Proc. SPIE FPGAs for Fast Board Development and Reconfigurable Computing, 2607, October 1995

Reconfigurable Processor for a Data-Flow Video Processing System

Edward K. Acosta, V. Michael Bove, Jr., John A. Watlington, and Ross A. Yu

Massachusetts Institute of Technology Media Laboratory
20 Ames Street, Cambridge, Massachusetts 02139

ABSTRACT

The Cheops system is a compact, modular platform developed at the MIT Media Laboratory for acquisition, processing, and display of digital video sequences and model-based representations of moving scenes, and is intended as both a laboratory tool and a prototype architecture for future programmable video decoders. Rather than using a large number of general-purpose processors and dividing up image processing tasks spatially, Cheops abstracts out a set of basic, computationally intensive stream operations that may be performed in parallel and embodies them in specialized hardware. However, Cheops incurs a substantial performance degradation when executing operations for which no specialized processor exists. We have designed a new reconfigurable processor that combines the speed of special purpose stream processors with the flexibility of general-purpose computing as a solution to the problem. Two SRAM based field-programmable gate arrays are used in conjunction with a PowerPC 603 processor to provide a flexible computational substrate, which allows algorithms to be mapped to a combination of software and dedicated hardware within the data-flow paradigm. We review the Cheops system architecture, describe the hardware design of the reconfigurable processor, explain the software environment developed to allow dynamic reconfiguration of the device, and report on its performance.

Keywords: video compression, model-based coding, reconfigurable hardware, data-flow computing

1. BACKGROUND: APPLICATIONS

The computational requirements of current digital video and audio representations are significant, but are proving manageable through either computational optimizations for general-purpose CPUs or hard-wired processors for single algorithms. More demanding scenarios that involve flexible encoding and decoding, support for multiple representations, or algorithmic extensibility, though, cannot be addressed through fixed processing hardware, and also may not be well-matched to current or planned general-purpose processor architectures.

A particularly computationally intensive set of tasks is the decoding of structured, model-based, or object-oriented representations, in which moving scenes are represented in terms of constituent parts (to the extent that machine-vision-like analysis algorithms are capable of understanding them) and scripting information that describes how to assemble the component parts for viewing.[1] Such representations have advantages in terms of efficiency, semantic searchability, and personalized/interactive display, and it appears to the authors that object-oriented video and audio representations will begin to take a place alongside waveform-based coding methods as soon as the resource demands can be met.

Always searching for ways to put larger amounts of reconfigurable computation in smaller packages, we in this paper explore the boundaries between specialized and general-purpose hardware.

2. BACKGROUND, CONTINUED: RECONFIGURABLE HARDWARE

Dynamic reconfiguration of hardware to customize a machine is not a new idea although the current generation of reconfigurable logic devices has generated renewed interest in the concept. Many early computers had microcoded control units that allowed their instruction sets to be altered, or tailored to specific applications, to improve performance.[2] The early 70's saw extensive research on dynamically altering the instruction set of a machine at compile time to improve performance by reducing bus traffic to main memory.[3,4] Most notably, the IBM 360 series of computers popularized this method of CPU control in the middle 70's. These machines allowed not only improved performance through custom modifications to the instruction set, but also the ability to emulate other instruction sets and hardware platforms.[5] Among the most well known (or infamous) of application specific microcoded instructions was the polynomial evaluation instruction of early VAX computers. This instruction was designed to improve performance by reducing instruction fetches to main memory.[6]

These machine designs had the advantage of a somewhat flexible instruction set. Custom instructions could be created for special applications or the instruction set could be tailored to the application as desired. They provided the ability to customize the architecture at the instruction level. That is, they provided instruction specific flexibility. Still, this method of specialization was severely limited by the fact that microcode only allowed changes of the sequencing of control signals and data-flow in a fixed datapath. The datapath itself was static and unalterable.

Even after improved memory and cache technology eliminated most of the traditional advantages of microcoding, the desire to implement application specific instructions continued. There are countless examples of the implementation of application specific instructions that contributed to the proliferation of CISC machines in the late 80's and early 90's.[7,8] Thus it is clear that even within the realm of general-purpose processing there is still a desire to employ custom hardware whenever it is economically feasible, or at least custom instructions for specific applications.

