1985

Experience with the CMU programmable systolic chip

Allan L. Fisher
Carnegie Mellon University

H. T. Kung

Kenneth Sarocky

CMU-CS-85-161

Experience with the CMU Programmable Systolic Chip

Allan L. Fisher, H. T. Kung, and Kenneth Sarocky

Department of Computer Science, Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Abstract

The CMU *programmable systolic chip* (PSC) is an experimental, microprogrammable chip designed for the efficient implementation of a variety of systolic arrays. The PSC has been designed, fabricated, and tested. The chip has about 25,000 transistors, uses 74 pins, and was fabricated through MOSIS, the DARPA silicon broker, using a 4 micron nMOS process. A modest demonstration system involving nine PSCs is currently running. Larger demonstrations are ready to be brought up when additional working chips are acquired.

The development of the PSC, from initial concept to a silicon layout, took slightly less than a year, but testing, fabrication, and system demonstration took an additional year. This paper reviews the PSC, describes the PSC demonstration system, and discusses some of the lessons learned from the PSC project.

Introduction

Using massive parallelism and pipelining, the systolic array concept[1] allows a system implementor to design extremely efficient machines for specific computations. But for some applications such as computer vision that call for hundreds of subroutines to be used routinely, it is impractical to produce a new systolic array processor for each subroutine. In this case, *programmable* systolic array processors must be used to provide the required flexibility.

However, to make a processor programmable takes additional hardware. This concern is especially significant for systolic arrays, as their performance relies on the use of large numbers of cells in the array. To be cost-effective, each cell should use as few chips as possible.

The purpose of the PSC project has been to study the feasibility and issues of implementing a cell (for a variety of systolic arrays) with one *single, programmable* chip, as depicted in Figure 1. A particular systolic cell can be implemented by microprogramming a PSC, and

(To appear in Microarchitecture of VLSI Computers, Martinus Nijhoff Publishers, 1985)

many PSCs can be connected at the board level to build systolic arrays of many different types and sizes.

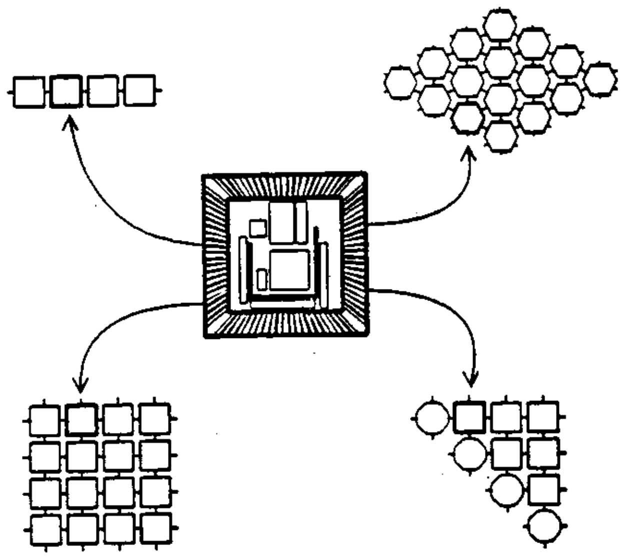


Figure 1. PSC: a building-block chip for a variety of systolic arrays.

The PSC is perhaps one of the first microprogrammable chips designed to be used in large groups. Besides being an architectural experiment, the PSC project also represents a major chip design experiment in a university environment. Prior to the PSC project, CMU had no experience in designing chips of this scale. The experience resulting from the project, with respect to both the architecture and design of the PSC, has been invaluable.

This paper reports some of these experiences, describes the current PSC demonstration system, and explains how the PSC implements the systolic array in the demonstration system to perform convolution or filtering operations. In the next section, we first give a brief overview of the PSC. Detailed descriptions of the PSC architecture and design have been reported in other papers.[2, 3, 4]

PSC: A Programmable Systolic Chip

The PSC project started in October of 1981. In order to ensure sufficient flexibility to cover a broad range of applications and algorithms, we chose at that time an initial set of target applications for the PSC to support, including signal and image processing, error correcting codes, and disk sorting. The demands of these applications have resulted in the following design features:

- 3 eight-bit data input ports and 3 eight-bit data output ports.
- 3 one-bit control input ports and 3 one-bit control output ports.
- Eight-bit ALU with support for multiple precision and modulo 257 arithmetic.
- Multiplier-accumulator (MAC) with eight-bit operands and 16-bit accumulator.

