# Partially Reconfigurable Vector Processor for Embedded Applications

Muhammad Z. Hasan and Sotirios G. Ziavras
Electrical and Computer Engineering Department
New Jersey Institute of Technology
Newark, NJ 07102
{mzh3, ziavras}@njit.edu

***Abstract*—Embedded systems normally involve a combination of hardware and software resources designed to perform dedicated tasks. Such systems have widely crept into industrial control, automotive, networking, and consumer products. These systems require efficient devices that occupy small area and consume low power. The device area can be minimized by reusing the same hardware for different applications. If possible, reconfiguring the hardware to adapt to the application needs is important for reducing execution time and/or power consumption. Partial reconfiguration facilitates minimum hardware changes to form a new configuration. We have designed a reconfigurable vector processor for embedded applications. Benchmark results on Xilinx FPGAs (Field-Programmable Gate Arrays) are presented involving partial reconfiguration for embedded applications that process vectors. Two approaches are studied toward performance evaluation. The first one estimates the required partial reconfiguration time based on the resources consumed by the corresponding vector kernels. The second approach uses the actual measurement of partial reconfiguration time on a platform that supports a particular type of partial reconfiguration. More than 20% performance improvement has been observed for benchmark kernels, without neglecting the reconfiguration overhead. A framework is proposed as well to efficiently manage the reconfiguration overhead for applications involving multiple kernels.**

***Index Terms*—vector processing, FPGA- reconfiguration, embedded applications**

## I. INTRODUCTION

Embedded systems have recently become important in many areas. For example, the TCP/IP protocol is implemented by dedicated hardware in many networking devices [1] and the anti-lock braking system (ABS) in cars is controlled by a program running on a microprocessor. The EDN (Electrical Design News) Embedded Microprocessor Benchmarking Consortium (EEMBC) has established itself as a pioneer in benchmarking for such applications [1]. It provides benchmarking for the consumer, telecommunications, industrial/automotive control, and office automation industries. Many of the above applications involve arrays and matrix data.

Vector processors can apply the same operation simultaneously to arrays of numbers (i.e., vectors). Therefore, a vector instruction could replace many loop iterations involving a single scalar instruction, where a single element of the resulting vector is calculated in each step as long as no hazards exist. Thus, the instruction bandwidth requirement is reduced whereas the data bandwidth requirement increases substantially [2]. Matrix arithmetic is often embedded in the auto/industrial control environment. Matrix elements are often accessed by a vector machine using stride addressing. Operation could also be carried out efficiently on a vector machine capable of index loading controlled by length.

Embedded systems often require efficient devices with respect to power and area consumption. Minimization of the area could be achieved by running different software components on the same piece of hardware. However, specialized hardware components could compensate for speed and power compared to software [3]. Moreover, reconfigurable hardware could ensure reusability of a given chip-area by many embedded applications, thus saving space. There is a clear demand for improved hardware platforms to enhance the performance of embedded applications under cost constraints. Current high-density FPGAs have the potential to satisfy this demand [4]. For a given number of I/O connections, FPGAs provide a low cost implementation platform for less than 1000 units. Moreover, their reprogrammable features make it easy to test and debug designs. Partial reconfiguration of FPGAs provides further potential for execution-time savings by customizing and adapting the hardware with minimum changes.

Related work on the Dynamic Instruction Set Computer (DISC) [5] involves partial reconfiguration to implement different instruction modules on an FPGA. It reports performance enhancement for a single customized application (image mean filtering) over a general-purpose implementation. OneChip [6] involves static configuration to implement a programmable functional unit (PFU). It reports performance enhancements for the discrete cosine transform (DCT) over a software implementation. The Reconfigurable Instruction Set Processor (RISP) [3] is a proposed model to target different design alternatives. It refers to other similar

works on custom reconfigurable logic rather than on FPGAs. The Vector Intelligent Random Access Memory (VIRAM**)** [7] involves a low cost, efficient architecture design for multimedia systems. It reports performance enhancement for EEMBC applications (from the consumer and telecommunication categories). Finally, [8] suggests the generation of several hardware cores with a scheduler to download these cores on-demand into the FPGA and motivates us towards the present work. Apart from [15] and [7], none of the above research considered the vectorized implementation of application kernels with customized hardware. Both works are based on simulation only and target ASIC-like implementations. Even in [15] and [7] hardware reconfiguration has not been considered. Although reconfiguration has been considered in the related work of [15], the studied operations are of generic nature, such as finding the absolute value, maximum/minimum, and bit-field extraction [16]. Knowing the potential benefits that vector customization and reconfiguration can provide, it is worth investigating them for embedded applications.