While the recent dominance of RISC architecture has obsolesced many of the earlier application-configurable or -optimized approaches to general-purpose computing, a new type of device has rekindled interest in the topic. These devices are SRAM based Field Programmable Gate Arrays (FPGAs), programmable logic devices that can be completely reconfigured under external control. This means that they are capable of assuming completely different architectures at different times. Unlike the earlier examples of customizable computing, though, these are aimed not at the instruction level, but instead at the application level. A datapath can be designed for each particular application and the devices can be reconfigured immediately before run-time. It is not only possible to change the order of occurrence of control signals for a fixed datapath, as with microcoded machines, it is possible to change the datapath itself. While custom computing machines (CCMs) can provide very high degrees of both application and datatype specificity, they are also very general in the sense that they can be quickly specialized to a wide range of possible applications. FPGAs are programmable logic devices that can be reprogrammed any number of times. They are similar to PAL devices, but offer much higher logic density and are capable of implementing more complex logic. The most advanced FPGAs currently offer 10,000 or more logic gates; however, a more appropriate measure of logic resources for these devices is the number of logic blocks offered, as will become clear shortly. Their objective is to obtain the density of sea-of-gates technology and the flexibility of PAL/PLD devices. Most FPGAs are constructed as an array of very fine grained logic resource blocks surrounded by routing resources which are used to wire logic together. The exact architecture and size of the blocks varies from vendor to vendor but generally consists of a 2-8 input/1-2 output combinational logic function, 1-2 flip-flops, and some control and multiplexing circuitry. Most FPGAs to date have been based on technologies like CMOS EPROM or EEPROM, which are electrically programmable read-only memory or electrically erasable programmable read-only memory respectively. Both require special programming apparatus that utilizes voltage levels not normally seen in the application circuit. Once programmed these devices are not re-programmable without being removed from the target circuit. As a result of the physical manner in which they are reprogrammed, they are typically good for only 100-10,000 programmings and start to fail after that.[9] Their use is thus limited to application development. Once the design has been debugged these devices become static in function, like PALs. The general trend of the current generation of FPGAs is towards SRAM based configuration methods. These FPGAs store their configuration information in standard static RAM cells interspersed throughout the chip. In contrast to their predecessors, they may be reconfigured any number of times without any wear or tear on the device. They use simple RAM based look-up tables (LUTs) to implement logic functions instead of gates made from transistors. The inputs to a logic function are treated as an address, the contents of the address is the logic function result. Since no special voltage levels are required for configuration, these devices can be configured without removing them from the circuit. In fact, it is possible for the target circuit itself to control the configuration. It is this last possibility that is most interesting and offers great potential for dynamic reconfiguration of application specific hardware.

A large number of researchers have very recently, and concurrently, begun to reexamine the feasibility from a cost/performance perspective of providing application specific hardware in light of the recent developments in FPGA technology. However, the emphasis is not on instruction specific specialization, but instead on application specific specialization (There are notable exceptions to this generalization. An example is the Spyder processor, a superscalar processor with reconfigurable execution units implemented with FPGAs, constructed by Iseli and Sanchez *et al.*[10]). Most research in this area is currently exploring two general trends. The first is the use of FPGAs in complete systems that allow the user arbitrarily to configure the hardware for specific applications to obtain super-computer type performance. Most of these target a specific application, like logic emulation, and do not have dynamic reconfiguration as a primary goal. Rather, they exploit the low cost and short design cycles of these devices as a welcome alternative to full custom ICs in obtaining improved performance. Many groups have reported spectacular success at speeding up various applications with FPGAs without the expense or time commitment of ASICs.[11,12,13]. A more interesting current area of research is the use of FPGAs in add-on boards and other closely coupled co-processors that assist GPPs in performing certain parts of applications. Typical designs attempt to move two distinct types of operations to hardware to speed applications:

- *Bit parallel computations:* These computations exhibit a high degree of parallelism and employ non-standard data types. GPPs generally provide poor performance in each of these cases and thus benefit greatly from co-processor assistance.
- *Computationally intensive inner program loops:* Typically less than 10% of program code accounts for 90% of program execution time,[5] so implementing these sections of code in hardware can greatly improve performance.

