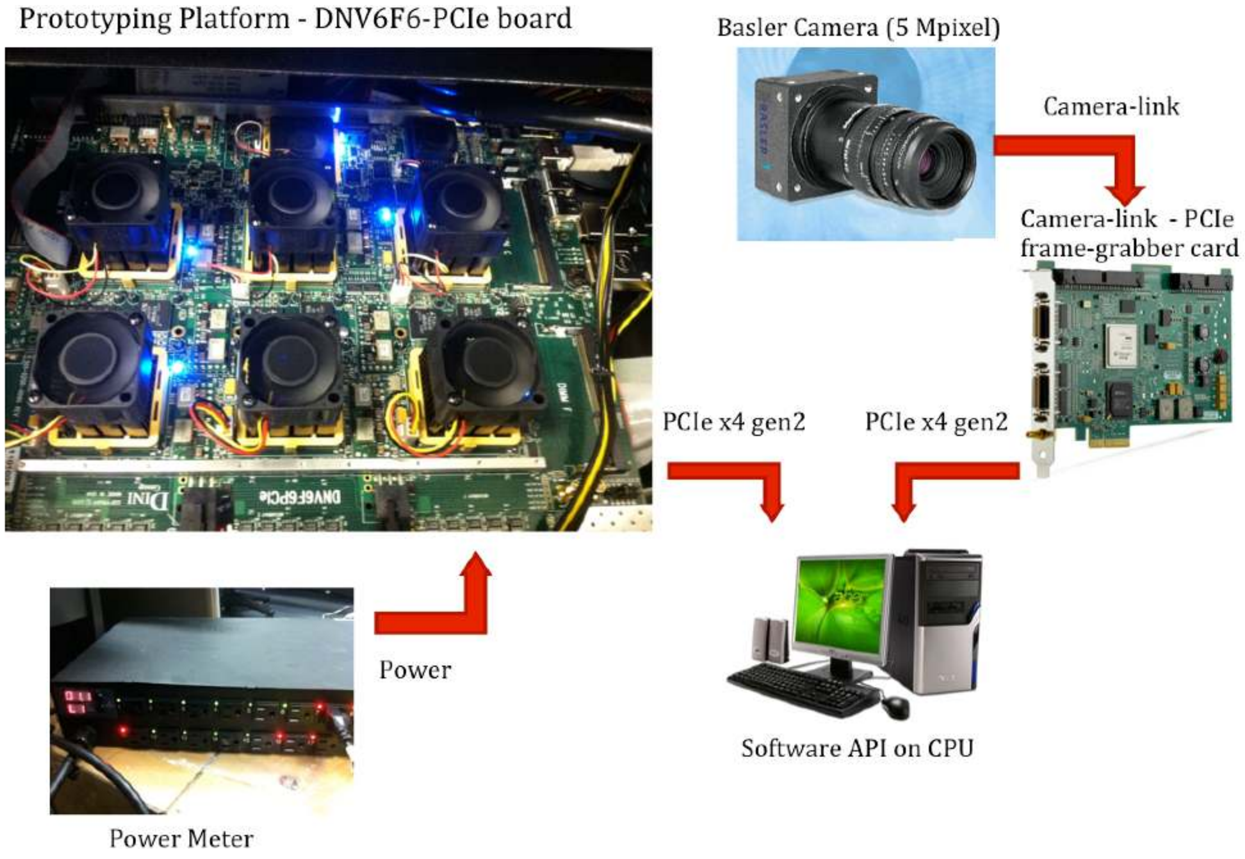


Figure 4.3: Multi-FPGA System Setup

accelerator design, implementation and evaluation of different bio-inspired vision algorithms, one option is to dedicate independent FPGA for each complex algorithm. Figure 4.3 illustrates the complete end-to-end attention-recognition system that was implemented.

