iPACE-V1: A Portable Adaptive Computing Engine for Real Time Applications[*]

Jawad Khan, Manish Handa, and Ranga Vemuri

Department of ECECS, University of Cincinnati, Cincinnati, OH 45221-0030
{jkhan, mhanda, ranga}@ececs.uc.edu

Abstract. The iPACE-V1 (Image Processing Adaptive Computing Engine) is a portable, reconfigurable hardware platform, designed for real time, in-field image processing applications. IPACE-V1 has ample memory and the capability of full or partial reconfiguration without the need of a host computer. This paper describes the architecture of the hardware board along with the software design environment. We shall also discuss a real-time background elimination application for video images implemented on iPACE-V1.

1 Introduction

Adaptive or Reconfigurable Computers (RC) are those that can be programmed to change their hardware configuration and functionality dynamically to suit the application at hand; Field Programmable Gate Arrays (FPGA) are an essential building block of such systems. During the past few years RC systems have reached a remarkable level of maturity [3]. Several commercial and research platforms have been realized; WILDSTAR [1], SLAAC [4] and Rapid [5] are some of the well known reconfigurable platforms available now. However, none of these systems appear to be readily portable: All of them seem to require a host computer that is responsible for configuration of the processing elements and management of input and output data. Since reconfigurable architectures are particularly well suited for applications with repeated compute intensive operations, it is natural to use this technology for image processing applications, [2, 11] which are characterized by a large number of such operations.

We have developed a portable reconfigurable hardware platform, iPACE-V1, along with a CAD environment. Some of the main features of iPACE-V1 are Xilinx Virtex FPGAs, a large memory to capture and store raw image data, standalone dynamic full/partial reconfiguration and portability. We shall demonstrate that iPACE-V1 can achieve significant speedup for compute intensive applications in general and image processing applications in particular. iPACE-V1 can be used to implement in-field face recognition or on-the-go finger print analysis for law enforcement or military applications.

[*] This work is sponsered by The Dayton Area Graduate Studies Institute (DAGSI) and The Air Force Research Laboratory(AFRL) Research Program under contract number IF-UC-00-07.

© Springer-Verlag Berlin Heidelberg 2002

The rest of this paper is organized as follows. Section 2 describes unique features of iPACE-V1 and its architecture. Section 3 discusses the software design flow. In Section 4 we describe a real-time background elimination algorithm implemented on iPACE-V1 to demonstrate its usefulness. We discuss the results in Section 5 and finally, in Section 6 we discuss ongoing work and enhancements.

2 iPACE-V1 Architecture

Some of the currently available RC platforms that can be used for in-field applications, can be described as reconfigurable accelerators that need a host CPU to configure and use them [7, 6, 5, 1, 4]. The CPU to RC board communication is done through a peripheral bus such as PCI (Peripheral Component Interconnect)[10, 1, 4] and the processor-RC communication speed is limited by the effective bandwidth of the peripheral bus. Also a CPU based system tagging along with the RC accelerator makes it cumbersome for portable applications. These boards do not provide mechanisms to achieve partial reconfiguration without a host CPU [1, 4, 5]. iPACE-V1 has some features that make it unique among the line of existing RC boards, which we shall discuss now.

2.1 Design Goals

We developed iPACE-V1 with these design goals in mind:

1. *Dynamic Reconfiguration:* iPACE-V1 has the capability of full and partial dynamic reconfiguration without the need of a host computer. The debug FPGA can program the Main FPGA via the SelectMap Port at anytime during the application execution. The SelectMap port is retained after the Main FPGA is configured and hence we can read back the state of the FPGA too. Two different kinds of memories are provided to store several different configuration bit streams, that can be programmed one after the other. Xilinx XC18V04 serial PROMs provide a simple JTAG based programming solution and the larger 2Mbyte FLASH PROM is used as a configuration cache for several design bit streams.
2. *Portability:* iPACE-V1 is a small board measuring just 6 x 9 inches. All the necessary logic and memory is provided on this eight layer PCB, with components on the both, top and bottom layers. A small battery provides power for this system, which makes it portable. An LCD screen with a touch pad is interfaced, which provides the necessary I/O capabilities; obliterating the need of a bulky CRT monitor and a keyboard. An onboard color camera provides image data frames.
3. *Real-time Video Processing:* The iPACE-V1 has a fast SDRAM (Synchronous Dynamic Random Access Memory) connector, which can have SDRAM DIMMs upto a GigaByte; This memory is large enough to hold several image frames. The SDRAM can run upto 133MHz, therefore, providing a very fast and cheap storage area for data intensive applications. Along with the

SDRAM, the board has 4Mbytes of Zero Bus turnaround (ZBT) SRAM that has the unique feature of having no dead cycles between read and write cycles, therefore improving the effective throughput of the system. This memory can be used as a scratch pad area for storing intermediate values in a computation.