In both cases designs of this nature attempt to use FPGAs to provide a flexible computational substrate that is dynamically adaptive to the application of interest. We chose to discuss a few of these systems further, as their presence has been a motivating factor in the design of the programmable processor for Cheops.

Perhaps the first custom computing machine was the Anyboard, a PC plug-in board that provided five Xilinx FPGAs together with local RAM store to provide a rapid-prototyping environment for digital system development.[14] Anyboard, constructed in 1992, utilized Xilinx XC3042 FPGAs and took over five seconds to reconfigure.[15]. For this proprietary system, dynamic reconfiguration at run time was too computationally costly. Anyboard's chief contribution was the creation of a pioneering reconfigurable hardware platform for custom computing. Although its purpose was not to augment the functionality of a GPP, it served as the inspiration for other custom computing machines with this goal in mind.

Another notable custom computing machine is SPLASH, constructed at the IDA Supercomputing Research Center. This architecture consists of up to sixteen SPLASH boards that connect to a Sun Sparcstation through a separate interface board. The interface board provides four DMA channels capable of providing a total bandwidth of 50 MBytes/sec. to the SPLASH boards. Each SPLASH board consists of 16 Xilinx XC4010 FPGAs fully connected through a 16x16 crossbar switch. Additionally, 36-bit paths run between adjacent FPGAs which are laid out in a linear array.[16] The linear bus that connects adjacent FPGAs may also be extended across multiple boards should they be employed. SPLASH has several key advantages over its predecessors. First the FPGAs used are high density, 10,000 gates per chip, and thus provide enough hardware resources to implement large scale architectural designs. In addition, the programming environment is vastly improved over earlier machines through the use of HDLs (Hardware Description Languages) which allow designs to be specified at the behavioral level. Finally, the newer FPGAs have configuration times that are several orders of magnitude less than Anyboard. Several applications have been successfully demonstrated on this machine and SPLASH2 is under development to overcome I/O and internal connect bandwidth problems encountered.[17]

The virtual computer designed by the Virtual Computing Corporation and described by Casselman, *et al.*,[18] is not in the class of machines under discussion. It is a complete super-computing platform designed to assist a host workstation. It consists of a large array of Xilinx 4010 FPGAs interspersed with I-Cube IQ160 reconfigurable routing chips. Despite the difference in design, we mention it here because the authors report that it can be completely reconfigured in 25 milliseconds. A dedicated fast 64-bit I/O port facilities the reconfiguration operation. No other researchers have reported reconfiguration times similar to these for this family of FPGAs.

The PRISM-II machine, built at Brown University by Athanas, Agarwal, and Silverman, *et al.*.[3] employs three Xilinx 4010 FPGAs closely coupled to a AM29050 processor to provide flexible co-processor support. This project is perhaps the most sophisticated to date in that the group has made considerable progress in generation of hardware modules directly from high level C source code.[19] The goal of PRISM is to improve performance of computationally intensive tasks by using information extracted at compile time to automatically synthesize application specific hard-

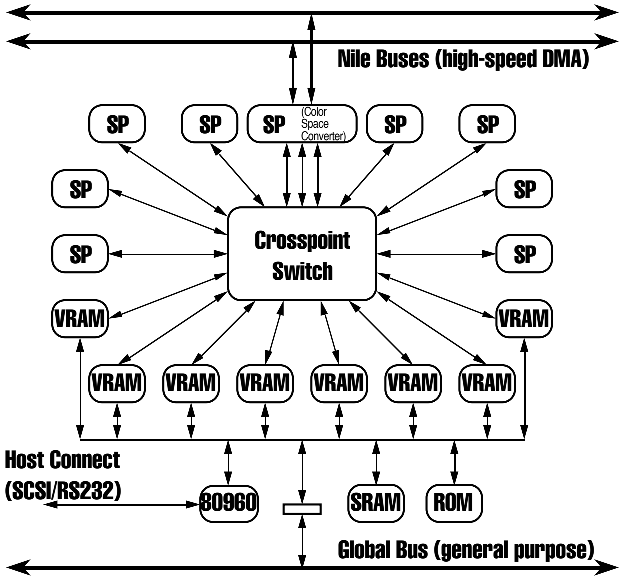

Figure 1: Block diagram of the Cheops processor module. Blocks labeled SP represent specialized stream processors, while those labeled VRAM are banks of memory equipped with high-speed DMA controllers. Resource allocation and user interaction are handled by an Intel 80960CF processor. The Nile Buses are high-speed DMA channels used for transferring data from module to module.

