PipeRench: A Virtualized Programmable Datapath in 0.18 Micron Technology

Herman Schmit, David Whelihan, Andrew Tsai, Matthew Moe, Benjamin Levine, R. Reed Taylor
Department of Electrical and Computer Engineering
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{herman,whelihan,tsai,moe,blevine,rt2i}@ece.cmu.edu

Abstract - PipeRench is a programmable datapath that can be used to accelerate numerically intensive applications. The unique aspect of PipeRench is its ability to virtualize hardware through self-managed dynamic reconfiguration. This capability provides application portability and scalability without redesign or recompilation. This paper describes the implementation of PipeRench in a 0.18 micron process. The implementation has 3.65 million transistors and runs at 120MHz. Performance is competitive with high-end commercial DSP architectures and more than five times faster than a commercial microprocessor. Executing at 33MHz, a FIR filter without virtualization consumes 519mW. When virtualization is required, the implementation consumes approximately 675mW.

I. Introduction

A customized data path, with appropriate levels of parallelism and pipelining, is intrinsically more efficient than traditional software execution for many applications [1]. The primary impediment to the use of reconfigurable computing technologies, however, is the cost of developing and re-using such hardware pipelines. The PipeRench architecture addresses these problems by introducing a virtual hardware abstraction. Applications expressed in this hardware abstraction can be run on a family of compatible devices at different cost and performance points. This virtualization of hardware in PipeRench is accomplished by run-time reconfiguration of the programmable hardware fabric. Unlike other run-time reconfigurable devices, PipeRench manages its own reconfiguration without any host or user interaction.

This paper presents the results from an implementation of the PipeRench architecture in a 0.l8 micron process. In particular, we focus on the additional power and silicon area required to implement this dynamic reconfiguration.

II. Architecture and Virtualization

PipeRench supports virtualization of hardware pipelines with limited feedback. Transformations such as FFT, DCT, and many encryption routines do not require any feedback. The limited feedback supported by PipeRench is sufficient to support operations such as FIR filtering and convolution. The virtual hardware model, as well as the process of hardware virtualization is shown in Figure 1. The top of this figure shows a 6-stage (or stripe) virtual hardware pipeline. The bottom of this figure illustrates the first five cycles of reconfiguration of a four stage physical hardware pipeline executing

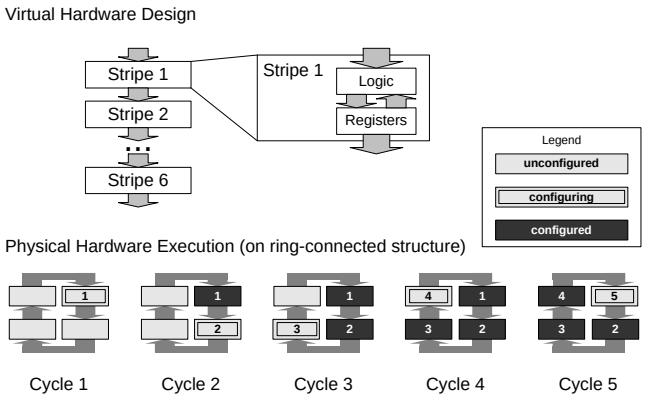

Fig. 1. Virtual hardware and execution through dynamic reconfiguration.

the six stage virtual design. Reconfiguration is performed by storing the configuration bits of the entire virtual hardware on chip and moving this information from the on-chip memory into the physical fabric every cycle. This process is described in more depth in [3].

In PipeRench, the functionality in each stripe consists of sixteen processing elements (PEs), which contains logic and registers. All PEs within a stripe are interconnected within that stripe, which facilitates easier placement and routing of operations.

Figure 2 is the block diagram of PipeRench's eight-bit PE. All select inputs to multiplexors and shifters in this figure are connected to configuration bits stored in that PE. Special purpose interconnects are used to combine adjacent PEs to perform operations more than eight bits in width. The shifters are also connected to the corresponding shifters in adjacent PEs to allow for efficient multi-PE shift operations.