- 64-word by 60-bit writable control store.
- 64-word by 9-bit register file.
- Three 9-bit on-chip buses.
- Stack-based microsequencer.

Note that no conventional, commercially available microprocessor components fulfill the needs of such a programmable systolic chip. Unlike the PSC, conventional microprocessors do not have fast, on-chip multiplier-accumulator circuits which are crucial for high-speed signal and image processing, they do not have enough off-chip I/O bandwidth and on-chip bus bandwidth to pass data from chip to chip with a speed sufficient to balance the computation speed, they are not equipped with I/O ports for passing "systolic control bits," they are not suited for the modular arithmetic needed in applications such as error-correction, and they usually do not have on-chip RAM for program memory. A number of more specialized processors having some of these features have appeared in the past several years, but none has all of them.

With optimized circuit and layout designs, the PSC should operate at a cycle time of no more than 200 ns, although the prototype PSCs we now have are found to be three to eight times slower. Reasons for this are discussed below. Assuming a 200 ns period, microcode examples indicate the following performances:

- A decoder of Reed-Solomon error-correction codes[5, 6] that can correct up to 16 erroneous bytes in 256-byte blocks can be implemented with 112 PSCs with a throughput of 8 Mbits/second. Encoding at the same rate can be achieved with only 32 chips. The fastest existing decoder of which we are aware operates at 1 Mbit/second and uses 500 chips.

- A digital filter (FIR or IIR) with eight-bit data and coefficients and k taps can be computed with k PSCs, taking one sample each 200 ns. For a 40 tap filter, this amounts to 400 million operations per second (MOPS), counting each inner product step (eight-bit multiply, 16-bit add) as two operations. This is equivalent to 600 MOPS for pure eight-bit arithmetic.

- For applications requiring more accuracy, a filter with 16-bit data and eight-bit coefficients and m taps can be computed with m PSCs, taking one sample each 1.2 μs. Thus with 40 PSCs, a 40 tap filter can be computed at a rate of 67 MOPS, counting each inner product step as two operations. This is equivalent to 200 MOPS for eight-bit arithmetic.

- A disk sorter implemented with 17 PSC chips and 16 Mbytes buffer memory can achieve an order of magnitude of speed-up over conventional minicomputers.

<u>Use of the PSC in Implementing Systolic Arrays for Convolutions</u>

The current demonstration system for the PSC performs two-dimensional (2-D) convolutions using general 3×3 kernels. In this section we describe briefly how the PSC is used to implement a systolic array for 2-D convolutions.

Mathematically, the 2-D convolution problem with a $k \times k$ kernel is defined as follows:

Given the 2-D kernel of weights w_{ij}, $i = 1, 2, \ldots, k$, $j = 1, 2, \ldots, p$, and the 2-D input image x_{ij}, $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$, with $k \ll m$ and $p \ll n$,

compute the output image y_{rs}, $r = 1, 2, \ldots, m-k+1$, $s = 1, 2, \ldots, n-p+1$, defined by

$$y_{rs} = \sum_{i=0}^{k-1} \sum_{j=0}^{p-1} w_{i+1,j+1} \cdot x_{i+r,j+s}.$$

The 2-D convolution problem is one of the most computation-intensive tasks in signal and image processing. For example, a 2-D convolution using a general 9×9 kernel requires 81 multiplications and 80 additions to generate each pixel in the output image.

1-D Convolution Implementation

We first illustrate a systolic array design for the one-dimensional (1-D) convolution problem, which is simpler than the 2-D one. The 1-D problem is defined as follows:

Given the sequence of weights $\{w_1, w_2, \ldots, w_k\}$, and the input sequence $\{x_1, x_2, \ldots, x_n\}$,

compute the result sequence $\{y_1, y_2, \ldots, y_{n+1-k}\}$ defined by

$$y_i = w_1 x_i + w_2 x_{i+1} + \cdots + w_k x_{i+k-1}.$$

Figure 2 depicts one of the well-known systolic arrays[1] for the case $k = 3$.