ware to supplement the computational abilities of the GPP. The FPGAs fulfill the role of a custom co-processor, but an adaptive one that can be specialized to each individual application. The PRISM architecture is very similar to that of our programmable processor as we shall soon see.

As a final note DeHon, *et al.*[20] have proposed the integration of FPGAs with conventional GPPs in a new device called a DPGA, Dynamically Programmable Gate Array. These augment a core general-purpose processor with an array of reconfigurable logic resources on the same chip. Each logic element in the array has local store to store several configurations for fast context switches between applications. While the availability of these devices is many years away, their feasibility is unquestionable. Consequently, they represent a natural extrapolation of dynamically reconfigurable hardware technology and indicate that the paradigm for flexible custom computers has been established.

3. THE CHEOPS SYSTEM

The hardware environment in which we are conducting the present experiment is Cheops, a compact data-flow video processing system designed by the Information and Entertainment Section of the MIT Media Laboratory for experiments in multidimensional digital signal processing and object-oriented video representations.[21] The philosophy behind the Cheops processor module is to abstract out a basic set of computationally intensive operations required for real-time performance of a variety of desired applications. These operations are then embodied in specialized hardware provided with a very high throughput memory interface, and controlled by a general-purpose processor. The processor module comprises eight memory units communicating through a full crosspoint switch with up to eight stream processing units. Each memory unit is made up of dual ported dynamic memory (VRAM) and a two-dimensional direct memory access (DMA) controller for transferring a stream of 16-bit data through the crosspoint at up to 40Msample/sec. Specialized stream processors attached to the switch perform common mathematical tasks such as convolution, correlation, matrix algebra, block transforms, spatial remapping, or non-linear functions. These processing units are on removable sub-modules, allowing reconfigurability and easy upgrade. A general-purpose central processing unit – an Intel 80960CF, clocked at 32MHz – is provided for sequencing and controlling the flow of data among the different functional units, implementing the portions of algorithms for which a specialized processor is not available, and performing higher-level tasks such as resource management and user interface. Figure 1 shows a simplified view of the topology of a single "P2" processor module.

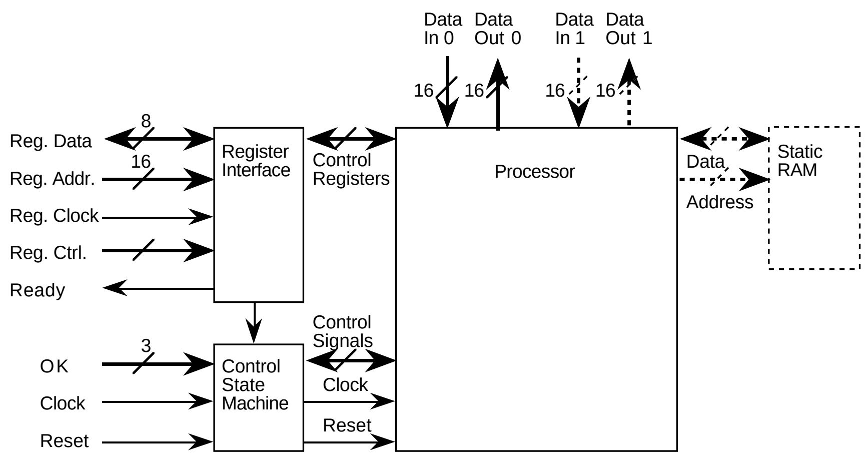

Figure 2: Block diagram of a typical Cheops stream processor. Datapaths at top are the high-speed stream interface to the crosspoint switch; the second exists only on two-port processors. SRAM exists only on two-phase processors, or those with lookup tables. The register datapath is used by the general-purpose CPU on the processor module to configure the stream processor.