Each PE contains a register file with eight registers. The output from the functional unit can be written to any one register in the register file in that PE. If the value from the functional unit is not written to a register, the value from the corresponding register in the previous stripe is clocked into that register.

If a stripe is the first in the pipeline, the input of the R0 register is connected to a global input bus. If a stripe is the last in the pipeline, the output of R0 is connected to the global output bus. The PE also connects to a dedicated horizontal interconnect line, which goes to other PEs in the stripe. This output can be programmably connected to the outputs of the

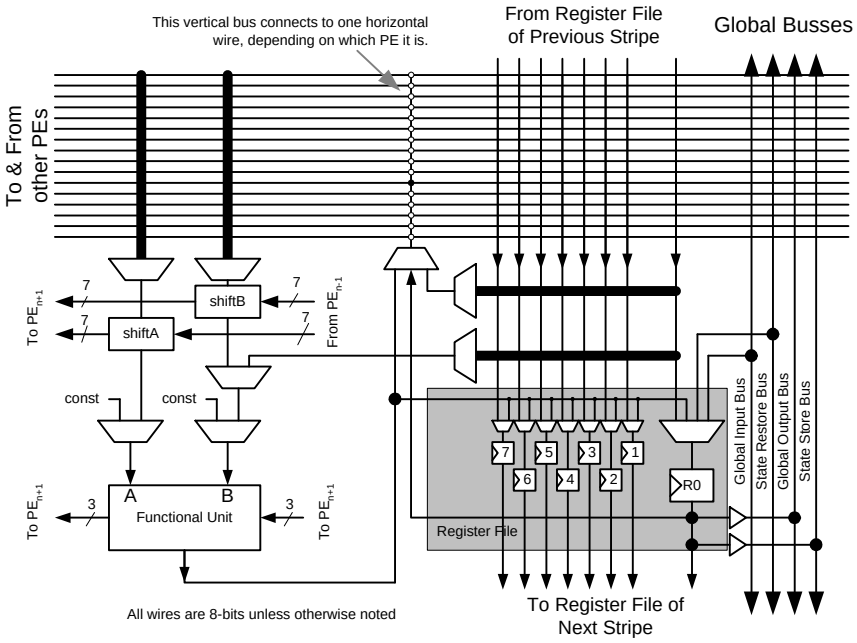

Fig. 2. PE block diagram

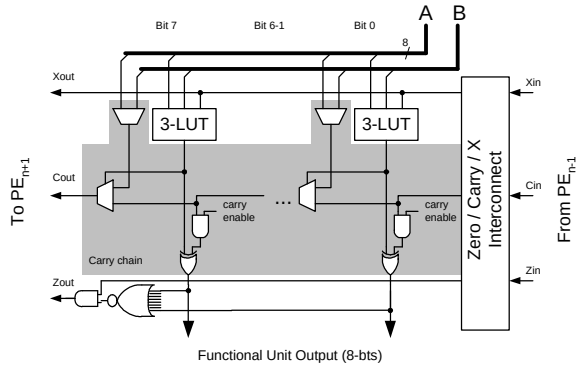

Fig. 3. Functional unit block diagram including LUTs, carry chain logic and zero detect circuitry.

previous stripe's register file, the output of that stripe's R0, or the output of the functional unit.

The functional unit in each PE consists of eight 3-input look-up tables (3-LUTs) that are identically configured (Figure 3). The single-bit X input connects to all eight LUTs, which is useful for implementing multiplexors or multiplier stages. To implement subtraction and addition functions, the functional unit includes a carry chain. The evaluation of output conditions is facilitated by the zero detector. All three signals, carry, zero, and X, can be programmably connected to form wider functional units. It is also possible to route the zero and carry outputs of one PE to the X input of the adjacent PE. This is useful for implementing selection logic after a comparison operation. The functionality of the PE is specified by 42 configuration bits, which means that each stripe