This paper is based on the work in [14]. It extends our earlier work with experiments of partial reconfiguration to support embedded benchmarks. Moreover, a detailed scenario of host-based dynamic embedded systems is presented and a framework is proposed as well to efficiently manage the reconfiguration process. In [9], a vector processor architecture in an FPGA is presented. We extended the architecture with partially reconfigurable hardware for a set of embedded benchmark applications that process vectors to investigate the value of partial runtime FPGA reconfiguration. Test results indicate 23% to 38% savings in execution time.

The paper is organized as follows. In Section II, we present the vector machine architecture. In Section III, estimated performance results for partial reconfiguration of a Xilinx Virtex II FPGA with selected EEMBC and MiBench [12] application kernels are presented. Experimental results are presented in Section IV for partial reconfiguration of a Xilinx Spartan II FPGA to support the above benchmark kernels. Section V enlists a framework for efficient reconfiguration management in dynamic embedded systems. The paper ends with conclusions.

## II. The Vector Machine Architecture

In our implementation, the target vector machine consists of a vector processor with pipelined 32-bit FPUs, a memory controller for pre-fetching, and several memory modules, as shown in Figure 1 [9]. It contains a program memory and eight interleaved data memory modules. There are six vector registers of 16 elements each. All the operations on vector data are vector-register based. Instructions are grouped into the load/store (standard, index, stride), arithmetic, comparison, and preprocessing categories.

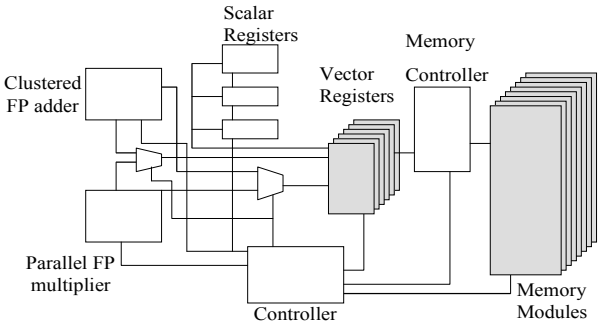


Figure 1. The vector machine architecture

The vector architecture was modeled in VHDL. Modelsim from Mentor Graphics was used to represent the model and simulate the design. Subsequently it was synthesized using Synplify-Pro from Synplicity. Finally, it was mapped to a Xilinx Virtex II XC2V6000 FPGA using the Xilinx 'Place and Route' tool. This FPGA resides in the Annapolis Micro Systems Wildstar II board [11], a configurable computing platform that can be accessed from a host computer using Application Programming Interfaces (APIs) developed by Annapolis. The system board communicates with the host machine using a PCI interface. The two FPGAs on the board can be programmed and reset from the host computer using APIs. APIs can also be used to transfer data to and from the FPGAs.

## III. Embedded Vector Customization on The Virtex II FPGA

We investigated various EEMBC [1] benchmarks for vector processing suitability. A selected group of applications from several categories of EEMBC embedded benchmarks were found suitable for vector processing. Proposed are various customizations for a set of EEMBC applications based on heuristics.

The autocorrelation between two samples of data is an analysis tool that represents correlation between two samples of a random process defined as follows:

$$\text{Autocorrdata}[k] = 1/N \sum_{n=0}^{N-1} \text{Data}[n] * \text{Data}[n+k], \text{ for}$$

k = 0, 1, 2,…, (K-1), N is the size of the vector.

We created a customized hardware piece that schedules a shifted version of the input as the multiplicand, avoiding another load.

