An Architectural Level Design Methodology for Embedded Face Detection

V. Kianzad[1*], S. Saha[1*], J. Schlessman[2], G. Aggarwal[1], S. S. Bhattacharyya[1], W. Wolf[2] and R. Chellappa[1]
[1]ECE Dept. and Institute for Advanced Computer Studies, Univ. of Maryland,College Park, MD
[2]Dept. of Electrical Engineering, Princeton Univ., Princeton, NJ
{vida, ssaha, gaurav, ssb, chella}@umd.edu, {jschless, wolf}@princeton.edu

ABSTRACT

Face detection and recognition research has attracted great attention in recent years. Automatic face detection has great potential in a large array of application areas, including banking and security system access control, video surveillance, and multimedia information retrieval. In this paper, we discuss an architectural level design methodology for implementation of an embedded face detection system on a reconfigurable system on chip. We present models for performance estimation and validate these models with experimental values obtained from implementing our system on an FPGA platform. This modeling approach is shown to be efficient, accurate, and intuitive for designers to work with. Using this approach, we present several design options that trade-off various architectural features.

Categories and Subject Descriptors

C.3 [**Real-time embedded systems**]

General Terms

Design

Keywords

Design space exploration, face detection, reconfigurable platforms, system-level models.

1. INTRODUCTION

In this paper, we present our study of design, modeling, architecture exploration, and synthesis of a face detection system. Face detection pertains to the discernment of the existence of human faces within an image or video sequence. This operation holds interest in fields such as surveillance, video archiving, and tracking, amongst others. Furthermore, many of the aforementioned fields have shown an increased focus on mobile and outdoor applications. This mandates consideration of power consumption, memory size, and area. Face detection is not a simple problem, however. While a significant body of work exists for elegant solutions to this problem, these approaches require computational power well beyond that deemed acceptable for mobile systems. It is also important to note that hardware solutions, while pursued, tend to consume significant system resources for what is in many cases a subsystem of a larger, more complex system.

As history has shown, area minimization has been graced by technology scaling. Unfortunately, however, the International Technology Roadmap for Semiconductors predicts a change in this trend. This prediction is coupled with an expected increase in leakage power, simply for powering devices. Intuitively, many of the application areas for face detection require fairly constant operation, making reliance upon device scaling questionable. In addition, these applications have an implicit and in many cases critical need for real-time performance. The real-time realization of such applications can only be achieved by aggressive application of parallel processing and pipelining as provided by multiprocessor systems. Such implementation involves the interaction of several complex factors including scheduling, interprocessor communication, synchronization, iterative execution and memory/buffer management. Addressing any one of these factors in isolation is itself typically intractable in any optimal sense.

With these issues in mind, we study in this work the design and synthesis of an embedded face detection system for a class of reconfigurable system-on-chips. In addition to providing design details and experimental results for a useful family of face detection architectures, the following contributions on design methodology emerge from our study.

- We develop a number of useful generalizations to the *synchronization graph* performance analysis model [6]. First, we demonstrate the first formulation and use of *multirate synchronization graphs*, which we show to be useful for compactly representing repetitive patterns in the execution of a multiprocessor system. Second, we integrate aspects of ordered transaction execution [5] with more conventional, self-timed execution in a flexible and seamless way. This demonstrates a new class of hybrid self-timed/ordered transaction designs. Furthermore, it demonstrates that the syn-

*These authors made equal contribution.

CODES+ISSS'05, Sept. 19–21, 2005, Jersey City, New Jersey, USA

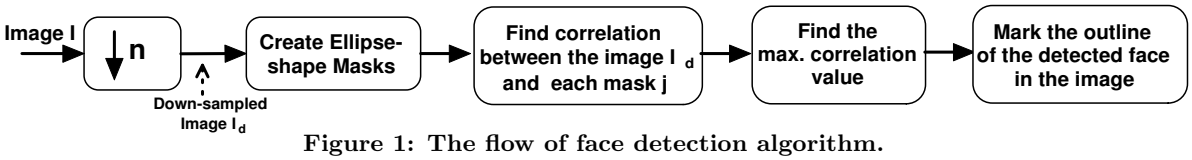

Figure 1: The flow of face detection algorithm.

chronization graph modeling methodology can be used to unify analysis across the entire spectrum of systems encompassing pure self-timed execution, pure ordered transaction execution, and the set of hybrid self-timed/ordered transaction possibilities that exist between these extremes. Third, we show how our multirate synchronization graph approach, together with hybrid self-timed/ordered transaction scheduling, can be used to effectively design systems involving extensive multi-dimensional processing (in our case, processing of video frames). Previous development of synchronization graph modeling has focused primarily on single-dimensional signal processing systems.