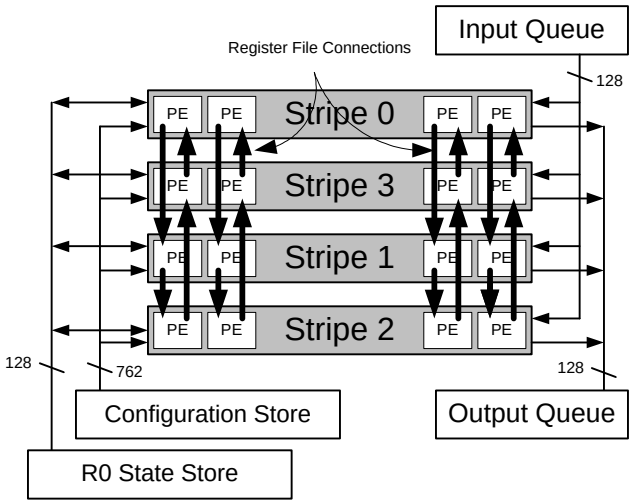

Fig. 4. Architecture and abstract layout of PipeRench with four stripes.

has 672 bits of configuration. The configuration for an entire application is compiled by our tools from a dataflow graph and stored in the on-chip configuration store [8].

As illustrated in Figure 4, the stripes are connected, through their register files, in an interleaved ring. The register file connections in this wire correspond to the register file wires entering the PE at the top of Figure 2 and exiting at the bottom of that figure. This layout implied by this figure is extended to 16 stripes in the current silicon implementation.

A stripe is configured by reading the configuration information from the on-chip configuration store in Figure 4 and

writing it into the configuration latch of any physical stripe. The process of configuring a stripe takes one cycle, so the pipeline can be configured one cycle before the first data of the pipeline arrives at that stage.

If the virtual hardware is larger than the real hardware, physical stripes will eventually be reconfigured with new virtual stripes. The state of the over-written virtual stripes are preserved by writing the value in R0 into the R0 state store memory, which is illustrated in Figure 4. The state will be restored when that virtual stripe is returned to the fabric. The process of reconfiguration continues until the last stripe in the virtual design is configured. After that, reconfiguration of a physical stripe starting with the first virtual stripe will occur and the computation proceeds with new inputs. Virtualization is efficient using this process. Only one stripe in the fabric is ever being reconfigured, while all other stripes are executing concurrently.

III. Hardware Interface

As illustrated in Figure 5, input and output interfaces include a 32-bit data bus and a valid and full bit. This interface is compatible with most FIFO memory devices. The preferred system implementation includes input and output FIFOs to deal with the burstiness of IO traffic caused by the virtualization of hardware. All control is facilitated by structures in the 32-bit data stream. The data stream is separated into packets, each consisting of a header that describes the destination and function of the packet, a marker that describes the length of the packet, and a payload. The payload can contain configuration data that specifies the virtual hardware design, user data to be executed on an already uploaded hardware design, or state information (the initial or final state of R0 in each stripe of the hardware design). The packet structure is described in depth in [4].

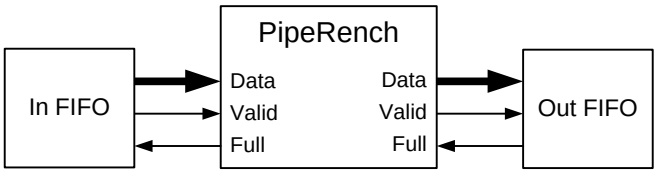

Fig. 5. FIFO interface for PipeRench

IV. Implementation

Students and faculty at CMU designed PipeRench, and ST Microelectronics fabricated it in a six-metal layer 0.18 micron CMOS process. The total die area is 7.3 x 7.6 mm^2. Transistor count is 3.65 million. The chip has 132 pins, which includes a 72-pin data interface, 5-bit test interface and 53 pins for power and ground. There are 3.3V and 1.8V supplies for the I/Os and core, respectively. The core area is divided into two