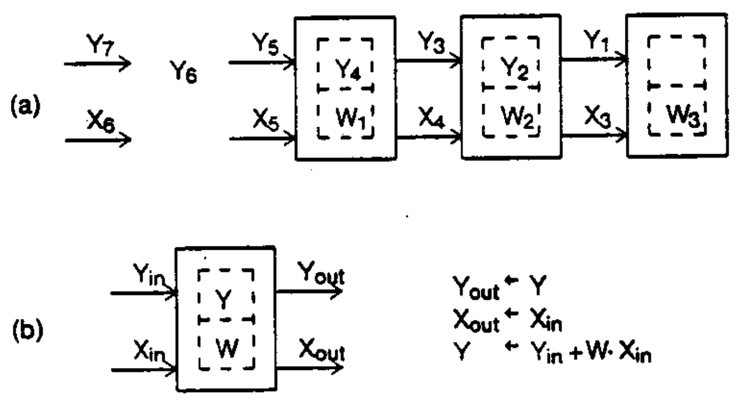


Figure 2. (a) Systolic array for 1-D convolutions, and (b) its cell definition.

We can program the PSC to implement each of the systolic cells; the program takes only one instruction to implement all the operations depicted in Figure 2(b). After an initialization phase in which the weights are loaded, the inner loop of the algorithm uses one PSC microinstruction, coded as follows:

```
Bus1=Sda, Bus2=Sdb, Bus3=Lo,
SdaOut=Val3, SdbOut=Val2,
MacX=Hold, MacY=Val2, MacZ=Val1, MacOp=AddZ,
Jump=OnCCO, CCO=Sca, ScaOut=Pass.
```

The lines of this microinstruction have the following effects, all in a single cycle:

1. Bus 1 carries Y_{in}, read from systolic data port A, bus 2 carries X_{in}, read from port B, and bus 3 carries Y_{out} from the previous operation, available as the output of the MAC.

2. Output port A receives Y_{out} from the previous operation (the value on bus 3), and port B receives $X_{out}=X_{in}$

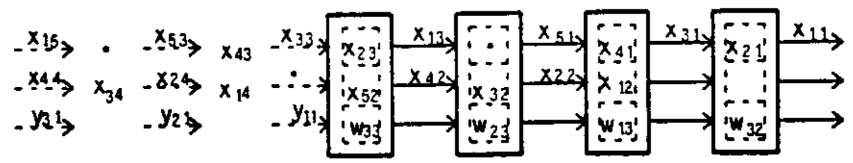
3. The MAC holds the cell's weight W in its x register, sets its y register to X_{in} (the value on bus 2), and sets its z register to Y_{in} (the value on bus 1). It then computes $x \cdot y + z$, or $W \cdot X_{in} + Y_{in}$. This value will be sent to output port A during the next cycle.

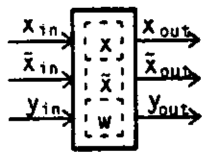
4. The instruction loops in place until systolic control signal A arrives, meaning it is time to reinitialize. Control then passes to the next instruction, and the control bit is sent on to the neighboring cell.

2-D Convolution Implementation

The above systolic array design for 1-D convolutions can be generalized to designs for 2-D convolutions. In particular, we will use a linear systolic array of k^2 cells to perform 2-D convolutions using $k \times k$ kernels. This systolic array will have the nice "scalable" property that its interface with the outside world is independent of k. That is, when the kernel size is increased, we need only expand the linear array accordingly, without changing its I/O interface. There exist at least two such "scalable" systolic array designs for the 2-D convolution problem, one requiring a memory associated with each systolic cell to buffer one line of image,[7] and the other one requiring no such memory for each cell.[8] For implementation simplicity we use the latter one.

The input image x_{ij} is fed to the systolic array in columns $2k-1$ pixels high, and the output image y_{ij} is generated by the systolic array in swaths which are k pixels high. As depicted in Figure 3, pixels from the input image enter the systolic array in two x-data streams—the x_{ij} with odd j come in with the top x-data stream, and the x_{ij} with even j come in with the bottom x-data stream. In each cycle, a cell will choose a value from one of the two x-data streams to multiply by the weight stored in that cell. The simple rule is that a cell should use values from one x-data stream for k consecutive cycles before switching to the other stream; and continue alternating in this manner. By utilizing the systolic control ports provided by the PSC, a control signal can be sent conveniently from cell to cell to signal the switching from one x-data stream to the other for each cell. It takes no more than two PSC instructions to implement all the cell operations depicted in Figure 3. This implies for example that 2-D convolutions with 9×9 kernels can be implemented by 81 linearly connected PSCs, capable of producing one output pixel every 400 ns assuming a state-of-the-art design.