In a High Pass Grey-scale (HPG) filter, the intensity of an image pixel is calculated considering the values of its neighbors within a window, as follows:

PelValue = F11*P(c-w-1) + F21*P(c-w) + F31*P(c-w+1) + F12*P(c-1) + F22*P(c) + F32*P(c+1) + F13*P(c+w-1) + F23*P(c+w) + F33*P(c+w+1)

A customized hardware piece was created that performs three consecutive loads into the same register avoiding

three registers; it completes one (longer) multiplication avoiding three (shorter) ones.

Special instructions invoking each of the above application customizations were created and synthesized in hardware to find the equivalent gate requirements for each. The full configuration time for 6M gates of Virtex II FPGA (using a 50 MHz clock) is 54 ms [10]. Based on this, the reconfiguration time for each customization was calculated proportional to its gate count. Each application was run on a basic vector machine [9] and on the vector machine with the appropriate customized hardware piece. The results are summarized in Tables I and II with the reconfiguration time (considered as an overhead) for different types of customizations.

In order to further extend the validity of our work, we searched for other embedded benchmarking suites. The University of Michigan has created such a suite called MiBench [12]. It is similar to that of EEMBC as far as the application groups are concerned but the applications within the groups are different. To its advantage, the MiBench suite makes available the source code (in the C language) for the various applications.

TABLE I.
DYNAMIC PERFORMANCE OF THE CUSTOMIZED AUTOCORRELATION

| Number of function values | Execution time(μs) | | Gate count | Reconfiguration time (μs) | Execution time savings (%) |
|---|---|---|---|---|---|
| | Standard platform | Customized hardware | | | |
| 100 | 514 | 339 | 1222 | 11.00 | 31.90 |
| 200 | 1028 | 678 | | | 34.04 |
| 300 | 1542 | 1017 | | | 33.33 |

TABLE II.
DYNAMIC PERFORMANCE OF THE CUSTOMIZED HPG

| Number of pixel values | Execution time(μs) | | Gate count | Reconfiguration time (μs) | Execution time savings (%) |
|---|---|---|---|---|---|
| | Standard platform | Customized hardware | | | |
| 100 | 1016 | 767 | 1556 | 14.00 | 23.13 |
| 200 | 2032 | 1534 | | | 23.82 |
| 300 | 3048 | 2301 | | | 24.05 |

A major vector operation in the modified two-dimensional Discrete Cosine Transform (2D-DCT) involves shuffling, selectively changing the sign, and adding different elements of an input vector, as follows for an n-element input:

*for i = 0 to (n-1)/2 do {*
*Tmp [i] = data[i] + data[n-i]*
*Tmp [n-i] = data[i] - data[n-i] }*

A customized hardware piece was designed that creates two vectors from the input data array such that they can be added to form the Tmp array implying savings for a load and an arithmetic operation.

The FFT transform that requires reordering of an array through bit-reversal operations is an application in the telecommunication category. In this process, applied to a $2^n$-element vector A, the elements are interchanged according to the following equations:

$A((2*i)+1) = A((2*i)+ 2^{n-1})$ and $A((2*i)+ 2^{n-1}) = A((2*i)+1)$.

We created another customized hardware piece that reorders a loaded register accordingly with minimum swapping.

Experiments were carried out with these customized pieces as described earlier. The results are summarized in Tables III and IV for different types of customizations. From all these tables, it is evident that even considering the reconfiguration overheads the customized machine performs better than the standard machine for these benchmark applications. Generally, the performance improvement is larger for a larger number of output samples. Moreover, the amount of improvement depends on the level of customization in the operations involved. The performance improvement for the customization of the HPG filter is smaller as it involves a single level of vector operations. For the DCT and FFT applications, shorter input vector lengths spell out performance degradation. But for larger input vector lengths, the performance improvements are impressive.

## IV. Partial Reconfiguration of The Spartan II FPGA

An important feature of the Xilinx FPGA architecture, our target platform in this work, is the ability to reconfigure a portion of the FPGA while the remainder of the design is still in operation; this is called *partial reconfiguration* [10]. The *difference-based method of partial reconfiguration* is accomplished by making a small change to the current design and then generating a bitstream based on only the differences between the current and the new designs. Switching the configuration is very quick, as the reconfiguration bitstream may be much smaller than the entire device configuration bitstream. We have chosen this approach (i.e., difference-based partial reconfiguration) to obtain good performance.