- We then apply our generalized synchronous graph modeling approach as a designer's aid for architecture analysis and exploration. In contrast, previous development of synchronous graphs has focused on their use as an intermediate representation in automated tools [6]).

- We show how for application/domain-specific design, the system designer can effectively use our new modeling approach as a way to understand and explore performance issues (regardless of whether the model is supported by the synthesis tools employed). Our parameterized construction of the multirate synchronization graph for the targeted class of application/architecture mappings together with our analysis of cycles in this parameterized synchronization graph concretely demonstrates the steps that are needed for this type of design methodology.

2. RELATED WORK

There is a vast expanse of work related to face detection, with significant focus on accuracy at the expense of computational complexity [10][3]. This work, as mentioned, lends itself well to software implementations on general purpose processors, however, does not make embedded and mobile implementations feasible. In recent years, more focus has been placed on hardware implementations of face detection. In addition, much of this work used a starting point of reconfigurable platforms for system implementation, similar to the work presented in this paper, e.g. a face detection system was implemented on a reconfigurable platform, with area results requiring multiple FPGA boards [3]. An ASIC implementation of a fairly complex face detection algorithm was presented, with real-time frame rates possible. This work, however, did not mention architectural exploration [8]. A framework for generalized object detection on a general purpose embedded processor was presented, as well [9].

This body of work suffers from a distinct lack of focus on design methodology and architectural exploration and there is little work that treats the problem of face detection with stringent attention to synchronization of data accesses and transfers, as well as resource assignment and task scheduling. Bridging this gap and facilitating more systematic coupling between face detection algorithms and their embedded implementations are major objectives of this work.

Figure 2: Results of applying the face detection algorithm to two images.

3. FACE DETECTION ALGORITHM

Face detection research has been an active area of research for the past few decades. There are several approaches that make use of shape and/or intensity distribution on the face. In this work, we use a shape-based approach as proposed in [4]. A face is assumed to be an ellipse. This method models the cross-section of the shape (ellipse) boundary as a step function. Moon [4] proves that derivative of a double exponential (DODE) function is the optimal one-dimensional step edge operator, which minimizes both the noise power and the mean squared error between the input and the filter output. The operator for detecting faces is derived by extending the DODE filter along the boundary of the ellipse. The probability of the presence of a face at a given position is estimated by accumulating the filter responses at the centroid of the ellipse. Quite clearly, this approach of face detection seems like a natural extension of the problem of edge detection at the pixel level to shape detection at the contour level. Moon [4] even provides formulations for propagation of the error by the shape geometry. This way one can predict both the localization and detection performance of the algorithms and adjust parameters according to the imaging conditions and performance specifications. Figure 1 shows the complete flow of the employed face detection algorithm. A few examples of detection outputs using the described approach are presented in Figure 2.

4. MODELING APPROACH

In this work, we build on the synchronization graph model [6] to analyze and optimize multiprocessor implementation issues of our system. This representation is based on iterative synchronous dataflow (SDF) graphs [2]. A brief introduction to these concepts is presented in this section.

In SDF, an application is represented as a directed graph in which vertices (actors) represent computational tasks, edges specify data dependencies, and the numbers of data values (tokens) produced and consumed by each actor is fixed. Delays on SDF edges represent initial tokens, and specify dependencies between iterations of the actors in iterative execution. Mapping an SDF-based application to

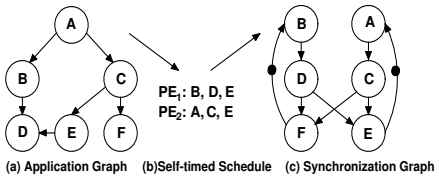

Figure 3: An example of (a) an application graph, (b) an associated self-timed schedule and (c) the synchronization graph resulting from this schedule.

a multiprocessor architecture includes i) assignment of actors to processors, ii) ordering the actors that are assigned to each processor, and iii) determining precisely when each actor should commence execution. In this work, we focus on the *self-timed scheduling strategy* and the closely-related *ordered transaction strategy* [6]. In *self-timed scheduling*, each processor executes the tasks assigned to it in a fixed order that is specified at compile time. Before executing an actor, a processor waits for the data needed by that actor to become available. Thus, processors are required to perform run-time synchronization when they communicate data. This provides robustness when the execution times of tasks are not known precisely or when they may exhibit occasional deviations from their compile-time estimates. It also eliminates the need for global clocks that coordinates all processors in lockstep.