$x_{out} := x; \quad x := x_{in}$

$\bar{x}_{out} := \bar{x}; \quad \bar{x} := \bar{x}_{in}$

$y_{out} := y_{in} + w \cdot x_{in}$ OR $y_{in} + w \cdot \bar{x}_{in}$.

Figure 3. Linear systolic array for 2-D convolutions and its cell definition.

For large kernels, it is necessary to save partial results y_i in double or higher precision to ensure numerical accuracy of the computed results. As for the case of systolic arrays for 1-D convolution,[1] this can be achieved most cost-effectively by using a dual of the design in Figure 3, where partial results y_i stay in cells but the x_i and w_{ij} move from cell to cell in the same direction but at two speeds. With this dual design, high-precision accumulation can be effectively achieved by the on-chip multiplier-accumulator circuit, and the number of bits to be transferred between cells is minimized. We of course still need to transfer the final computed values of the y_i out of the array, but they, being the final results, can be truncated and transferred in single precision. It is generally preferable to have the w_{ij} rather than the x_i going through an additional register in each cell, since there are two x-data streams but only one weight stream. In fact the communication cost for the weight stream can be totally eliminated if the register file of the PSC is large enough to hold a complete weight table.

Note that in generating adjacent output swaths, some input pixels are fed into the systolic array twice. To avoid having to bring these pixels out from memory twice, a cache that can hold $k-1$ lines of input pixels can be used, as shown in Figure 4.

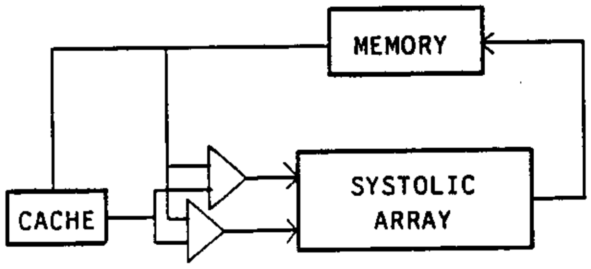


Figure 4. Use of cache to buffer lines from the input image.

PSC Demonstration System

Figure 5 depicts the current PSC demonstration system built around a SUN workstation. The system includes a PSC array board capable of holding 25 or more PSCs. As of May

1984, this board hosts a one-dimensional systolic array of nine PSCs, performing 2-D convolutions using 3×3 kernels on a video image of 512×512 8-bit pixels. Several 3×3 Gaussian and Laplacian kernels have been implemented for the demonstration. In the demonstration, 512×512×8 displays are processed at the rate of one display every 1.8 seconds. Using the "scalability" of the systolic array design as described in the previous section, the demonstration system can run at this speed *regardless* of the kernel size (assuming of course that there are as many PSCs in the PSC array board as the kernel size). For instance, with 25 PSCs the demonstration system can perform 2-D convolutions using 5×5 kernels still in the same 1.8 seconds. Indeed, it is our plan to do such a demonstration, as soon as enough working chips are acquired.

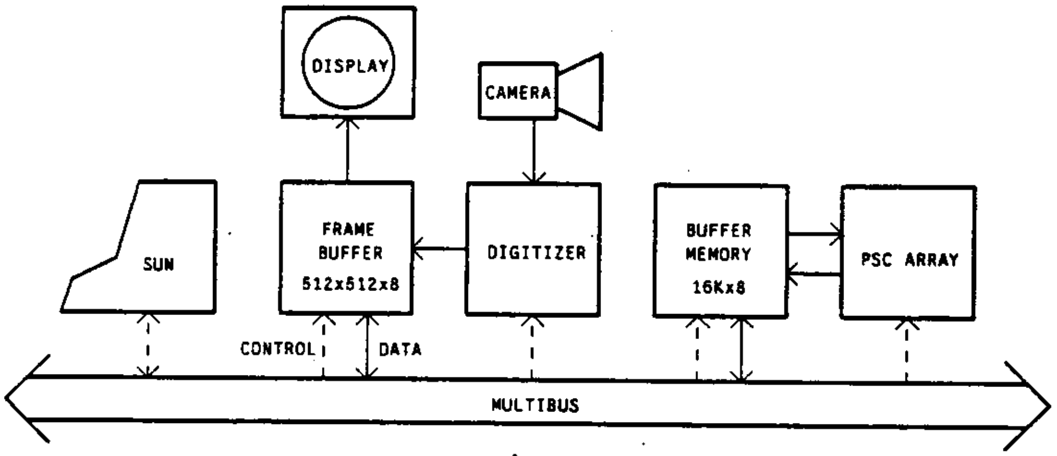


Figure 5. PSC demonstration system.