4.2.1 Multi-FPGA Interconnection Network

The architecture of the proposed HMAX accelerator was intergrated into an end-to-end attention recognition system and implemented on a multi-FPGA platform. As shown in Figure 4.4 the attention (Saliency) and recognition (HMAX) accelerators are configured on independent FPGAs . The Dinigroup DNV6F6-PCIe board [2] was chosen as the multi-FPGA platform to implement this attention-recognition system. The DNV6F6-PCIe board is populated with up to six Virtex6-

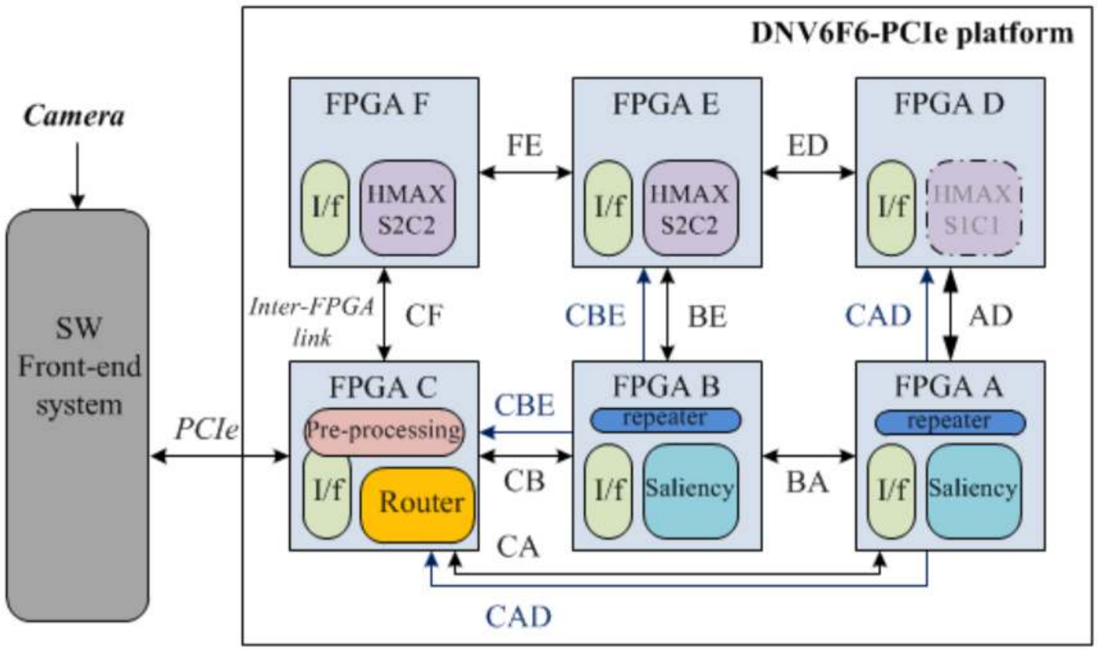


Figure 4.4: Saliency and HMAX configuration on Multi-FPGA Platform

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, a high- speed connector DNSEAM-PCIe [1] was used to obtain direct PCIe connection to one of the compute FPGAs via the GTX expansion headers. Currently, there are 4-lane GEN2 PCIe connection from a host PC to FPGA C on the DNV6F6 board. The interconnection network as depicted in Figure 4.4 uses inter-FPGA LVDS links for communicating between devices. Asynchronous FIFOs are used to synchronize data between clock domains. This multi-FPGA setup includes a simple router to help route data between the devices. This router is mapped on FPGA C, and the host computer communicates to the other FPGAs on the board through this router interface. Dedicated or

Shared one-hop or multi-hop links are provided between FPGA C and the other FPGAs on board. The router is optimized to operate 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 computer sets the fields in the frame header and initiates the data transfer and is abstracted from the complexities of the network.

4.2.1.1 Clocking

One challenge in inter-FPGA communication is working with source-synchronous clocks. In this setup (Fig. 4.4), each FPGA uses Xilinx Clock managers (MMCM) 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.

4.2.1.2 Multicast Support

The network is designed to support multicast - specific device IDs 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. 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, FPGA C was chosen as the network interface since the DNSEAM PCIe is connected to this device as shown in Figure 4.4. 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,FPGAs B and A are 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.