In order to allow the same software to run optimally (*i.e.* in as parallel a fashion as possible) on differently-configured Cheops systems, and in conjunction with other simultaneous tasks, we have developed a resource mangement daemon called NORMAN. It manages the system resources, such as the currently installed stream processors and the Nile Buses, maintaining scoreboards and queues for each. A user program – a process which handles file I/O, interaction, and so forth – will pass a linked-list data structure describing the data-flow portion of the algorithm to the daemon. An individual stream transfer making up a sub-part of the data-flow graph occurs when all prerequisite data, an appropriate stream processor, and a destination are available. Transfers may optionally be made dependent upon a real-time clock (counting video frames on a selected input or output module) to permit process synchronization with video sources or displays.

All stream processors constructed to date conform to the generic model diagrammed in Figure 2. The register interface is used by the 80960 to set up the processors. The register interface and control state machine are usually implemented as individual PALs, but if a large FPGA is used as the main processor it may have enough extra terms available to implement these functional blocks as well. Because one removable submodule can hold two stream processors, it is possible for a single stream processor to have two inputs or two outputs, though in so doing it disables the other unit on the submodule when enabled. Stream processors may maintain as much internal state as necessary, though if the delay between input and output is very long and the processor can itself hold an entire transfer's worth of data, the processor operates in a two-phase fashion, where first it acts as only a destination, then as a source. While some of Cheops' stream processors (*e.g.* the sequenceable multiply-accumulate processor) are quite flexible, Cheops lacks support for high-speed parallel execution of tasks for which the specialized hardware is not a good match. In addition, use of the 80960 for significant amounts of processing detracts from its efficient execution of NORMAN, a demanding task in its own right.

4. THE STATE MACHINE PROCESSOR

The State Machine is a new type of computational resource for the Cheops processor module. It appears to the processor module as an ordinary submodule and uses the standard submodule interfaces. However, the State Machine is really a complete high-performance computational platform. The objective of the design of the State Machine was to implement a stream processor that would work within the physical (10 $\times$ 14cm), electrical and logical constraints of the current stream processor interface while satisfying the dual requirements of general-purpose flexibility and special purpose speed. Thus, within the design requirements imposed by the Cheops system, we tried to organize computational resources such that the architecture remained general enough to accommodate as many algorithms and computations as possible. The State Machine uses the current generation of dynamically reconfigurable FPGAs

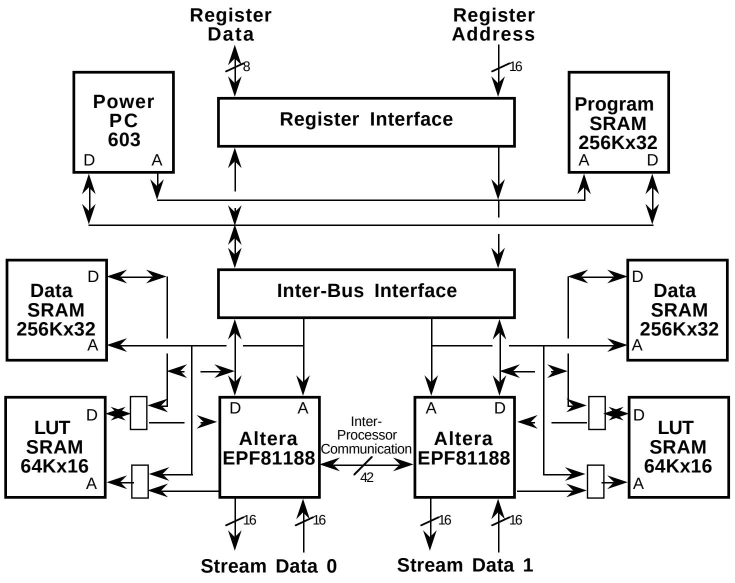

Figure 3: Block diagram of the State Machine processor. Register interface datapaths are at top, while stream datapaths are at bottom. Control datapaths are not shown.

under the control of a general-purpose processor to satisfy these requirements. The flexibility is provided by the run-time reconfigurability of the FPGAs, the main computational elements. Each State Machine application can have its own specific hardware architecture. The resulting performance derives from the ability to implement a custom computational architecture for any arbitrary task (subject to the gate-count and signal-routing resources available), in many cases achieving a result per clock cycle.