The host system for the demonstration is the SUN workstation that controls the PSC array board through its MULTIBUS interface. A buffer memory board buffers data for the PSC array board. Matrox graphics boards are used to perform the frame buffering and video A-to-D, D-to-A functions. The limited bandwidth of the Matrox DMA imposes one of the speed limits for the demonstration system. Besides the PSCs themselves, the PSC array board also contains a finite state controller, clock drivers, microcode loading circuitry, and address generation circuitry for the buffer memory board.

The demonstration system operates as follows:

1. The SUN's 68010 processor initializes the PSC array by writing to its control registers, which are mapped into the Multibus address space.

2. The 68010 loads the microcode of each PSC individually.

3. The 68010 initializes the Matrox graphics boards.

4. The Matrox VAF-512 board grabs one frame of video data from the camera, and loads it into the frame buffer.

5. The 68010 initiates a DMA transfer of five video lines from the frame buffer to the PSC buffer memory.

6. When above transfer is complete, the 68010 starts the PSC array, at a buffer address which it has supplied.

7. The PSC array reads in five lines of data, and produces three lines of output, writing it into the buffer. The 68010, meanwhile, initiates a DMA transfer of the next three lines of data into the buffer, which takes place in parallel with the PSC processing.

8. The 68010 periodically checks the status of the PSC array, and, upon sensing the DONE flag, restarts the array at the next buffer location. It then begins a DMA transfer of output data into the Matrox frame buffer. When the output transfer is completed, the 68010 initiates another DMA of input data to the PSC buffer, and waits for the PSC DONE flag.

Hindsights

The design of the PSC is a moderately large project by university standards. As of August 1984, the total effort has been about 4 man-years, with the following rough breakdown: architecture (1), logic and circuit design (.5), layout (.7), testing (.5), demonstration system (.8), and tool development (.5).

As often occurs in large, experimental system projects, the nature and demands of the PSC project did not become clear to us until the project was more than half-way through. We are pleased that the chip works and a modest demonstration system is running. At the same time, it is disappointing that up to now we have not been able to run large demonstrations, or to do experiments with many applications such as the decoder implementation for Reed-Solomon error-correction codes.

The PSC has not been applied on a large scale for two basic reasons. First, fabrication yield of the PSC has been low and has varied substantially over different MOSIS runs, and as a result it is difficult to predict when we will have a large number of working chips. Second, as the first of its kind, the PSC has no sophisticated software and interface support. This has prevented substantial applications from being brought up on the PSC at this time.

There were also many problems encountered in building, testing, and demonstrating the PSC. Some of these problems are inherent to the fact that the PSC project is experimental, and that the resources available to the project have been severely limited. (As far as we can tell, the cost of the PSC project is no more than one hundredth of the development cost of a typical commercial microprocessor!) However, there are a number of things that we would definitely do differently next time. In the following we discuss some of these hindsights under three categories: architecture, design, and management.

Architecture

The PSC architecture seems to be very well-suited to the implementation of a wide variety of systolic algorithms. The 64 instruction words available accommodate most straightforward computations, and suffice even for such complex cell computations as those found in the systolic Reed-Solomon decoder. The processor's control structure imposes very little overhead relative to the arithmetic heart of the computation; instruction fetch occurs in parallel with execution. Concurrency and parallel I/O result in much smaller instruction counts than for conventional microprocessors. Finally, the incorporation of control and communication circuitry onto the same chip as the arithmetic units achieves a large savings in chip count over systems built with standard parts. A PSC equivalent built with LSI arithmetic, memory and control and with TTL latches and multiplexers would require on the order of one hundred chips, and a similarly constructed single-purpose cell for even a slightly complicated algorithm would require a dozen chips.