4. *Debug Support:* iPACE-V1 provides excellent debug facilities. A debug environment [14] is being developed targeted exclusively for this board, as a part of this project. We can read and write to any memory in the system. The clock is programmable and can be run in step mode as well. FPGA read-back is available through the retained SelectMap port.
5. *Interface to existing PC Platforms:* Virtex FPGAs are large enough to house a CPU softcore should the need for a processor based computing paradigm arise. However, we provide two mechanisms for interfacing existing PC platforms to iPACE-V1: An RS232 port is available along with a USB port for weak coupling of a processor board with iPACE-V1. For a more tighter coupling, an industry standard PC104 bus is provided.

2.2 Architecture Details

We present iPACE-V1 performance numbers in Table 1. Figure 1 shows the six basic modules of iPACE-V1 and their relationships to each other. Figure 2 illustrates a more detailed system architecture and finally, Figure 3 is a photograph of the actual board. The Basic modules of iPACE-V1 are explained below:

Main Processing Element	XCV 800 Virtex(upgradable)
Maximum Usable Gates	800,000(upgradable)
Programmable Clock Range	138 KHz - 100 MHz
Maximum Clock Frequency	200 MHz
Maximum ZBT SRAM Capacity	4 Mbytes
Maximum ZBT Bandwidth	400 Mbytes / sec
Maximum SDRAM Capacity	1 Gbytes
Maximum SDRAM Bandwidth	114 Mbytes / sec
Maximum FLASH ROM	4 Mbytes
Maximum Dynamic Reconfiguration Rate	400 Mbits / sec

Table 1. *iPACE-V1 Performance Numbers*

The *Debug Module* is responsible for bringing up the system to a known state at start up. It is housed in a XCV50 Virtex FPGA, which is referred to as Debug FPGA in this paper. This module is made up of several controller cores such as the programmable clock core and the configuration core etc.

The *Data Capture Module* is responsible for the initialization of the camera module or any other image acquisition device on board. Once the camera is programmed, this module can start capturing the data and dumping it in the

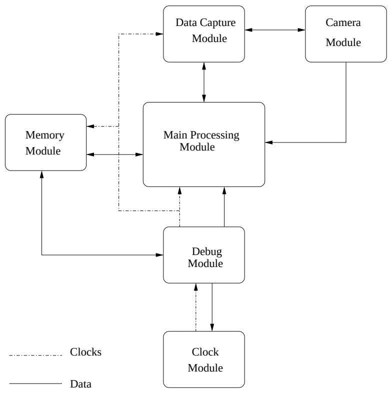

Fig. 1. iPACE-V1 Top Level Hardware Architecture

SDRAM. The data capture module is mapped to a XCV50 Virtex FPGA, which we will refer to as SDRAM FPGA.

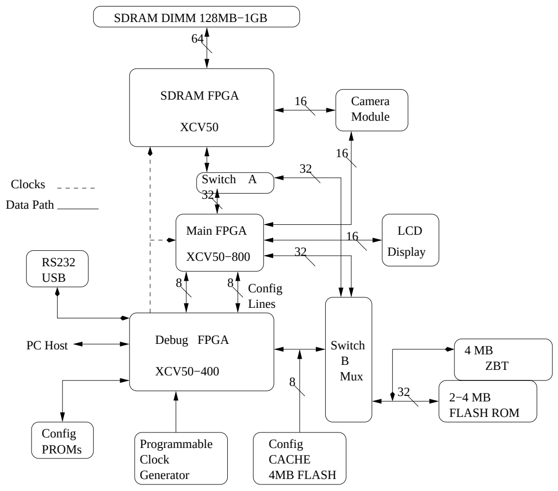

Fig. 2. iPACE-V1 Detailed Architecture

The *Main Processing Module* is the actual workhorse of the system. It can be mapped to any Virtex FPGA ranging from XCV50 (50,000 Gates) to XCV800 (800,000 Gates) [9]. In the current configuration we have a XCV300 FPGA (300,000 Gates). This is called the Main FPGA. The user design is targeted to this module. This module is directly connected to the Data Capture Module for using the data stored in the SDRAM. Further, it is connected to the Memory Module that contains ZBT SRAM and the FLASH PROM. Should the user need to work on the streaming video data without the need to capture it, we have provided a direct link from the camera to the Main FPGA. Sixteen general-purpose input/output lines are provided for interfacing an I/O device to this module. Currently we have a 320x240 LCD (Liquid Crystal Display) unit interfaced to the system.