The *ordered transaction* method is similar to the self-timed method, but it also adds the constraint that a global, linear ordering of the interprocessor communication operations (communication actors) is determined at compile time, and enforced at run-time [5]. The linear ordering imposed is called the transaction order of the associated multiprocessor implementation. Enforcing of the transaction order eliminates the need for run-time synchronization and bus arbitration, and also enhances predictability. The synchronization graph G_S [6], is used to model the self-timed execution of the given parallel schedule for an iterative dataflow graph. Given a self-timed multiprocessor schedule for graph G, we can derive G_s by instantiating a vertex for each task, connecting an edge from each task to the task that succeeds it on the same processor, and adding an edge that has unit delay from the last task on each processor to the first task on the same processor. Each edge (v_j, v_i) in G_s is called a synchronization edge representing *synchronization constraint*

$$\forall k, start(v_i, k) \geq end(v_j, k - delay(v_j, v_i)), \qquad (1)$$

where $start(v, k)$ and $end(v, k)$ respectively represent the time at which invocation k of actor v begins execution and completes execution, and $delay(e)$ represents the delay associated with edge e.

Once we construct G_s for a system, we use the *maximum cycle mean (MCM)* of the graph for performance analysis. The MCM is defined by

$$MCM(G_s) = \max_{cycle C in G_s} \{\frac{\sum_{v \in C} t(v)}{Delay(C)}\}, \qquad (2)$$

where $Delay(C)$ denotes the sum of the edge delays over all edges in cycle C. The MCM is used in a wide variety of analysis problems, and a variety of techniques have been developed for its efficient computation (e.g., see [1]). Examples of an application graph, a corresponding self-timed schedule and synchronization graph are illustrated in Figure 3.

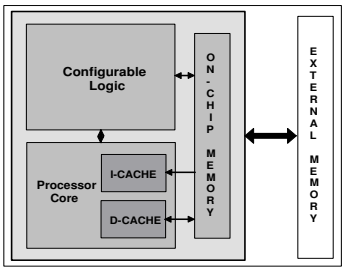

Figure 4: Target Architecture.

5. ARCHITECTURAL EXPLORATION

5.1 Target Architecture

The architecture we are targeting in this work is a reconfigurable system on chip. Such a system has both hardware and software resources and has potential for multiple processor development. It also includes on-chip and off-chip memory resources. The access to the off-chip memory is assumed to be through a shared bus. The on-chip memory access can be performed through a shared bus or via DMA. An example of such a system is Xilinx's Virtex II Pro. An abstract version of such a system is presented in Figure 4.

5.2 Profiling

Software profiling is usually the first step to any performance optimization approach. Profiling gives us information about the run-time of different program modules and where the most time is spent. Since we are targeting embedded systems, we are concerned with several optimization metrics such as time and energy consumption and hence it is important to identify the critical sections of the code that can benefit from hardware implementation. In this work we have used the FLAT profiler that provides loop/function level information [7]. The FLAT profiler identifies the correlation module that computes the correlation between the mask and the image (see Figure 1) as the candidate *core* (a core is defined the set of all loops whose execution is higher than a threshold value) for optimization and mapping to hardware. The output of FLAT is given in Table 1.

5.3 Base Model

Based on the results of profiling, we derived a system model such that multiple processing elements (PE) can concurrently execute multiple instances of the correlation function and thereby process masks simultaneously. This model requires us to have multiple masks and copies of the frame (for concurrent access) available on the chip, e.g 3 PEs require 3 frame copies present on chip. This increases the rate of memory accesses and hence power consumption, to minimize that we process the image one stripe at a time (we define a stripe to be the minimum size of the image that is processed in one pass), i.e. we run our masks set on a given stripe of the image and find the maximum correlation value and repeat the process for the next stripe, and continue until we have exhausted all the stripes and hence the image. For a set of N masks and n processing elements (PEs) (i.e. n masks can be processed simultaneously) it will take $m(= \lceil \frac{N}{n} \rceil)$ processing passes to cover all masks for a single stripe.