The general-purpose CPU may augment the computational abilites of the FPGAs for certain configurations. Additionally, it is capable of controlling the FPGA configuration process. This ability allows the designer of a State Machine application arbitrarily to choose the boundary between hardware and software implementation of a design. The State Machine is able to appear as a dedicated stream processor for an arbitrary application without violating the data-flow model within Cheops. For any given application, throughput and latency characteristics are dependent on the extent to which custom hardware is utilized.

To be maximally flexible the State Machine must accommodate three different types of usage. First it must be capable of acting as two completely independent stream processors, each stream processor using one of the two crosspoint ports on the card. It must be able to operate as one large stream processor using one or both ports. It may also sometimes be useful for it to act as two stream processors working in tandem. A further requirement is that it must be completely configurable from the register interface. Figure 3 is a block diagram showing the main datapaths. For configurations where the State Machine is to be used as two independent stream processors, each FPGA and its associated memory bank can be operated independently of the other with no interference. For these applications the processor can assist one or both (or neither) of the FPGAs with their respective tasks. The inter-FPGA bus allows the FPGAs to pass control signals and data for single processor applications. Also, a set of pass gates connect the two address buses so that they may appear as a single bus when enabled. This feature is useful for applications that require address remapping such as motion compensation and image warping, allowing one FPGA to generate addresses while the other shuffles the data. For cases where the State Machine is configured as two stream processors working together, the general-purpose processor facilitate the transfer of data from one stream processor to the other without interfering with the operation of either one.

The general-purpose processor employed is the PowerPC603, a superscalar processor with a 32-bit address space and instruction word width. The data bus can be configured in either a 32-bit or a 64-bit mode. The State Machine uses the 32-bit mode because of board area limitations. The FPGAs used are Altera FLEX EPF81188 devices. The State Machine uses two of these devices, one for each stream port. These devices provide approximately 12,000 usable logic gates organized in 1008 logic elements and 180 I/O elements.[22]. These are sufficient programmable logic

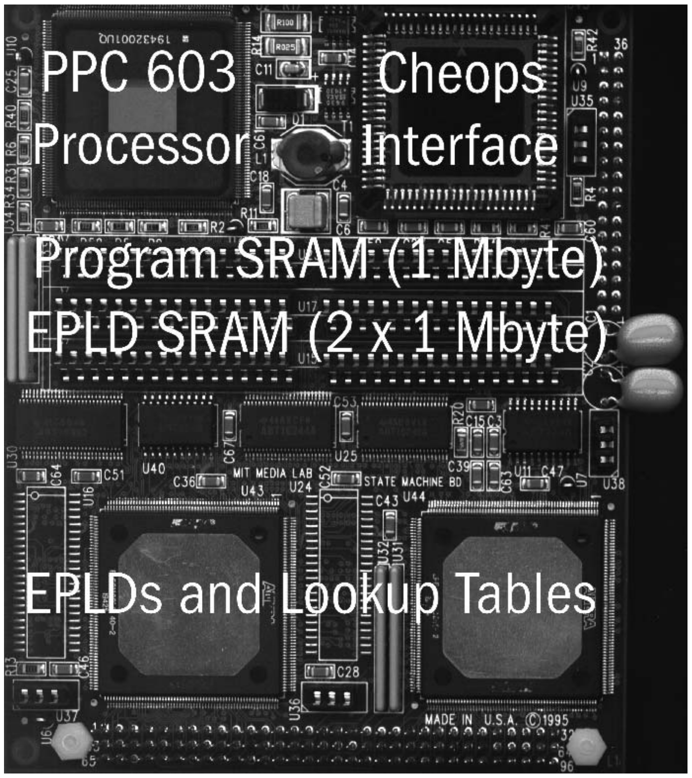

Figure 4: The State Machine stream processor, 75% of actual size.

resources to implement small- to medium-sized computational architectures. Each device has access to a 256K $\times$ 32 data memory as well as a 64K $\times$ 16 lookup table (LUT) memory. In addition, each device has a direct interface to the crosspoint. That is, data coming from the VRAM memory banks on the processor module flow directly into the FPGAs on the State Machine submodule.