Our original testbed was the Annapolis Micro Systems Wildstar II board [11] containing two Xilinx Virtex II FPGAs. As this system supports only standard (full) configuration, we decided to migrate to a new

TABLE III.
DYNAMIC PERFORMANCE OF THE CUSTOMIZED DCT

| DCT size | Execution time(μs) | | Reconfiguration time (μs) | Execution time savings (%) |
|---|---|---|---|---|
| | Standard platform | Customized hardware | | |
| 16 * 16 | 116.71 | 39.53 | 85.8 | -7.39 |
| 32 * 32 | 233.41 | 79.06 | | 29.37 |
| 64 * 64 | 466.82 | 158.82 | | 47.75 |

TABLE IV.
DYNAMIC PERFORMANCE OF THE CUSTOMIZED FFT

| FFT data points | Execution time(μs) | | Reconfiguration time (μs) | Execution time savings (%) |
|---|---|---|---|---|
| | Standard platform | Customized hardware | | |
| 1024 | 91.61 | 40.16 | 94.49 | -46.93 |
| 2048 | 183.22 | 80.31 | | 4.60 |
| 4096 | 366.43 | 160.63 | | 30.38 |

testbed, the Digilent II [13] board containing a Xilinx Spartan II FPGA. This is a smaller FPGA with 30,000 gates. The advantage of this board is that the FPGA can be programmed using Xilinx programming tools that support partial reconfiguration. The steps involved in the configuration process were studied so that we can measure the exact (re)configuration time avoiding other overheads. To evaluate the effectiveness of partial reconfiguration, we synthesized and implemented the customized hardware kernels, as described in Section III, for the Xilinx Spartan II FPGA (XC2S30). Their functionality was verified on the FPGA board by applying a preset test vector at the design input and thus observing the result at the connected device output. Then, two partial reconfiguration files were created for switching the hardware between configurations. The partial reconfiguration files (12KB) were smaller than the full configuration file (42 KB). We measured the reconfiguration time in each case using the operating system time stamp and verified the target functionality after each reconfiguration. Both the standard and the customized hardware run at 51 MHz. The operating frequency of the hardware was doubled using a pipelined configuration in a later experiment.

In a real life application, several computation intensive kernels are executed one after the other to complete the application run. If these kernels are to be implemented in a piece of configurable hardware, then the hardware needs to be reconfigured each time a new kernel execution is scheduled. For example, the kernel execution involved in a CJPEG encoder are RGB to YCC, FDCT, quantization, run length coding, and Huffman coding [1]. As such, a configurable hardware needs to be reconfigured four times to accommodate them all during the complete process of encoding. We emulated such an environment by considering successive execution of two kernels (one each from the consumer and telecommunication categories) within each benchmark suite. Similar techniques of synthetically generated application cases have been considered in [16] and [24]. The combined execution time of the two application kernels was calculated in each benchmark category. The first execution time study focuses on the combined execution time of both application kernels on the vector platform that contains only the standard instructions. The second execution time study focuses on the combined execution time of both application kernels on their customized hardware. We considered the partial reconfiguration time for switching the hardware from the standard platform to the various customizations as an overhead. The partial reconfiguration time depends on the size of the file that is generated from the difference in the two individual configuration files for the pair of application kernels. The reconfiguration overhead for various kernels was measured as follows: 348 μs for Autocor, 352 μs for HPG, 200 μs for DCT, and 287.5 μs for FFT. The total overhead is the sum of the two partial reconfiguration times in each category. Considering this overhead, we calculated the total execution time on the customized hardware. The results are shown in Figures 2 and 3 for the EEMBC and Mibench applications, respectively.

The performance improvement ranges from 4.84% for a small number of output samples to 18.56% for a large number of output samples with the EEMBC applications. These figures are -26.65% and 51.25%, respectively, for the MiBench applications. As we can see from the above figures for a small size of output data, partial reconfiguration with customization provides nominal performance improvement. But as the size of the output data increases, there is a gradual improvement in performance. Also, we can see from the above figures that performance degradation occurs for a smaller size of output data in the MiBench applications. However, the rate of improvement is notable for each normal increase in the results space.