Figure 5 shows the implementation, transaction/execution order and synchronization model of our system. In this Fig-

Table 1: FLAT's output for the Face Detection Algorithm.

Loop Name	Frequency	Loop Size	Total Ins. (10^6)	% Exec
Program	1	96912	87053	100.0
mask-correlation	56392	600	83978	96.47

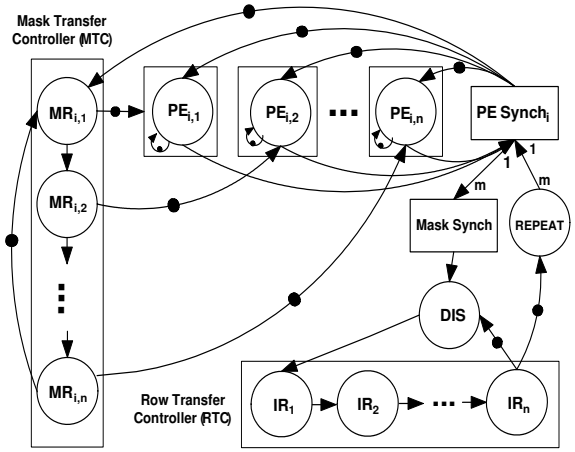

Figure 5: An SDF representation of the mapping of the algorithm onto the targeted architecture along with the associated synchronization structure.

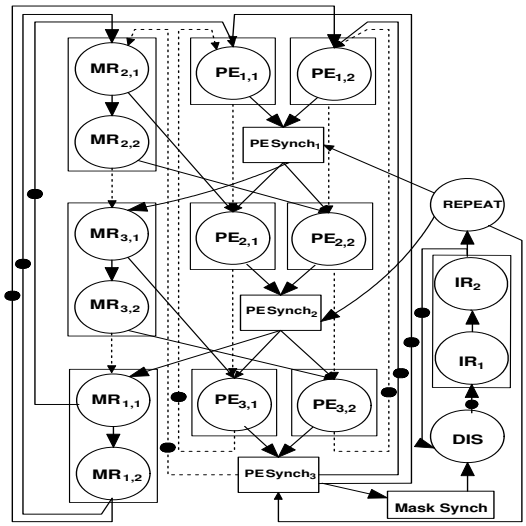

Figure 6: Example of an unfolded HSDF representation of the algorithm (by a factor of 3).

ure, the $MR_{i,j}$ actors represent the reading of mask j from the mask set i by processing element j, where i varies from 1 to m and j varies from 1 to n. This process takes $t(MR)$ time units. The masks are stored in the external memory and the Mask Transfer Controller (MTC), controls the reading to the dedicated Block RAMs (BRAMs) for each PE. The MTC conducts mask transfers one-at-a-time according to a repeating, pre-determined sequence in a fashion analogous to that of the ordered transaction strategy. Unlike conventional ordered transaction implementation, however, a transaction ordering approach is not used for all dataflow communications in the enclosing system. Each $PE_{i,j}$ actor represents the processing of the ith mask by PE_j, which takes $t(PE)$ time units. The DIS actor or the *Downsampled Image Source* represents the downsampling of an image stripe whose execution time is $t(DIS)$. The IR actor represents the reading of the downsampled stripe into the BRAMs one row at a time (execution time $t(IR)$). The *REPEAT* actor is a conceptual vertex that ensures that exactly m mask sets are processed for each new row of image data. No data actually needs to be replicated by the *REPEAT* actor; the required functionality can be achieved through simple, low-overhead synchronization and buffer management methods. The $PESynch_i$ represents the synchronization unit that synchronizes the start of the ith iteration of the PEs with the reading of mask set $i+1$. This unit receives data from the PEs and the *REPEAT* actor. The messages from the PEs confirm that they have completed the processing of one mask. These messages are sent at the end of each iteration. The production rate of m and consumption rate of 1 (shown on the $(REPEAT, PESynch_i)$ edge) indicates that the $PEsynch$ unit has to execute m times before the *REPEAT* actor is invoked again. The *Mask Synch* actor is again a conceptual actor and is not needed for any functional purpose. It represents the synchronization between m executions of the *PE Synch* actor and the corresponding execution of the *DIS* actor. That is, for each execution of the *DIS* actor, the *PE Synch* executes m times.