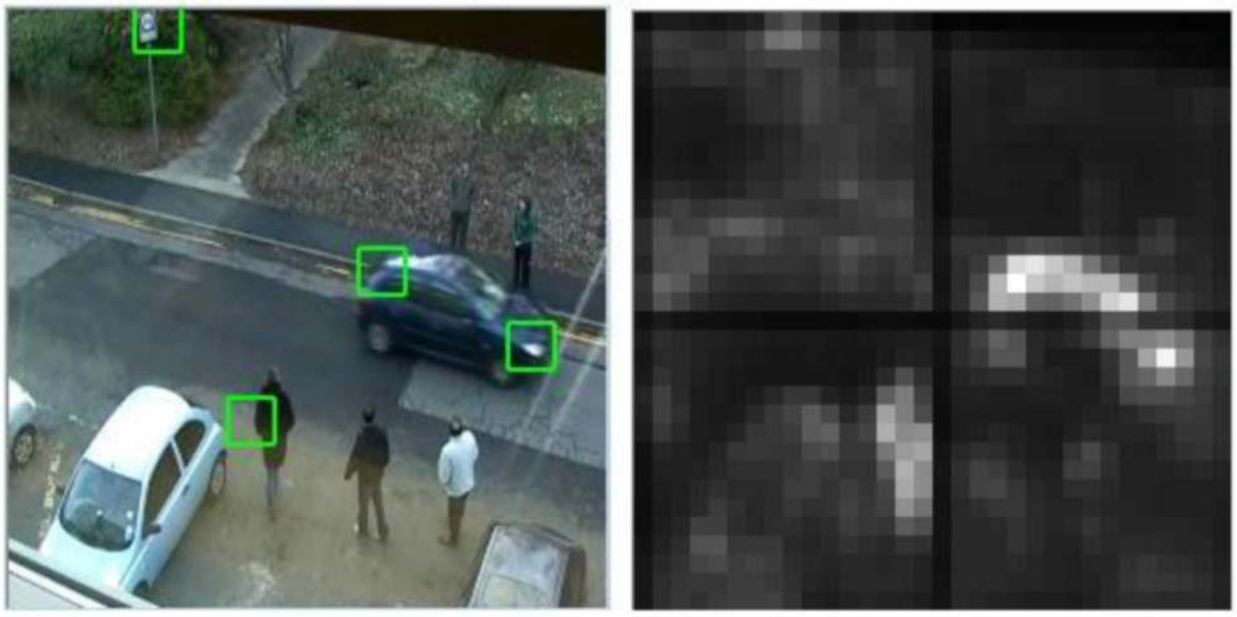


Figure 4.5: Result from Saliency based Attention accelerator- left image is 512x512 input frame and right image is corresponding grayscale saliency map from FPGA

4.2.2 Attention Results

An example of the output from the attention accelerator is shown in Figure 4.5. 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. The most salient chips are then input to the HMAX object recognition system.

4.2.3 Recognition Results

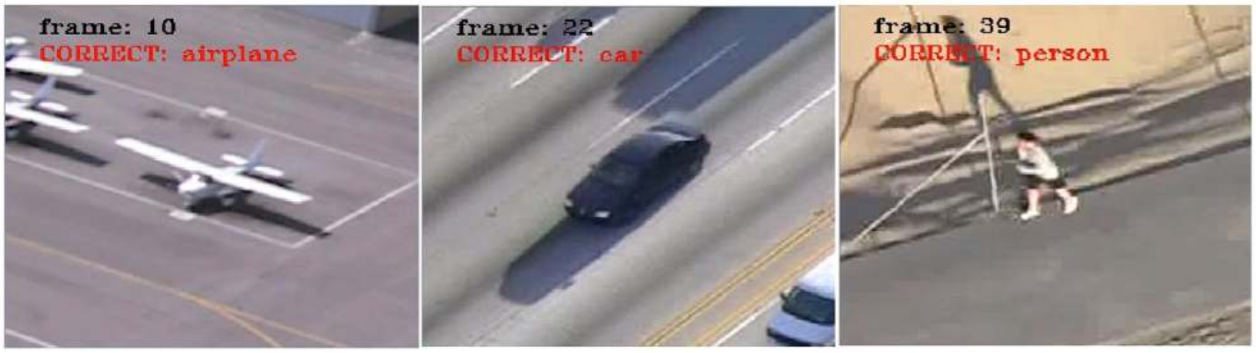


Figure 4.6: Results from the SVM Classifier with C2 vector generated from HMAX accelerator - Images are labeled with ground truth and the results of classification

Table 4.4. Classification Accuracy of implemented HMAX system for 53 Images

Number of S2-Patches	Correct Images	Percent Accuracy
1176 (only 4x4)	35	66.04
2000 (mixed)	51	96.22

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. The recognition system was validated using a database of 53 images and the classification accuracy is shown in table 4.4. It is to be noted that, while the size and nature of the patch dictionary directly affects the accuracy, the hardware remains the same.

4.3 Discussion and Futurework

The overall objective of this project was to design an artificial vision system which emulates the visual cortex. Since attention and recognition are the two main components of the visual process, the focus of this work was on one of those two these steps. Modeling cortical vision is in itself an active research area and there exists several variants of the models that were used in this work.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.