To compare the performance of the FPGA-based customized hardware modules with that of a general purpose processor for these applications, we created C codes modeling the functions of the aforementioned hardware. These codes were compiled and executable

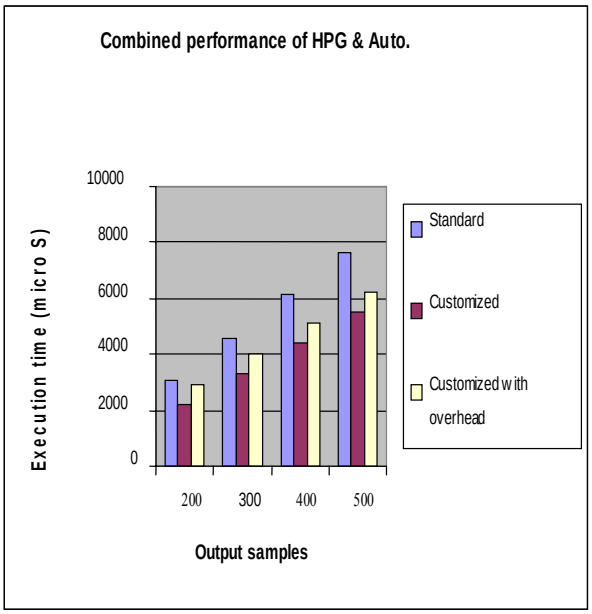


Figure 2. Combined performance of the EEMBC applications

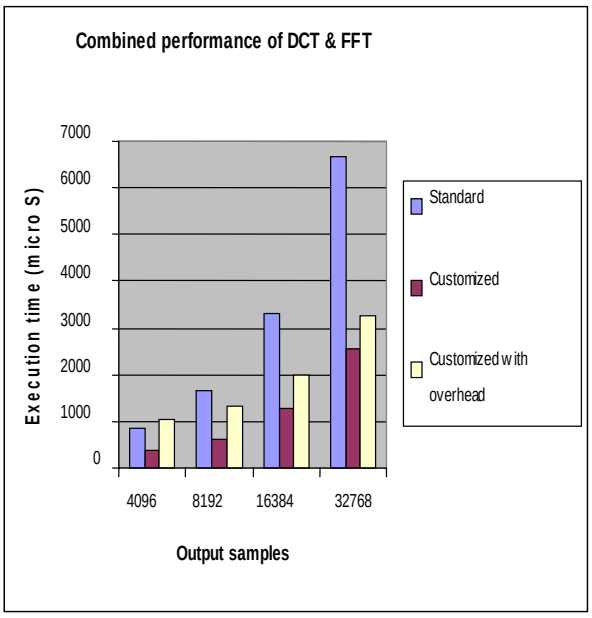


Figure 3. Combined performance of the MiBench applications

files were created using a Visual C++ compiler. They were executed on a PC in the batch mode and their execution time measured using operating system time stamps minimizing the measuring overhead. The results are furnished in Table V for the EEMBC benchmark category. As we can see from this table, the speed up obtained is generally significant. Note that the Xeon processor in the PC runs at 2.6 GHz whereas the custom hardware runs at just 51 MHz. We assumed a data transfer scheme for the custom hardware that overlaps host communications with computation. Although there are multiple, time multiplexed operations that run on the PC, still the above figures spell out performance improvement for those applications.