The *Memory Module* contains the ZBT SRAM and the FLASH PROM, that are accessible from the main FPGA. A switch has been provided which transfers control to the Debug FPGA, to examine the contents of these memories for debugging purpose. In the current configuration, the data path width is 32 bits.

The *Clock Module* consists of a reference crystal oscillator and a programmable clock chip. The programmable clock chip can provide clock with frequencies in the range of 380 KHz up to 100 MHz. The resultant synthesized clock is fed to the Debug FPGA, which provides two final clocks MCLK and PCLK for the system. MCLK is the clock going to the Memory Module as well as to the other two FPGAs and the PCLK is processing clock, which is available to the other FPGAs exclusively.

The *Camera Module* has an OV7620 color camera chip by OmniVision Technologies Inc. This is a 640 X 480 interlaced/progressive scan CMOS digital color camera. This camera supports YCrCb 4:2:2 16 bit or 8 bit format, ZV port output format, RGB raw data format, CCIR601 and CCIR656 formats.

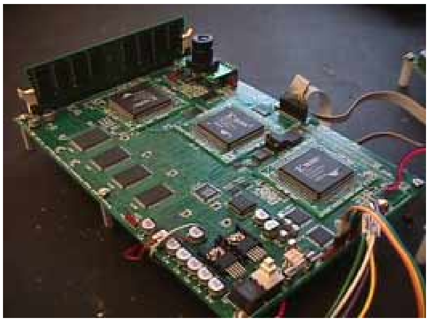

Fig. 3. iPACE-V1 Photograph

3 iPACE-V1 Software Design Environment

An integrated design environment has been developed to map an application to iPACE-V1. The software allows designs to be specified at the behavioral level of abstraction. Further, this environment takes advantage of partial reconfiguration capabilities of Virtex FPGA.

In a conventional FPGA design flow placement and routing of a netlist with a large number of nodes is a time consuming process. Additionally, it is very difficult to find correspondence between signal names in the high level description

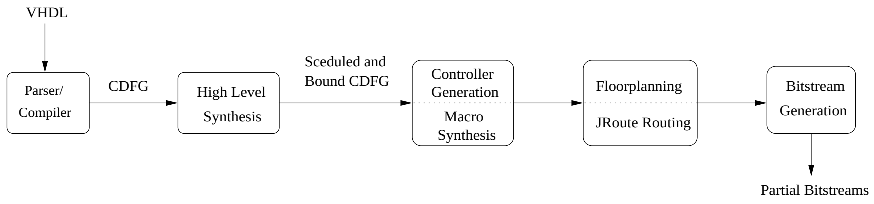

Fig. 4. iPACE-V1 Design Flow

and the signal names in the gate level netlist. We have proposed and implemented a methodology which uses macro synthesis. No logic synthesis is performed on the data path of the design as it is done in a conventional design flow. Macro netlist has a small number of nodes as compared to those of a gate level netlist. Therefore, placement and routing is much faster.

Our design environment is shown in Figure 4. An application is specified using a subset of hardware description language VHDL. We use Asserta [13] for high level synthesis [12]. Scheduled and bound CDFG (control flow data flow graph) is represented in SBlox [15]; SBlox is an uninterpreted operation sequence specification language for high level synthesis systems. SBlox representation specifies the task with a single thread of control. We generate the controller from the information specified implicitly in SBlox. Macro synthesis and interconnect generation is performed on the SBlox representation. Then, the operations in the SBlox descriptions are mapped to the macros specified in the macro library and inter-macro interconnect is generated. Also, the data path and control path macros are mapped to a JBits [8] based macro library: These macros are parameterized and relationally placed and routed. Floorplanning is performed on these macros and these are placed onto a two dimensional grid representing the FPGA. Finally, partial bitstreams are generated.

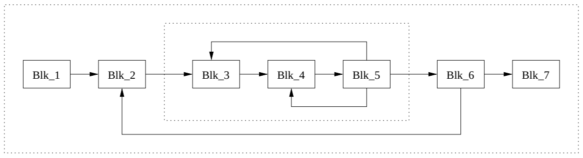

Fig. 5. Basic block partitioning and clustering

Any given task is partitioned into basic blocks, as shown in figure 5. Basic blocks are the entities that are responsible for computation or interaction with environment using ports. Conditional and unconditional control transfer and clustering of the blocks can be done depending upon resources available in the FGPGA. Partial bitstreams are generated for these clusters and the FPGA is configured using these bitstreams.