We use a self-timed model to coordinate interaction between the MTC and the PE cluster. At the start of reading a new set of masks, the controller must synchronize with the PEs to make sure that they are done with processing of the current masks. This synchronization process is represented as the edge directed from the *PE Synch* actor to the starting actor of the MTC block. The edge delay connecting $MR_{i,n}$ to $MR_{i,1}$ represents the initially-available mask data from pre-loading the first set of masks for a new image to their associated BRAMs.

A multirate SDF graph unfolds unambiguously into a homogeneous SDF (HSDF) graph [2], which in general leads to an expansion of the dataflow representation. An example of an SDF-to-HSDF transformation is given in Figure 6 for $m = 3$ and $n = 2$. For performance analysis, it is necessary to reason in terms of directed cycles in the HSDF representation.

5.4 Performance Analysis

We use the above modeling approach to understand and explore performance issues. From the model it may be observed that the HSDF cycles can be decomposed into a limited set of classes, where each class exhibits very similar patterns of cyclic paths. Also, many cycles in the graph are isomorphic, where by isomorphic cycles, we mean those in which the vertices and edges can be placed in one-to-one correspondence with one another so that corresponding vertices have the same execution times and corresponding edges have the same delays. Understanding these isomorphic relationships allows us to greatly reduce the number of MCM computations that actually need to be considered. Similarly, understanding the patterns of variation across certain sets

of similarly-structured cycles makes it easy to extract the critical cycles (the cycles with maximum MCM) from these sets.

Table 2 tabulates the different classes of cycles, along with a description and the MCM for each class, where the MCM is obtained by extracting the critical cycle in each class.

6. EXPERIMENTAL RESULTS

We have evaluated our proposed designs on a Xilinx ML310 development board.

6.1 Design Space Exploration

As it can be seen from Table 2, the system performance is a function of m, n, $t(PE)$, $t(MR)$ and $t(DIS)$. In this work, we assume that the number of masks and the mask sizes are fixed — as mentioned before, we use a shape-based face detection algorithm, where face is modeled as an ellipse. For operational value, the implementation should be able to handle variability in size of the faces as that information is not usually available a-priori. We handle this by creating several elliptical masks of varying sizes. Using all these masks, the detection is performed and the position of the face is determined by the quality of the filter response. The number of the masks required is bounded by the possible ellipticity of faces and by the size of the image. Our target application is a smart-camera based vision system. Given a particular smart-camera, the size of the images shot by that camera can be assumed to be fixed. Under these assumptions, and above masks parameters, the algorithm is able to perform robustly for faces of very different sizes.

Given the above justification, we can assume the $t(MR)$ and m are not variable (The mask size considered is 65×81 and the size of the mask set is 93.) $t(PE)$ is a function of several parameters but we only consider the degree of fine-grain parallelism (i.e. how many simultaneous operations can be performed within each PE), image size and the resolution (number of rows/columns considered) at which the image is compared with the mask. $t(DIS)$ is a function of frame sizes. To keep the design space manageable we fix the frame size (240×320) and vary the number of PEs (n), the steps and fine-grain parallelism (up to the HW limits). The stripe size for this implementation is 65×160. The execution times are obtained by multiplying the number of execution cycles for each node by the inverse of the clock frequency, which is $125MHz$ for the given board. The delays are given by the number of cycles required for the initial values to load from the source node to the destination node in the synchronization graph. From this we obtain the cycle means for the stated classes of cycles, which give the throughput as the inverse of the cycle mean amongst all possible cycles.

We observe that the cycles from *class 6* yield the largest cycle mean. This is expected, as the most critical path is that which involves the execution of PEs along with the reading of the next mask. To obtain a satisfactory throughput besides keeping up with the camera frame rate we need to vary the parameters without affecting accuracy. We observe that the throughput cannot be improved by merely increasing the number of PEs. We can increase the number of columns being skipped but that is also bounded by the accuracy of the face detection. Also the actual throughput is a function of the number of image stripes present and hence is affected by the resolution of the camera.