However, the current PSC architecture is not completely optimized, partially for reasons of simplicity and flexibility that were considered important in view of the experimental nature of the PSC project. As our insights into the nature of systolic computation have increased, we have found several improvements that can be made. Some of the improvements described below have already been incorporated into the design of the CMU Warp processor, now under development.[9, 10]

For simple computations, the PSC's arithmetic, internal communication and external communication capacities are fairly well balanced. For more complex algorithms, the most common limiting factor in program performance is the number of buses. Adding more buses would be quite expensive in area needed for routing and for code storage and distribution; for a general-purpose part, this expense would probably not be justified. Other possibilities would be to use some specialized buses, to allow a single bus to be broken into independent parts, or to multiplex the use of the buses within a machine cycle. One other option, which seems to be very useful in the frequent case where a value needs to be delayed as it enters or leaves a cell, would be to put small programmable delays on the chip's input or output ports.

Another limitation is in the bandwidth of the register file. Only one word can be read or written in a cycle, making the storage and retrieval of intermediate results time-consuming compared to computation. Possible improvements include the use of multiported registers.

For large filtering problems where large numbers of terms may be accumulated at each cell, the multiplier-accumulator needs a high-precision accumulator. Preferably, the width of the accumulator should be at least 24 bits.

One way to reduce the cost of a PSC-based system would be to reduce the chip's complement of I/O pins, 54 of which are dedicated to the data ports. The PSC's use of three input and three output ports is due mainly to simplicity considerations: almost all systolic algorithms' data flows can be implemented in a straightforward way with a minimum of control. Since all six ports are needed simultaneously only for rather simple algorithms where communication dominates computation by a large factor, it may be possible to reduce the pincount of the chip without greatly reducing its overall performance. This could be achieved by multiplexing bidirectional ports (perhaps four), at a modest cost in control complexity.

Another area where the cost of the PSC might be reduced is in microcode space. Again for reasons of simplicity and flexibility, no attempt was made to squeeze the microinstruction size by limiting the number and kind of operations the PSC's parallel functional units could perform. While this is a useful property for an experimental system, it would be advantageous for production chips to sacrifice some flexibility for higher yield (due to smaller size) or more words of data or instruction memory.

Design

As mentioned earlier, the PSC project was the first major chip design effort at CMU. The actual process of bringing the PSC from its initial architectural concept to the current demonstration system has been a great learning experience. Part of the teaching was done by some serious technical difficulties, which were mostly related to chip operating tolerances (clock waveform and supply voltage sensitivity), yield and programming.

Electrical design is probably the weakest point in the PSC design. The memories, especially, have been less than robust over voltage, temperature, and clock waveforms. The yield problem has been mostly due to failures in dynamic RAM.

A related problem is the complex timing scheme used, which necessitated many off-chip clock signals, making speed testing difficult. The complexity of the memory timing scheme resulted in several patches being applied; it would have been better to clean it up.

A 700 ns cycle time has been observed for some PSC prototypes, but many of the prototypes have been found to run around 1.2 μs. One reason that the chip is not as fast as possible is that performance tuning of the layout was never done, for example, for the multiplier-accumulator circuit; speed was not one of the project's primary goals, and timing analysis tools like Berkeley's Crystal[11] and Stanford's TV[12] were not available. Another contributor has been a drift in MOSIS circuit parameters. The chip was designed under the assumption that diffusion resistance was 10 ohms/square, as in Mead/Conway (and as assumed, by default, by Crystal). A number of MOSIS runs have had resistances of 11 or 12 ohms/square. Under this assumption, the microcode bits have an estimated maximum delay of 50 ns. Recent MOSIS runs have had a diffusion resistance of 40 ohms/square, increasing that figure to 200 ns.

Since the available simulators capable of handling large numbers of transistors were not capable of handling the memory and some other features of the chip, full-chip layout simulation was never done. All of the pieces were tested and/or simulated, but test results in some cases were not available until after the entire chip had been sent off. At the time the chip was assembled, the only means available of checking connectivity of the parts was manual inspection of a 60 page condensed wirelist. This process caught one or two bugs, but one bug slipped by. Chip testing, though, was fairly straightforward: except for the sequencer, which could be tested only if the memory worked, everything on the chip was accessible over the buses.

A lack of good documentation is one reason the design was difficult to change. Another, bigger reason was the difficulty of routing. The PSC contains a lot of square microns of wire, painfully drawn by hand. Turnaround time was another factor discouraging major changes.