Some of the performance limitations of this setup are the overheads due to software. Currently the intermediate Focus Of Attention (FOA) step, which is the intermediate step between attention and recognition, and the S1C1 stages of HMAX are implemented in software. In the current system the computed saliency map is returned to the host CPU, where the software then identifies the most salient chips per frame, compute S1C1 and then transfers the C1 images of the salient chips back to the HMAX FPGA. Due to this increased I/O the PCIe link becomes the bottleneck.

In the current architecture, all pixels (12 orientations) at a given position of the C1-pyramid are broadcast to all the Processing Blocks (PB) in the CoRe16 module. Broadcasting C1-pixels of all orientation to all the Processing Blocks in the accelerator creates routing congestion issues. However, this would not be an issue if dense-S2 prototypes are used instead of sparse-S2 prototypes. In case of dense S2-prototypes we only need to broadcast C1-pixel of one orientation, and this could significantly reduce routing congestion issues.

One possible optimization to the current architecture would be to decompose all 8x8,12x12 and 16x16 S2-prototypes into smaller 4x4 prototypes. In other words, a 8x8 prototype would decompose into four 4x4 prototype, a 12x12 would decompose into nine 4x4 prototypes and 16x16 would decompose into sixteen 4x4 prototypes. This decomposition could eliminate some of the multiplexer and adder tree overheads in the CoRe16 module. However, this decomposition would require 'non-uniform' convolution with some of the decomposed 4x4 prototypes and could be complicated.

Another observation is that the HMAX model requires iterating through all the 4075 prototypes in the S2-prototype dictionary even if the same object was processed in the previous few frames(chips). In other words, the model has no memory of recently observed objects and simply interates through the entire dictionary for every frame. This brute-force computation seems inefficient. One possible solution could be to compare a new incoming frame with some of the recently processed frames (say last fifty or hundred frames with unique objects) and calculate a Maximum Likelihood Estimate(MLE). The MLE can be computed either in the attention (saliency) stage or S1 stage of recogntion (HMAX). If the MLE is high the entire S2C2 computations of HMAX can be skipped and the class of the object can be predicted.

It is also observed that the classifier needs a complete C2-vector (4075 x1) to classify an object. Instead, if we can accurately classify 'certain' objects by only computing a partial C2-vector, we could save the number of S2 computations and therefore potentially increase the system performance.This would require a dynamic re-ordering of the S2-prototypes to be processed and can be supported by the proposed automation flow.

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.

4.4 Conclusion

In this work, a parallel and reconfigurable accelerator for HMAX-based multi-class object recognition algorithm was presented. The core of the accelerator is a run time reconfigurable convolution engine which can perform variable size kernel computations in parallel. Intelligent multi-port memory architectures are designed to provide low-latency access to image data and patch coefficients. The architecture scales with number of pipelines and patch distribution. An automation flow for run-time configuration and execution is presented. Experimental results on Virtex6 FPGA platforms show 5X to 11X speedups and 14X to 33X higher performance-per-Watt over a CNS-based implementation on a Tesla GPU. This acceleration results in faster recognition which may be used to build quick-response systems for autonomous vehicle navigation, unmanned surveillance and robotics. Further, the higher energy efficiency of the accelerator could mean longer and reliable operation in a battery operated device. Prototype of the complete attention-recognition vision system was implemented on a multi-FPGA platform. Results show stable performance and high recognition accuracy, while providing speedups over existing CPU, GPU and FPGA implementations.

Bibliography

[1] Dini group DNSEAM documentation.

[2] Dini group DNV6F6 documentation.

[3] Nvidia tesla C1060 documentation.

[4] 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. In *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, pages 41–48, May 2011.

[5] Shekhar Borkar and Andrew A. Chien. The future of microprocessors. *Commun. ACM*, 54(5):67–77, May 2011.

[6] 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*, ISCA '10, pages 247–257, New York, NY, USA, 2010. ACM.

[7] Yongwha Chung, S. Choi, and V.K. Prasanna. Parallel object recognition on an FPGA-based configurable computing platform. In *Computer Architecture for Machine Perception, 1997. CAMP '97. Proc. Fourth IEEE Intl. Workshop on*, pages 143 –152, oct 1997.

[8] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 365–376, New York, NY, USA, 2011. ACM.