TABLE V.
PERFORMANCE COMPARISONS WITH A PC CONTAINING A 2.6 GHZ XEON PROCESSOR

| Correlation functional values | HPG pixel values | Combined execution time | | Partial reconfiguration time (µs) | | Speed up |
|---|---|---|---|---|---|---|
| | | Xeon machine (ms) | Customized hardware (µs) | For Autocor | For HPG | |
| 200 | 200 | 60 | 2212 | 348 | 352 | 20.60 |
| 300 | 300 | 72 | 3318 | | | 17.92 |
| 400 | 400 | 60 | 4424 | | | 11.71 |
| 500 | 500 | 90 | 5530 | | | 14.45 |

## V. DYNAMIC EMBEDDED SYSTEMS

There are several benefits of having dynamically reconfigurable hardware in a system. First, it facilitates dynamic adaptability of resources. Second, it enables hardware implementation of a large design in a piecewise fashion as the complete design could not be accommodated in the system. Reconfigurable hardware provides these benefits at a higher execution speed than respective software. Many dynamically reconfigurable systems (both embedded and non-embedded) involve a host processor. Such systems have been considered for case studies in [17][18][19][20][21][22]. The host is mainly used for control oriented, less computation intensive tasks. They are also used for making and supporting reconfiguration decisions. Hybrid software-hardware systems exploit the parallelism in hardware leaving the control of the overall system in software. As such, it may result in superior performance [23]. In this section, we consider host-based dynamically embedded systems that change behavior at run-time and/or process time-varying work loads. An example is a video encoding system that employs MPEG2 or JPEG depending on whether a recorded play or videoconferencing is chosen [24]. We assume target systems having limited numbers of reconfigurable hardware modules, such as FPGAs. We can target either a single FPGA embedded with multiple partially reconfigurable modules or several individually reconfigurable FPGAs. We propose a framework that considers the associated reconfiguration overheads for executing kernels on the FPGA(s) and dynamically decides on kernel execution either in the host or in the FPGA. In doing so, it ensures only performance gain, no degradation. Additionally, we present a kernel replacement policy that reduces the number of reconfigurations of the FPGA leading to overall power conservation.

The use of specialized hardware for embedded systems often implies shorter execution times and

possibly lower energy consumptions. Thus, reconfigurable hardware, such as an FPGA, is suitable for embedded system designs. However, the reconfiguration time of such devices may become a performance bottleneck for many applications involving several types of kernels. Also, the communication time between a host processor containing the various configurations and the reconfigurable hardware may represent a substantial overhead. As such, in an application it is imperative, if possible, to judiciously select kernel implementations either with host software or on the reconfigurable hardware so that net performance gain can be obtained. Moreover, since the reconfigurable hardware resources cannot often accommodate simultaneously all the kernels in an application, the replacement of kernels realizable in hardware is necessary. As this process involves additional power requirements [25], reducing the number of reconfigurations is of great importance. So, a befitting kernel updating strategy for the reconfigurable hardware should also be in place.

Let us first analyze the problem by considering a scenario of having a host processor and several FPGA units (either full or partially reconfigurable) that could target kernel execution. We assume that each FPGA can be programmed only from the host. Also, the execution time of the current kernel for the full set of data on the host is $t_H$ and on the FPGA is $t_{FPGA}$, the time to reconfigure the FPGA from its present configuration to the one required by the kernel is $t_{overhead}$, and the corresponding communication overhead involving the host is $t_{comm}$. A kernel is 'ready-to-execute' if all its predecessors in the task graph have completed execution. If there are multiple ready-to-execute kernels, then we choose the one with the smallest identification number. Our initial scheduling/reconfiguration policy, called Break-Even (BE) policy, contains the following steps for a given kernel; these steps are repeated until all the kernels of the application are scheduled.

1. Estimate the execution time $t_H$ on the host of the ready-to-execute kernel.

2. Check if the present FPGA configuration is the one required by the kernel. If 'yes', then set $t_{overhead} = 0$ and go to the next step.

3. If $t_H \leq t_{overhead} + t_{comm} + t_{FPGA}$, then execute the kernel on the host and exit. Else, proceed to the next step.

4. Reconfigure, if $t_{overhead} \neq 0$, an appropriate FPGA (as outlined below) with the customized kernel configuration.

5. Transfer any necessary data from host to the FPGA for execution.

6. Upload the results from the FPGA.

The time complexity of the Break-Even policy is $O(V)$; it grows linearly with the graph size making it suitable for execution on host at runtime.

To place a ready-to-execute kernel in an appropriate FPGA, we can follow these steps.