The chip had only three layout bugs, which were corrected early on; one was in the memory, one in one set of ALU registers, and one in the multiplier condition code (hard to notice, since the condition codes are very obscure). There was one logic bug (discounting the memory's problems, which were mostly electrical); it related to the timing of the sequencer's stack, was generated by a change in memory timing and was caught very late (summer '83) because of the dependence of circuit testing on the memory and because of a lapse in testing effort between March and June.

Many of these difficulties could be avoided next time by using appropriate design methodologies, conservative design styles, and new design tools. Some of the improvements that we can make seem to be rather "common sense" and obvious now. These include designing for worst-case technology and avoiding complex timing schemes. Given our multi-source fabrication through MOSIS, we must be prepared to deal with a relatively large process variation. Specifically, clever but risky circuit design should not be used. Mead and Conway simplified design rules exist, in part, for this purpose. The example of diffusion resistance mentioned earlier shows that pessimistic assumptions are important in the electrical domain, as well. Complex timing should be avoided because it will make the design difficult to understand, hard to change, and hard to deal with when testing and interfacing to the chip. The gain in performance through complex timing is hardly worth its cost.

As mentioned earlier, new timing aids like Crystal and TV will help remove critical timing paths. Using modern design workstations, such as Daisy and Mentor, schematics of the chip can be fully simulated and documented at the logic level with reasonable effort. Also, these workstations can generate netlists to be compared with those extracted from the layout, using the CMU wirelist comparison program, Gemini.[13]

A careful floorplan can also make a lot of difference in the quality of the design. Since useful global routers are still not available, a good floorplan can make the routing problem much easier. The floorplan should be drafted in the very beginning of the design, and constantly updated during the design as detailed layout information becomes available. From the floorplan one can tell if a certain design optimization is worthwhile. Also, the availability of the updated floorplan can substantially help the communication between a team of designers.

Management

Building a prototype research chip in a university environment is very different from the same task in an industrial environment, in the sense that we are severely limited by resources. It would be unreasonable to expect that universities have the same level of design support and engineering skill as semiconductor industry. Graduate students should be involved but shouldn't be expected to grow old in the course of a project. Probably one of the biggest lessons we have learned from the PSC project is a true appreciation of this limitation in resources.

But prototype chips, built by universities or not, must work in a system, otherwise there would be very little value in the prototyping. Therefore it is important that we do only those designs which are within the power of the available people and tools. Designs that industry does well and that require great skill and experience should be avoided by universities. Fancy dynamic RAM is an example.

An overall plan for simulation, verification, documentation, testing, and demonstration should be developed at the very beginning of a project. We must see to it that there is a very good chance that the chip will work at reasonable speed in a system for its first silicon, and that system demonstrations can be brought up without an excessive amount of effort. A related issue is the yield problem. We must learn (at last!) to trade architectural features for reduced die sizes in order to increase yield. These steps are necessary conditions for a large scale chip to be built successfully and smoothly. It is useful to remind ourselves that university prototype chips are not exempt from that.

The lack of sufficient system support mentioned earlier is a common problem for any prototype chips. One way to deal with it is to stage research programs so that software and interface support can be developed first on systems built with off-the-shelf components. Ideally, prototype custom chips should be built only after system support has already been developed.

Conclusions

Despite the problems discussed above, the PSC project produced a number of positive results. The chip works, albeit not as robustly as we would like. From the architectural point of view, the project demonstrated the "scalability" of systolic array design in the demonstration system, proved the feasibility of having a programmable "building-block" chip for the implementation of systolic algorithms and, through setting a concrete benchmark on which to base improvements, set the stage and provided initial ideas for further work. A natural step to follow is the development of an industrial version of the PSC. Several companies have expressed their interests in this. In theory, companies who produce PSC-like chips should be able to sell hundreds of copies of the chip to each customer, to form large systolic-like arrays!

The PSC project did not contribute to the low-level chip design knowledge of the world at large, but we learned a lot of things locally about chip design, both personally and in terms of the VLSI community at CMU. This includes not only the lessons mentioned above, but also the use of new tools and methods. The PSC experience has had profound impacts on the ways in which how some new CMU chips are being designed, as suggested in the preceding section.