The number of PEs that may be implemented is limited by the area constraints. The board has 136 BRAMs present of which few are allocated for the use of other modules such as down-sampling unit, the powerPC and so on. 8 BRAMs are required for each PE, that sets an upper bound of number of PEs to 15. Also parallelization is possible within each PE: since the multiplications required for the calculation of each correlation value are independent of each other, at the same instant more than one multiplication may be performed. The number of multipliers available on the board limits this parallelization. There are 136 multipliers on board which limits the number of PEs to 13. The I/O buffers present on the board also impose serious restrictions on the number of PEs, since each PE communicates with the down-sampling unit, the BRAMs, the external DDR SDRAM memory controller and the output interface it has a significant number of I/O ports. This limits the maximum number of PEs that may be practically implemented to 6. We obtain execution times with parameters bounded by the above analysis. The results are presented in Table 3 in which n is the number of PEs, *degree of parallelism* is the number of additional multiplications done simultaneously in each PE and *steps* is granularity at which the image is correlated with a mask.

The maximum frame rate of applications of security and video surveillance toward which this work is targeted is 30 fps. With the current board and available hardware resources the implementation achieves a maximum frame rate of 12 fps. This entails the discarding of every two out of three frames at most, which can be tolerated for such applications.

6.2 Fidelity Analysis

In this section we calculate the fidelity of our performance estimations as the design parameters are varied,

$$Fidelity = \frac{2}{N(N-1)} \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} f_{ij} \tag{3}$$

where,

$$f_{ij} = \begin{cases} 1 & if sign(S_i - S_j) = sign(M_i - M_j); \\ 0 & otherwise \end{cases} \tag{4}$$

the S_is denote the simulated execution times ; and the M_is are the corresponding estimates from the MCM expression.

In case of fixed frame size for the 14 design points that we presented in Table 3 we get the fidelity of 0.868. In our model for simplicity we have assumed a small constant value for all synchronization actors (*PE Synch*) and that causes the difference between the estimated and experimental results. However, the high value of fidelity demonstrates the accuracy and robustness of our modeling technique. Alternatively, the models and design space exploration techniques employed in this paper can be applied on more powerful boards to explore implementations that approach or achieve the 30 fps target.

7. CONCLUSIONS

In this paper, we have targeted architectural design, modeling, and exploration of an embedded system for face detection on a reconfigurable system on chip. Our work provides design details and experimental results for a useful family of face detection architectures. Additionally, we have developed contributions to design methodology for embedded multiprocessor design through useful generalizations of

Table 2: Maximum Cycle Mean Expressions

no	Description of Classes	Cycle	MCM
1	reading of the masks	$MR_{2,1} \rightarrow MR_{2,2} \cdots \rightarrow MR_{2,n}$ $\rightarrow MR_{3,1} \cdots \rightarrow MR_{1,1} \cdots \rightarrow MR_{1,n} \rightarrow MR_{2,1}$	$\frac{m \times n \times t(MR)}{D_0}$
2	reading of image rows	$DIS \rightarrow IR_i \rightarrow DIS$	$\frac{t(DIS)+n \times t(IR)}{D_1+D_2}$
3	reading of image rows with synchronizations	$DIS \rightarrow IR_i \rightarrow REPEAT \rightarrow PESynch \rightarrow$ $MaskSynch \rightarrow DIS$	$\frac{t(DIS)+n \times t(IR)+3}{D_1}$
4	PEs synchronization	$PESynch_m \rightarrow PE_{1,i} \rightarrow PE_{2,i} \cdots \rightarrow$ $PE_{m,i} \rightarrow PESynch_m$ or $PESynch_m \rightarrow PE_{1,i} \rightarrow PESynch_1 \rightarrow$ $PE_{2,i} \cdots \rightarrow PESynch_m$	$\frac{m \times (t(PE)+1)}{D_1}$
5	synchronization of PEs and reading of masks	$PESynch_m \rightarrow MR_{2,1} \cdots \rightarrow MR_{3,1} \cdots \rightarrow$ $MR_{3,n} \cdots \rightarrow MR_{1,1} \rightarrow \cdots \rightarrow MR_{1,n} \rightarrow PE_{1,i}$ $\rightarrow PE_{2,i} \cdots \rightarrow PESynch_m$ or $PESynch_m \rightarrow MR_{2,1} \cdots \rightarrow MR_{3,1} \cdots \rightarrow$ $MR_{3,n} \cdots \rightarrow MR_{1,1} \rightarrow \cdots \rightarrow MR_{1,n} \rightarrow PE_{1,i}$ $\rightarrow PESynch_1 \rightarrow PE_{2,i} \cdots \rightarrow PESynch_m$	$\frac{m \times n \times t(MR)+m \times (t(PE)+1)}{D_3+D_4}$
6	synchronization of PEs with reading of masks in i) m=2k, and in ii) m = 2k+1	a)$PESynch_m \rightarrow MR_{2,1} \cdots \rightarrow MR_{2,n} \rightarrow$ $PE_{2,i} \rightarrow PESynch_2 \rightarrow MR_{4,1} \cdots \rightarrow MR_{4,n} \rightarrow$ $PE_{4,i} \cdots \rightarrow PESynch_m$ b) $PESynch_m \rightarrow PE_{1,i} \rightarrow PESynch_1 \rightarrow$ $MR_{3,1} \cdots \rightarrow MR_{3,n} \rightarrow PE_{3,n} \rightarrow PESynch_3$ $\cdots \rightarrow PE_{m,n} \rightarrow PESynch_m$	i) $\frac{\frac{m}{2}(1+n \times t(MR)+t(PE))}{D_3}$ ii) $\frac{\frac{m+1}{2}(t(PE)+1)+\frac{m-1}{2}n \times t(MR)}{D_3}$ i) $\frac{(\frac{m}{2}+1)(t(PE)+1)+(\frac{m}{2}-1)n \times t(MR)}{D_3}$ ii) $\frac{\frac{m+1}{2}(t(PE)+1)+\frac{m-1}{2}n \times t(MR)}{D_3}$