1. Check if any FPGA is completely available. If 'yes', then place the kernel in this FPGA and exit. Else, proceed to the next step.

2. For each FPGA, do the following:

   Compare the kernels present in this FPGA with the tasks/kernels in a window containing a preset number of kernels following the current kernel in the task graph. If there is a match, proceed to the next FPGA to repeat this process. Else, implement the kernel on this FPGA.

The time complexity of this replacement policy is $O(R*W)$, where W is the window size in number of kernels; R is the number of FPGAs. For practical systems, R is fixed and lies between 1 and 15. So, the time grows linearly with the window size. We assume that the execution flow in the task graph is partly known in order to identify the required kernels in the window.

In order to implement the above policy, it is needed to find $t_H$, $t_{FPGA}$, $t_{overhead}$, and $t_{comm}$ experimentally for various embedded kernels and data sizes. Thus, we could calculate the performance gains offered by these hardware kernels over their software implementations following the above policy. We have set up a test-bed to measure performance results for actual implementations of benchmark kernels on an innovative multi-FPGA system.

## VI. Conclusions

The importance of embedded systems has grown tremendously in the last decade. Such systems often require low energy consumption and small size. Reconfigurable hardware with partial reconfiguration capability can meet these demands. Embedded benchmarking indicates execution time savings of more than 20% for our partially reconfigurable, customized vector system, even when considering the overhead. The performance improvement figures are obtained both for a system supporting partial reconfiguration (by actual measurement of time) and for a system that does not support partial reconfiguration (by considering the amount of gate logic to be reconfigured). A framework has been formulated to efficiently take advantage of FPGA reconfiguration for the performance enhancement of applications containing multiple vector kernels.

## References

[1] The EDN Embedded Microprocessor Benchmarking Consortium, http://www.eembc.org/.

[2] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, Third Edition, *Morgan Kauffman Publishers Inc*., 2003.

[3] F. Barat, R. Lauwereins, and G G. Deconinck, "Reconfigurable Instruction Set Processors from a Hardware/Software Perspective", *IEEE Transactions on Software Engineering*, Vol. 28, No. 9, September 2002.