The PSC is one of the first major chips made through MOSIS to have been integrated into a system. Work on the PSC also helped the MOSIS community gain experience in packaging, testing, and medium-volume production.

Acknowledgments

The PSC is a result of a team effort: its architecture and design have been reported in separate papers,[2, 3, 4] on which some of the material of this paper is based. F. H. Hsu wrote the PSC image processing code for the demonstration system. Some chip testing software was developed by Monica Lam. The PSC research was supported in part by the Defense Advanced Research Projects Agency (DoD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory under Contract F33615-81-K-1539. The fabrication of the PSC has been done through MOSIS, the DARPA silicon broker.[14]

References

1. Kung, H.T., "Why Systolic Architectures?," *Computer Magazine*, Vol. 15, No. 1, January 1982, pp. 37-46.

2. Fisher, A.L., Kung, H.T., Monier, L.M. and Dohi, Y., "The Architecture of a

Programmable Systolic Chip," *Journal of VLSI and Computer Systems*, Vol. 1, No. 2, 1984, pp. 153-169. An earlier version appears in *Conference Proceedings of the 10th Annual Symposium on Computer Architecture*, Stockholm, Sweden, June 1983, pp. 48-53.

3. Fisher, A.L., Kung, H.T., Monier, L.M., Walker, H. and Dohi, Y., "Design of the PSC: A Programmable Systolic Chip," *Proceedings of the Third Caltech Conference on Very Large Scale Integration*, Bryant, R., ed., Computer Science Press, Inc., California Institute of Technology, March 1983, pp. 287-302.

4. Walker, H., "The Control Store and Register File Design of the Programmable Systolic Chip," Tech. report CMU-CS-83-133, Carnegie-Mellon University, Computer Science Department, May 1983.

5. MacWilliams, F.J. and Sloane, N.J.A., *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, Holland, 1977.

6. Peterson, W.W. and Weldon, E.J., Jr., *Error-Correcting Codes*, MIT Press, Cambridge, Massachusetts, 1972.

7. Kung, H.T., Ruane, L.M., and Yen, D.W.L., "Two-Level Pipelined Systolic Array for Multidimensional Convolution," *Image and Vision Computing*, Vol. 1, No. 1, February 1983, pp. 30-36. An improved version appears as a CMU Computer Science Department technical report, November 1982.

8. Kung, H.T. and Picard, R.L., "One-Dimensional Systolic Arrays for Multidimensional Convolution and Resampling," *VLSI for Pattern Recognition and Image Processing*, Fu, King-sun, ed., Spring-Verlag, 1984, pp. 9-24. A preliminary version, "Hardware Pipelines for Multi-Dimensional Convolution and Resampling," appears in *Proceedings of the 1981 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Hot Springs, Virginia, November 1981, pp. 237-278.

9. Kung, H.T., "Systolic Algorithms for the CMU Warp Processor," *Proceedings of the Seventh International Conference on Pattern Recognition*, International Association for Pattern Recognition, 1984, pp. 570-577.

10. Kung, H.T. and Menzilcioglu, O., "Warp: A Programmable Systolic Array Processor," *Proceedings of SPIE Symposium, Vol. 495, Real-Time Signal Processing VII*, Society of Photo-Optical Instrumentation Engineers, August 1984.

11. Ousterhout, J. K., "Crystal: A Timing Analyzer for nMOS VLSI Circuits," *Proceedings of the Third Caltech Conference on Very Large Scale Integration*, Bryant, R., ed., Computer Science Press, Inc., California Institute of Technology, March 1983, pp. 57-70.

12. Jouppi, N.P., "TV: An nMOS Timing Analyzer," *Proceedings of the Third Caltech Conference on Very Large Scale Integration*, Bryant, R., ed., Computer Science Press, Inc., California Institute of Technology, March 1983, pp. 71-86.

13. Ebeling, C. and Zajicek, O., "Validating VLSI Circuit Layout by Wirelist Comparison," *Proceedings of 1983 IEEE International Conference on Computer-Aided Design*, IEEE, September 1983, pp. 172-173.

14. Lewicki, G., Cohen, D., Losleben, P. and Trotter, D., "MOSIS: Present and Future," *Proceedings of Conference on Advanced Research in VLSI*, Penfield, P. Jr., ed., Artech House, Inc., Massachusetts Institute of Technology, Dedham, Massachusetts, January 1984, pp. 124-128.