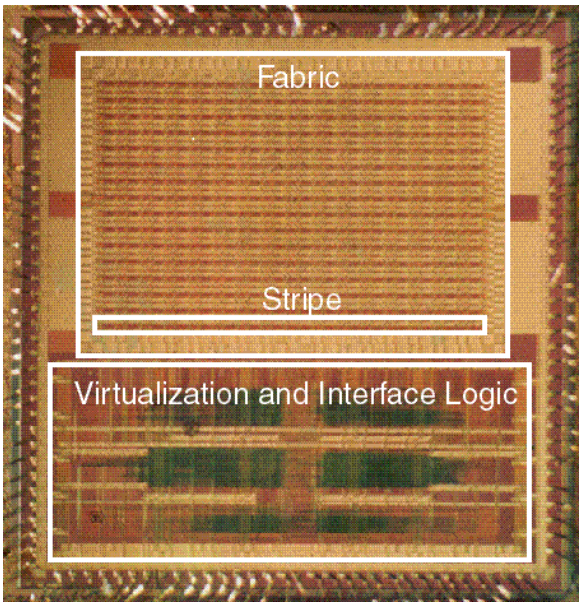

Fig. 6. Chip Micrograph of PipeRench

areas: (1) the fabric, and (2) the virtualization and interface logic. The fabric consists of sixteen stripes.

The virtualization and interface logic is implemented using standard cells. The configuration data is stored in 22 SRAMs, each with 256 32-bit words. Four 256 word by 32-bit dual-port SRAMs are used for storage of state information in the fabric. Sixteen dual-port SRAMs, (32 x 16 bits), are used to queue data between the interface and fabric. The virtualization storage and logic consumes less area than the 16 stripe fabric and stores 256 virtual stripes. This implementation can virtualize a hardware design that is sixteen times its own size.

The chip has two clock inputs: one clock controls the operation of the fabric and virtualization; the second clock controls the off-chip interface. These clocks are fully decoupled; all data transactions across the clock domains go through the memory queues between the interface and fabric and all control signals pass through a synchronizer. The fabric clock is designed to operate at 120 MHz under worst-case voltage and temperature conditions. The interface clock is designed to operate at 60 MHz.

The layout of a PE is shown in Figure 7 with the top four layers of metal transparent and with some basic components labeled. The dimensions of this cell are 325 µm by 225 µm. Sixteen of these PEs compose a stripe, and the fabric in Figure 6 contains 256 of these PEs. The PE area is dominated by interconnect resources such as multiplexors and bus drivers. The transistor density of this layout is not dense. The dimensions of the PE layout are dictated by the interconnect to other PEs in the stripe, and by the global busses, which run vertically over the PE cell.

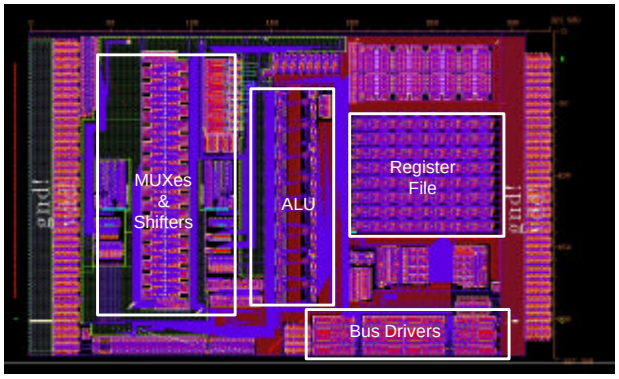

Fig. 7. PE floorplan

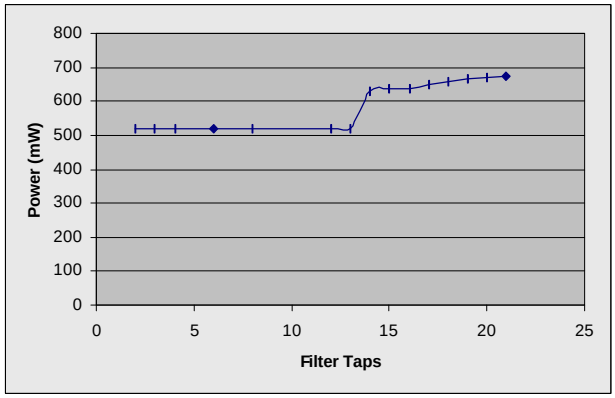

Fig. 8. Power versus filter size.