Table 3: Execution time for one frame for various design parameters

n	degree of parallelism	Estimation(ms) steps		Experimental(ms) steps	
		2	4	2	4
6	0	451	136	697	205
1	20	168	62	227	79
2	10	170	62	227	79
3	6	180	65	249	85
4	5	176	63	227	79
5	4	173	63	227	79
6	3	184	66	249	85

the synchronization graph modeling approach, and their application to processing of multidimensional signals. Using these approaches, we have presented multiple designs that expose important trade-offs between different architectural features. As future work, we plan to extend this model to explore hardware/software co-design. Additionally, in the current method, given several frames of a video, our approach will perform face detection independently for each frame, which discards the inherent temporal continuity between the frames. We plan to address these issues in our future work by employing more sophisticated techniques, such as Kalman filters and particle filters.

7.1 Acknowledgments

This research was supported by grant number 0325119 from the U.S. National Science Foundation.

8. REFERENCES

[1] A. Dasdan and R. K. Gupta, Faster maximum and minimum mean cycle algorithms for system performance analysis, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17, No. 10, pages 889-899. 1998.

[2] E.A. Lee, and D.G. Messerschmitt. Static scheduling of synchronous dataflow programs for digital signal processing. IEEE Transactions on Computers, February, 1987.

[3] R. McCready, Real-time face detection on a configurable hardware system, Proceedings of the The Roadmap to Reconfigurable Computing, 10th International Workshop on Field-Programmable Logic and Applications, 2000.

[4] H. Moon, R. Chellappa, and A. Rosenfeld, "Optimal edge-based shape detection," IEEE Transaction on Image Processing, Vol. 11, pp. 1209-1227, 2002.

[5] S. Sriram and E. A. Lee. Determining the order of processor transactions in statically scheduled multiprocessors. Journal of VLSI Signal Processing, March, 1997.

[6] S. Sriram and S. S. Bhattacharyya, Embedded Multiprocessors: Scheduling and Synchronization. Marcel Dekker, Inc., 2000.

[7] D. C. Suresh, W. A. Najjar, J. Villareal, G. Stitt and F. Vahid,"Profiling Tools for Hardware/Software Partitioning of Embedded Applications," Proc. ACM Symp. On Languages, Compilers and Tools for Embedded Systems (LCTES), June 2003.

[8] T. Theocharides, G. Link, N. Vijaykrishnan, M. J. Irwin, and W. Wolf, .Embedded hardware face detection,. in Proceedings of the 17th International Conference on VLSI Design, 2004.

[9] P. Viola and M. Jones, Robust real-time object detection,. in Proceedings of IEEE workshop on Statistical and Computational Theories of Vision, 2001.

[10] M-H. Yang, D. J. Kriegman, and N. Ahuja, Detecting faces in images: a survey,. IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 24, pp. 34.58, Jan. 2002.