The transfer registers that connect the PC603 buses to the FPGA buses consist of ABT type logic devices and two PAL devices. Because the burst modes of the PC603 processor are supported, the lower bits of the address buses must have the ability to count, and are thus housed in a 22v10 PAL. The second PAL holds control logic necessary to facilitate the bus transfers.

After allowing space for the two FPGAs and other system components, enough room remained to fit three 32-bit wide SIMM modules of fast static RAM. One SIMM is dedicated to each of the processing elements, local to each bus.

5. SOFTWARE SUPPORT

A State Machine application is divided into two components, an architectural description of the custom hardware to be implemented in the FPGAs, and a software image to be executed on the processor. Every application has these two components irrespective of device utilization. Applications that can be implemented entirely in hardware still need a software image to handle communications and transfer phase control information with the LP. Conversely, applications that carry out the entire application with the general-purpose processor still need hardware descriptions of the circuitry required to transfer the incoming data from the stream interface to the memory banks.

The State Machine is configured by the 80960 through the register interface. This interface consists of a 16-bit address bus and an 8-bit data bus. State Machine configuration consists of two phases. In the first, the 80960 loads both the software image and the device configuration files into the State Machine processor memory bank. In the second, the PC603 initializes the application code and reconfigures the FPGA devices with the new configuration data. The two parts may occur sequentially, or may be decoupled in time.

State Machine applications for which the PC603 plays more than a trivial role typically require some form of communication between the PC603 and the FPGA devices. This communication may occur through semaphore locations in the data SRAM, register locations on the FPGAs, or interrupts generated by the FPGAs.

The simple operating system we have developed for the PC603, called ChameleonOS, is relatively small (less than 32Kbytes). Thus even after we set aside memory space for event logging (necessary in debugging because the PC603 can't print directly to the system console), some 896Kbytes of the program SRAM is available. This is allocated in 128Kbyte slots, each of which can hold a code segment for the PC603 or two different configurations for each of the FPGAs. Since loading code or configuration information by the 80960 into this memory is slowed by the 8-bit register bus, we intend for the resource manager to use the SRAM as a cache area so that (up to the limits of the SRAM) it can load each configuration only once, and simply signal the State Machine to assume a particular functionality as required by a forthcoming data transfer, using a special **start_proc**() routine and **hdwe_config** messages. As of the submission date of this paper, integration of State Machine management into the resource manager was not yet complete, and initial testing was being done with preloaded, static configurations of the hardware.

Reference (23) gives a much more detailed description of the hardware design and low-level software interface.

6. RESULTS

To date, only a few tasks have been compiled for and executed on the State Machine processor. Perhaps the most complex of these is an engine for generating optical flow vectors (dx, dy) from the affine equations

$$dx = Ax + By + C$$

$$dy = Dx + Ey + F.$$

Such a parametric description of motion is used in certain motion-segmented video coding algorithms;[24] here we provide through the register interface the six parameters A through F and the range of x and y, and using a Bresenham-style algorithm the processor generates an output stream with the (dx, dy) pair for each pixel, every two clock cycles. These vectors then go to a remapping processor along with the image data.

It appears that routing constraints of the Altera FLEX architecture and the compiler software do not permit efficient utilization of chip resources when the pinout is totally specified in advance (as required by this card). We have to date been unable to fit designs that require more than 46% of the available logic resources.

7. CONCLUSIONS/ACKNOWLEDGEMENTS

SRAM-based FPGAs have advanced to the point that they can serve as the basis for useful, compact reconfigurable processing elements. In order to be successful in this application, though, devices require two architectural features. First is rapid reconfigurability. Second, the device must provide routing optimized for using the chip resources when the pinout is fixed in advance.

The research described in this paper has been supported by the Television of Tomorrow Consortium at the MIT Media Laboratory.