V. Performance

At 120 MHz, PipeRench executes a 40 tap 16-bit FIR filter at 41.8 MSPS, which means that its performance on mainstream DSP applications is in the same range as high end DSPs such as the Texas Instruments C64x family [7]. It does this at a much lower clock frequency than the high-end DSPs and without an actual multiplier within its fabric.

The comparative performance of PipeRench is even better on applications that are less multiplication intensive. On the IDEA encryption algorithm [6], PipeRench performs encryption or decryption at 450 Mbps. The encryption key is compiled directly into the hardware [5]. By comparison, a 800 MHz Pentium III processor executes this encryption algorithm at a rate of 75.4 Mbps.

These performance levels are similar to those that can be achieved by commercial FPGAs. Unlike FPGAs, however, these virtual pipelines, without modification, would run nearly twice as fast on a device with twice the number of physical stripes. In fact, even most commercial DSP cannot exploit additional parallelism without re-compilation of the application.

The power consumption of PipeRench has been measured using a 33.3MHz fabric clock and a 16.7MHz IO clock for a variety of FIR filter sizes[1]. These power results are shown in Figure 8. Power consumption is fairly level within certain regions. The power consumption significantly increases when the number of taps increases from 13 to 14. This is because the 14 taps filter requires 17 virtual stripes, which means that PipeRench must virtualize the hardware. The increase in power consumption is due to the operation of configuration and state memories that must now be accessed. Power consumption of small filters is not lower because the pass register file is not disabled in un-configured stripes. As a result, there is significant false switching in stripes that are not configured. This problem could be addressed by gating the clocks of these stripes. These results demonstrate that dynamic reconfiguration is possible with only a 30% increase in power dissipation.

1. The apparatus we used to measure power could only generate a 33.3MHz fabric clock and test vectors at 16.7MHz.

VI. Conclusions

This paper has discussed the implementation and characterization of the PipeRench architecture in a 0.18 micron process. The performance of the design is in the range of high-end DSPs and FPGAs for filtering, and exceeds the performance of a microprocessor on IDEA encryption by a factor of 5.96. Power dissipation is reasonable, even when dynamic reconfiguration is required by a large virtual hardware design.

Acknowledgments

This work was funded by DARPA ITO/TTO under contract DABT63-96-C-0083, the Pittsburgh Digital Greenhouse, MARCO/GSRC and ST Microelectronics. Herman Schmit is partially supported by an NSF CAREER grant.

References

[1] A. DeHon. “The Density Advantage of Configurable Computing.” *IEEE Computer*, 33(4):41-49, April 2000.

[2] S. Cadambi, J. Weener, S. C. Goldstein, H. Schmit, D. E. Thomas, “Managing Pipeline-Reconfigurable FPGAs,” *Proceedings ACM/SIGDA Sixth International Symposium on Field Programmable Gate Arrays*, p. 55-64, Feb. 1998.

[3] H. Schmit, “Incremental Reconfiguration for Pipelined Applications,” *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM)*, p. 47-55, 1997

[4] R. Laufer, R. R. Taylor, H. Schmit, “PCI-PipeRench and the SwordAPI: A System for Stream-based Reconfigurable Computing”, in *FCCM 99, 1999.*

[5] R. R. Taylor, S. C. Goldstein, “A High-Performance Flexible Architecture for Cryptography”, *CHES 1999*, published in the **Springer Verlag Lecture Notes in Computer Science (LNCS #1717)**.

[6] B. Schneier, **Applied Cryptography**, Wiley, New York, 1996.

[7] Texas Instruments, **TMS320C64x Technical Overview**, Literature Number: SPRU395, January 2001.

[8] M. Budiu, S. C. Goldstein, “Fast Compilation for Pipelined Reconfigurable Fabrics”, *Proceedings of the 1999 ACM/SIGDA Seventh International Symposium on Field Programmable Gate Arrays*, Feb. 1999.