4 Case Study: Real Time Video Background Elimination

The problem of background elimination [16] can be defined in broad terms as determining whether each pixel in the image belongs to foreground or background and then displaying the pixels on the foreground only and removing the background pixels. This is a very compute intensive application, if done in real-time; It involves several matrix multiplications and matrix inverse calculations. We have implemented this algorithm to demonstrate two things: to show functional correctness of the board and to illustrate its usefulness for image processing in general.

4.1 Algorithm Description

Consider an m x n image. Each pixel in the image has a color value as given by the equation 1.

$$x = (R, G, B)^T \tag{1}$$

In this algorithm, the color of each pixel is assumed to have a normal distribution. We build a background model which consists of the mean value and the covariance matrix for each pixel of a given frame. A few initial frames are analyzed that are known to have only background in them for building the background model. Then the problem of detecting the foreground and the background is reduced to finding the Mahalanobis Distance of every pixel from this model and checking whether it is greater than or less than a given threshold value. If it is greater than the threshold then it is foreground, otherwise, it is background. We now provide the mathematical basis for this algorithm.

4.2 Building the Model

For a given set of k frames that are known to have only background in them, we compute the mean and the covariance of the color at every pixel.

Mean is given by:

$$\mu = 1/k \sum_k x_k \tag{2}$$

Covariance is given by:

$$\Delta = 1/k \sum_k (x_k - \mu_k)(x_k - \mu_k)^T \tag{3}$$

Mean of each pixel is a 3x1 vector and covariance is a 3x3 matrix. After the background model is made we acquire a new image for which we wish to subtract the background. For each frame, we find the Mahalanobis distance of every pixel. That is, for every pixel we compute:

$$d = (x - \mu)^T \Delta^{-1} (x - \mu) \tag{4}$$

If $d > T$ (some threshold), then we mark this pixel as foreground otherwise mark it as background. T is usually in the range of 20-100. We chose a value of 45 for the results presented in this paper. For those pixels in which inverse of the covariance matrix does not exist, we replace the covariance matrix with a 3x3 identity matrix; This, in turn, amounts to just calculation of the Euclidean Distance for that particular pixel.

4.3 Real Time Implementation

For this implementation we chose a frame size of 314 x 208 and a frame rate of 30 frames per second. This algorithm can be divided into two parts: model building and d-Value calculation.

In the first part we analyze 64 frames and keep on updating the mean and covariance values for every pixel in the frame. This operation requires matrix multiplication in (3) and also addition of the result to the previously stored value for each particular pixel. For matrix- multiplication, we instantiate four fast parallel multiplier cores. This allows us to unroll the matrix-multiplication loop in time and in space. As a final step we calculate the inverse of the covariance matrices for each pixel using a similar loop unrolling. We then go on to the second partition of the design in which we analyze any given input frame using this model at runtime and calculate the Mahalanobis distance for every pixel and classify it as foreground or background. This operation entails three matrix-multiplications which are done in a similar fashion. The camera pixel rate is 32bits/296ns where every pixel has 24 bits with 8 bits for each Red, Green and Blue color. This application executes at 27MHz inside the FPGA. The d-value calculation is done in a pipelined fashion and it takes about 7 cycles per pixel, therefore, the system is well within the real-time constraint.

5 Results

We present the preliminary results in Table 2, where we compare the results of the application executing on iPACE-V1 with a similarly coded application in MATLAB. The MATLAB code was running on a Pentium III at 1GHz with 256Mbytes of memory. We are getting a speedup factor of about 8 in the model building part and a speedup factor of about 113 in the d-value calculation part. The reason of disparity between the two is that the model building part is not designed to match the frame rate, since it is only a one-time operation. However, the d-value calculation is done at the frame rate and the result is a significant improvement. Figure 6 shows the background image in the first row, some objects in front of the background are shown in the second row and the last row shows the final background eliminated images[1]. Note that in the background eliminated images some area in the shadow of the object is also detected, albeit incorrectly, as foreground; This is a known limitation of the algorithm used.

[1] These are all color images

	MATLAB (Execution time in sec.)	iPACE-V1 (Execution Time in sec.)	Speedup
Model Building (64 frames)	11.81	1.41	8.34
d-value calculation (per frame)	2.22	0.012	113.60