[4] X. Wang and S. G. Ziavras, "A Framework for Dynamic Resource Management and Scheduling on Reconfigurable Mixed-Mode Multiprocessor", *IEEE International Conference on Field-Programmable Technology*, Singapore, Dec. 11-14, 2005.
[5] M. J. Wirthlin and B. L. Hutchings, "A Dynamic Instruction Set Computer", *IEEE Workshop on FPGAs and Custom Computing Machines*, pp. 99-107, 1995.
[6] R. D. Wittig and P. Chow, "OneChip: An FPGA Processor with Reconfigurable Logic", *IEEE Workshop on FPGAs and Custom Computing Machines*, pp. 126-135, 1996.
[7] C. Kozyrakis and D. Patterson "Overcoming the Limitations of Conventional Vector Processors," *International Symposium on Computer Architecture*, San Diego, June 2003.
[8] D. Mesquita, F. Moraes, J. Palma, L. Moller, and N. Calazans, "Remote and Partial Reconfiguration of FPGAs: Tools and trends", *International Parallel and Distributed Processing Symposium* (IPDPS), April 2003.
[9] M. Z. Hasan and S. G. Ziavras, "FPGA-Based Vector Processing for Solving Sparse Sets of Equations**,"** *IEEE Symposium on Field-Programmable Custom Computing Machines***,** Napa, California, April 17-20, 2005**.**
[10] Xilinx, http://www.xilinx.com.
[11] Wildstar II Reference Manual, Document 12921, 2003, *Annapolis Micro Systems, Inc*., Annapolis, MD.
[12] Matthew R. Guthaus, Jeffrey S. Ringenberg, Dan Ernst, Todd M. Austin, Trevor Mudge, and Richard B. Brown, "MiBench: A free, commercially representative embedded benchmark suite**",** *IEEE 4th Annual Workshop on Workload Characterization*, Austin, TX, December 2001.
[13] Digilent D2XL board, http://www.digilent.cc/.
[14] M. Z. Hasan and S. G. Ziavras, "Runtime Partial Reconfiguration for Embedded Vector Processors," 4*th International Conference on Information Technology: New Generations*, Las Vegas, NV, April 2-4, 2007.
[15] R. Krashinsky, C. Batten, M. Hampton, S. Gerding, B. Pharris, J. Casper, and K. Asanovic "The Vector-Thread Architecture", *31st International Symposium on Computer Architecture (ISCA-31)*, Munich, Germany, June 2004.
[16] Krste Asanović, Brian Kingsbury, Bertrand Irissou, James Beck, and John Wawrzynek, "T0: A Single-chip Vector Microprocessor with Reconfigurable Pipelines," In *Proceedings 22nd European Solid-State Circuits Conference (ESSCIRC'96)*, Editor: H. Grunbacher, Editions Frontieres, September, 1996, pp 344-347.
[17] I. Robertson and J. Irvine, "A Design Flow for Partially Reconfigurable Hardware", *ACM Transactions on Embedded Computing Systems*, Vol. 3, Issue 2, May 2004, pp. 257-283.
[18] R. Lysecky, G. Stitt, and F. Vahid, "Warp Processors", *ACM Transactions on Design Automation of Electronic Systems*, Vol. 11, No. 3, July 2006, pp. 659-681.
[19] S. Ghiasi, A. Nahapetian, and M. Sarrafzadeh, "An Optimal Algorithm for Minimizing Run-time Reconfiguration Delay", *ACM Transactions on Embedded Computing Systems*, Vol. 3, Issue 2, May 2004, pp. 237-256.
[20] W. Fu and K. Compton, "An Execution Environment for Reconfigurable Computing", IEEE Symposium on Field-Programmable Custom Computing Machines**,** April 17-20, 2005.
[21] B. Greskamp, and R. Sass, "A Virtual Machine for Merit Based Run-time Reconfiguration", IEEE Symposium on Field-Programmable Custom Computing Machines**,** April 17-20, 2005**.**.
[22] E. M. Panainte, K. Bertels, and S. Vassiliadis, "Compiler-driven FPGA-area Allocation for Reconfigurable Computing," *Conference on Design, Automation and Test in Europe*, 2006, pp. 369-374.
[23] F. Ghaffari, M. Auguin, M. Abid, and M. B. Jemaa, "Dynamic and On-line Design Space Exploration for Reconfigurable Architectures," *19th International Conference on Architecture of Computing Systems,* Frankfurt/Main, Germany, March 13-16, 2006.
[24] J. Noguera, and R. M. Badia, "Multitasking on Reconfigurable Architecture: Micro Architecture Support and Dynamic Scheduling", *ACM Transactions on Embedded Computing Systems*, Vol. 3, Issue 2, May 2004, pp. 385-406.
[25] L. Shang, "HW/SW co-synthesis of low power real-time distributed embedded systems with dynamically reconfigurable FPGAs," *ASP-DAC'02* (2002) pp. 345–360.

**Muhammad Zafrul Hasan** received the B. Sc. in Electrical and Electronic Engineering from Bangladesh University of Engineering and Technology in 1988. He received the Master of Electronic Engineering from Eindhoven University of Technology (The Netherlands) in 1991 under Philips postgraduate scholarship program. Subsequently, he held faculty positions in an engineering college and in a university in Malaysia. Currently he is a Ph. D. candidate in Electrical and Computer Engineering department of New Jersey Institute of Technology. His research interest includes design and implementation of dynamically reconfigurable computing systems, computer architectures, and behavioral synthesis of digital systems.

**Sotirios G. Ziavras** received the Diploma in EE from the National Technical University of Athens, Greece, in 1984, the M.Sc. in Computer Engineering from Ohio University in 1985, and the Ph.D. in Computer Science from George Washington University in 1990. He was also with the Center for Automation Research at the University of Maryland, College Park, from 1988 to 1989. He was a visiting Professor at George Mason University in Spring 1990. He is currently a Professor of Electrical and Computer Engineering at NJIT, and the Director of the Computer Architecture and Parallel Processing Laboratory. He has authored more than 120 papers. His major research interests are advanced computer architecture, reconfigurable computing, and parallel processing.