8. REFERENCES

1. V. M. Bove, Jr., B. D. Granger, and J. A. Watlington, "Real-Time Decoding and Display of Structured Video," *Proc. IEEE ICMCS '94*, 1994, pp. 456-462.

2. Digital Equipment Corporation, *VAX Hardware Handbook*, Maynard MA, 1982.

3. M. Wazlowski and L. Agarwal, "PRISM-II compiler and architecture," *IEEE Workshop on FPGAs for Custom Computing Machines*, 1993, pp. 9-16.

4. A. K. Agerwala and T. G. Rauscher, *Foundations of Microprogramming Architecture, Software, and Applications*, Academic Press, New York, 1976.

5. J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Mateo CA, 1990.

6. S. Seshadri, "Polynomial evaluation instructions, a VAX/VMS assembly language instruction," *VAX Professional*, 10(2), 1988.

7. F. L. Williams and G. B. Steven, "How useful are complex instructions? A case study using the M68000," *Microprocessing & Microprogramming*, 29(4), 1990.

8. K. Ghose, "On the VLSI realization of complex instruction sets using RISC-like components," *Proceedings of VLSI and Computers. First International Conference on Computer Technology, Systems and Applications*, 1987.

9. R. C. Dorf. *The Electrical Engineering Handbook*, CRC Press, Boca Raton FL, 1993, pp. 1654-1656.

10. C. Iseli and E. Sanchez, "Beyond superscalar using FPGAs," *1993 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 1993, pp. 486-490.

11. I. Page, W. Luk, and V. Lok, "Hardware acceleration of divide-and-conquer paradigms: a case study," *IEEE Workshop on FPGAs for Custom Computing Machines*, 1993, pp. 192-201.

12. R. L. Haviland, G. J. Gent, and S. R. Smith, "An FPGA-based custom coprocessor for automatic image segmentation applications," *IEEE Workshop on FPGAs for Custom Computing Machines*, 1994, pp. 172-179.

13. C. P. Cowen and S. Monaghan, "A reconfigurable monte-carlo clustering processor," *IEEE Workshop on FPGAs for Custom Computing Machines*, 1994, pp. 59-65.

14. D. E. Van den Bout, *et al.*, "Anyboard: An FPGA-based, reconfigurable system," *IEEE Design & Test of Computers*, 1992, pp. 21-30.

15. D. E. Van den Bout, "The Anyboard: Programming and enhancements," *IEEE Workshop on FPGAs for Custom Computing Machines*, 1993, pp. 68-76.

16. J. M. Arnold, "The SPLASH 2 software environment," *IEEE Workshop on FPGAs for Custom Computing Machines*, 1993, pp. 88-93.

17. D. V. Pryor, M. R. Thistle, and N. Shirazi, "Text searching on SPLASH 2," *IEEE Workshop on FPGAs for Custom Computing Machines*, 1993, pp. 172-177.

18. S. Casselman, "Virtual computing and the virtual computer," *IEEE Workshop on FPGAs for Custom Computing Machines*, 1993, pp. 43-49.

19. L. Agarwal, M. Wazlowski, and S. Ghosh, "An asynchronous approach to efficient execution of parallel programs on adaptive architectures utilizing FPGAs," *IEEE Workshop on FPGAs for Custom Computing Machines*, 1994, pp. 101-110.

20. A. DeHon, "DPGA-coupled microprocessors: Commodity ICs for the early 21st century," Transit Note #100, MIT Artificial Intelligence Laboratory, Cambridge MA, January 1994.

21. V. M. Bove, Jr. and J. A. Watlington, "Cheops: A Reconfigurable Data-Flow System for Video Processing," *IEEE Transactions on Circuits and Systems for Video Processing, 5,* Apr. 1995, pp. 140-149.

22. Altera Corporation, *Altera FLEX 8000 Handbook*, May 1994.

23. E. K. Acosta, *A Programmable Processor for the Cheops Image Processing System*, SM Thesis, MIT, 1995.

24. T. Chang and V. M. Bove, Jr., "Experiments in Real-Time Decoding of Layered Video," *Integration Issues in Large Commercial Media Delivery Systems, 2615,* SPIE, 1995, (in press).