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

[10] D. Goshorn, Junguk Cho, R. Kastner, and S. Mirzaei. Field programmable gate array implementation of parts-based object detection for real time video applications. In *Field Programmable Logic and Applications (FPL), Intl. Conference on*, pages 582 –587, sept 2010.

[11] Venkatraman Govindaraju, Chen-Han Ho, and Karthikeyan Sankaralingam. Dynamically specialized datapaths for energy efficient computing. In *Proceedings of the 2011 IEEE 17th International Symposium on High Performance Computer Architecture*, HPCA '11, pages 503–514, Washington, DC, USA, 2011. IEEE Computer Society.

[12] H. Greenspan, S. Belongie, R. Goodman, P. Perona, S. Rakshit, and C.H. Anderson. Overcomplete steerable pyramid filters and rotation invariance. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, pages 222 –228, jun 1994.

[13] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. Understanding sources of inefficiency in general-purpose chips. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, pages 37–47, New York, NY, USA, 2010. ACM.

[14] Laurent Itti, Christof Koch, and Ernst Niebur. A model of saliency-based visual attention for rapid scene analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(11):1254–1259, November 1998.

[15] 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.

[16] S. Kestur, M. S. Park, J. Sabarad, D. Dantara, V. Narayanan, Y. Chen, and D. Khosla. Emulating mammalian vision on reconfigurable hardware. In *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, pages 141–148, April 2012.

[17] A. A. Maashri, M. DeBole, C. L. Yu, V. Narayanan, and C. Chakrabarti. A hardware architecture for accelerating neuromorphic vision algorithms. In *Signal Processing Systems (SiPS), 2011 IEEE Workshop on*, pages 355–360, Oct 2011.

[18] Janarbek Matai, Ali Irturk, and Ryan Kastner. Design and implementation of an fpga-based real-time face recognition system. In *Proceedings of the 2011 IEEE 19th Annual International Symposium on Field-Programmable Custom Computing Machines*, FCCM '11, pages 97–100, Washington, DC, USA, 2011. IEEE Computer Society.

[19] Florence Miau, Constantine S. Papageorgiou, and Laurent Itti. Neuromorphic algorithms for computer vision and attention, 2001.

[20] Jim Mutch, Ulf Knoblich, and Tomaso Poggio. CNS: a GPU-based framework for simulating cortically-organized networks. Technical Report MIT-CSAIL-TR-2010-013 / CBCL-286, Massachusetts Institute of Technology, Cambridge, MA, February 2010.

[21] Jim Mutch and David G. Lowe. Multiclass object recognition with sparse, localized features. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, CVPR '06, pages 11–18, Washington, DC, USA, 2006. IEEE Computer Society.

[22] Jim Mutch and David G. Lowe. Object class recognition and localization using sparse features with limited receptive fields. *Int. J. Comput. Vision*, 80(1):45–57, October 2008.

[23] H.E. Osman. Ensemble for high recognition performance FPGA. In *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE Intl. Conf. on*, pages 2187 –2192, oct 2009.

[24] M. S. Park, S. Kestur, J. Sabarad, V. Narayanan, and M. J. Irwin. An fpga-based accelerator for cortical object classification. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2012*, pages 691–696, March 2012.

[25] Robert J. Peters and Laurent Itti. Applying computational tools to predict gaze direction in interactive visual environments. *ACM Trans. Appl. Percept.*, 5(2):9:1–9:19, May 2008.

[26] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2:1019–1025, 1999.

[27] J. Sabarad, S. Kestur, Mi Sun Park, D. Dantara, V. Narayanan, Yang Chen, and D. Khosla. A reconfigurable accelerator for neuromorphic object recognition. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 813–818, Jan 2012.

[28] 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*, 29(3):411 –426, march 2007.

[29] V. Sriram, D. Cox, K. H. Tsoi, and W. Luk. Towards an embedded biologically-inspired machine vision processor. In *Field-Programmable Technology (FPT), 2010 International Conference on*, pages 273–278, Dec 2010.

[30] César Torres-Huitzil and Miguel Arias-Estrada. FPGA-based configurable systolic architecture for window-based image processing. *EURASIP J. Appl. Signal Process.*, 2005:1024–1034, January 2005.

[31] Tianguang Zhang, Haiyan Wu, A. Borst, K. Kuhnlenz, and M. Buss. An fpga implementation of insect-inspired motion detector for high-speed vision systems. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 335 –340, may 2008.