Table 2. Table of Results

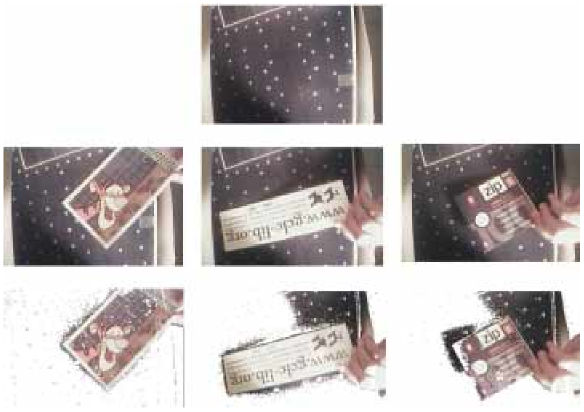

Fig. 6. Images with and without background

6 Conclusion and Ongoing Work

We have provided portability and all the necessary system resources on one RC board, which does not require a host computer to configure itself. We have also presented a software design flow for this system. A case study demonstrates the suitabiilty of this system for image processing applications. We are currently working on iPACE-V2 which will have Xilinx Virtex II platform FPGAs and better memory bandwidth than iPACE-V1. Some other demonstration applications are also underdevelopment.

7 Acknowledgments

We are grateful to Dr. Karen Tomko and Dr. Jack Jeans whose positive criticism of the board was a tremendous help in the design evolution process. We thank Ms. Kerry Hill, Al Scarpelli and Frank Scarpino for their support. Thanks go to Sohaib Khan, at University of Central Florida, Orlando, for help with background elimination application. Special mention must be made of Anna Acevedo in Xilinx Inc., Corninna Provant in Micron Technologies, and people in Texas Instruments, Maxim, Analog Devices and IDT who made the generous donations of IC parts on behalf of their companies.

References

[1] Inc. Annapolis Micro Systems. Wildstar Reference Manual. In *http://www.annapmicro.com/*, 2000.

[2] P. Athanas and A. Abbott. Image Processing on a Custom Computing Platform. In *4th International Workshop on FieldProgrammable Logic and Applications*, pages 156–167, September 1994.

[3] K. Compton and S. Hauck. Reconfigurable Computing: A Survey of Systems and Software, 2000.

[4] S. P. Crago, B. Schott, and R. Parker. SLAAC : A Distributed Architecture for Adaptive Computing. In *FPGA for Custom Computin Machines, Proceedings, IEEE Symposium on*, pages 286–287, April 1998.

[5] C. Fisher, K Rennie, G. Xing, S. G. Berg, K. Bolding, J. Naegle, D. Parshall, D. Portnov, A. Sulejmanpasic, and C. Ebeling. An Emulator for Exploring RaPiD Configurable Computing Architectures. In *11th International Conference on FieldProgrammable Logic and Applications*, pages 17–26, August 2001.

[6] Seth Copen Goldstein, Herman Schmit, Matthew Moe, Mihai Budiu, Srihari Cadambi, R. Reed Taylor, and Ronald Laufer. PipeRench: A Coprocessor for Streaming Multimedia Acceleration. In *ISCA*, pages 28–39, 1999.

[7] John Reid Hauser. *Augmenting a Microprocessor with Reconfigurable Hardware.* PhD thesis, North Carolina State University, 2000.

[8] Xilinx Inc. JBits Reference Manual Release 2.8. http://www.xilinx.com/products/jbits, 2001.

[9] Xilinx Inc. The Programmable Logic Data Book. www.xilinx.com, 2001.

[10] Ronald Laufer, R. Reed Taylor, and Herman Schmit. PCI-PipeRench and the SwordAPI: A system for stream-based reconfigurable computing. In Kenneth L. Pocek and Jeffrey Arnold, editors, *IEEE Symposium on FPGAs for Custom Computing Machines*, pages 200–208, Los Alamitos, CA, 1999. IEEE Computer Society Press.

[11] A. Lecerf, F. Vachon, D. Ouellet, and M. Arias-Estrada. FPGA based Computer Vision Camera. In *Proceedings of the 1999 ACM/SIGDA seventh international symposium on Field programmable gate arrays*, page 248. ACM Press, 1999.

[12] Giovanni De Micheli. *Synthesis and Optimization of Digital Circuits.* McGraw-Hill, Inc., 1994.

[13] N. Narasimhan. *Formal Synthesis: Formal Assertions Based Verification in aHigh-Level Synthesis System.* PhD thesis, University of Cincinnati, 1998.

[14] K. A. Tomko and A. Tiwari. Hardware/Software Co-debugging for Reconfigurable Computing. In *IEEE International High Level Design Validation and Test workshop*, November 2000.

[15] Ranga Vemuri and Rajesh Radhakrishan. SBlox: A Language for Digital System Synthesis. Technical Report No. 258/05/01/ECECS, University of Cincinnati, 2000.

[16] Christopher Wren, Ali Azarbayejani, Trevor Darrell, and Alex Pentland. Pfinder: Real-Time Tracking of the Human Body. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 780–785, July 1997.