Structuring a VLSI System Architecture

James H. Clark, *Stanford University*

Few of the very high performance integrated circuits being produced today are nMOS circuits. On the other hand, nMOS is almost synonymous with Very Large Scale Integration(VLSI). This article describes a graphics subsystem that illustrates that by employing architectures with a very high degree of parallelism, one can get high performance with nMOS. In fact, at the ultimate nMOS dimensions, the system's performance improves to a point that it is comparable with the fastest of super-computer technologies.

The system is designed to perform three of the very common geometric functions of computer graphics. A single chip type is used in 12 slightly different configurations to accomplish 4x4 matrix multiplications; line, character, and polygon clipping; and scaling of the clipped results to display device coordinates. This chip is referred to as the Geometry Engine.

The Geometry Engine is a four-component vector function unit that allows simple operations on floating-point numbers. It is composed of four identical function units, each one of which has an 8-bit characteristic and (presently) a 20-bit mantissa. It accomplishes the operations with a very simple structure that consists of an ALU, three registers, and a stack. This basic unit can do parallel adds, subtracts, and other similar two-variable operations on either the mantissa or the characteristic; since one of the registers can shift down and one can shift up, it can also do multiplies and divides at the rate of one multiply or divide step per microcycle. The 12-chip system consists of 1,344 copies of a single bit-slice layout that is composed of these five elements. Four pins on the chip are wired to tell its microcode which of the 12 functions it is to do, according to its position in the subsystem organization.

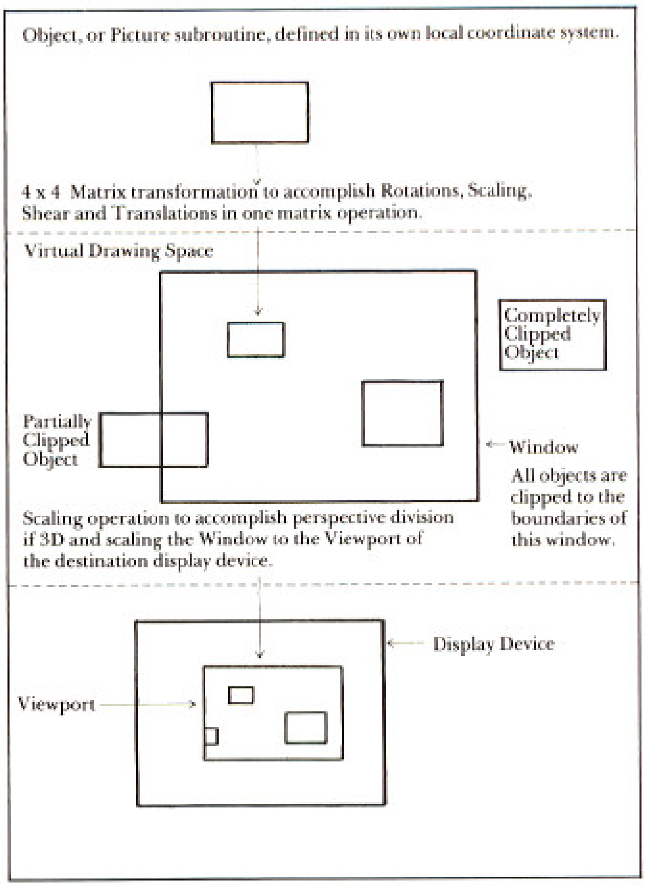
Key to the design has been the use of design techniques advocated by *Introduction to VLSI Systems* (Mead and Conway 1980). Since this book was the author's first exposure to nMOS circuit design, it has had a significant influence on the methodology used in arriving at the architecture. The regular structures advocated in this book are evident at several levels of the organization, and the timing methodology advocated by Seitz in chapter 7 is fundamental to the system.

After briefly discussing the graphics functions accomplished by the system, we will present a bit of the history of its development, with special emphasis on the architecture/design considerations motivated by the use of nMOS. Some of the reasons for choosing a self-timed convention will also be considered. Finally we will draw some conclusions about the feasibility of one person (or a small number of people) doing the architecture, circuit design, and layout of a VLSI system.

What Does the System Do?

There are three geometric operations that almost every graphics system must do. Figure 1 illustrates these three functions. The first one is transformations. Typically, "objects", or picture subroutines, are defined using such

FIGURE 1. Three basic operations performed by a graphics system: transformation, clipping, and scaling.


Object, or Picture subroutine, defined in its own local coordinate system.
4 x 4 Matrix transformation to accomplish Rotations, Scaling, Shear and Translations in one matrix operation.
Virtual Drawing Space
Completely Clipped Object
Partially Clipped Object
Window
All objects are clipped to the boundaries of this window.
Scaling operation to accomplish perspective division if 3D and scaling the Window to the Viewport of the destination display device.
Display Device
Viewport

primitives as lines, characters, or polygons. They are usually defined in their own local frame of reference, or coordinate system, since this makes their definition more general, *i.e.*, their coordinates can be entered without regard to how they might eventually be drawn on the destination display. Thus, the first thing that happens to the objects on their way from object memory to the display is a transformation of their coordinates, usually into an intermediate coordinate system that is convenent for clipping.

A convenient way to accomplish this transformation is with a 4x4 matrix. The motivation for this choice is treated very well in the excellent graphics textbook *Introduction to Computer Graphics*, by Newman and Sproull. A 4x4 matrix allows rotations, scaling, translations, and a number of other transformations to be done with a single entity. It also conveniently handles both 2- and 3-dimensional objects, and two successive transformations can be composed by a simple multiplication of their transformation matrices. Four Geometry Engines can transform an incoming four-component vector into a four-component vector expressed relative to a coordinate system that is "fixed to the observer." Figure 2 shows a block diagram of the subsystem organization; each of the four *mm* blocks is a separate Geometry Engine chip that does a four-component vector dot product. Although each multiply is done at the rate of one partial product per microcycle, the matrix multiplier has 16 of these products simultaneously active. Thus the total transformation time, which is the bandwidth limiting operation of the system, is about 12 microseconds.

After transformation, the objects are expressed relative to a virtual drawing coordinate system in which the viewing window is specified. In two dimensions, this window is as shown in Figure 1. In three dimensions, the window is a set of planes that extend outward from the eye of the observer; these planes can be thought of as extending through the rectangle defined by the 2-D boundaries shown in the figure, *i.e.*, the observer sees the planes on edge. This window represents the part of the picture that is to be mapped onto the destination display. Objects, that is points, lines, characters, and polygons (sequences of lines), are "clipped" to the planes, or window boundaries, one plane at a time in pipeline fashion, as indicated Figure 2 by the *clip n* units.

In the clipper configuration, a Geometry Engine holds the plane equation for both endpoints of the line in one of its function units, while it holds the 2- or 3-D coordinates for the endpoints in its other three units. The sign of the plane equation for each point tells which side of the plane the point is on. By convention, this sign is chosen to be negative if the point is on the "out" side of the plane, meaning that it is not visible to the observer because it is outside of the window. Thus a line segment having both signs negative is completely out of the viewing area, while one having both signs positive is completely in the viewing area. The function of the clipper is to discard segments that are completely outside its plane, pass segments that are completely inside to the next clipper chip, or find the point of intersection of the segment with the plane and pass the results on to the next clipper chip.

In the clipper configuration, the Geometry Engine first determines if a line is to be discarded or passed on to the next unit. Since these two cases are the most common, a clipper chip typically will spend very little time "clipping." Thus one might say that the clipper chip spends most of its time "sailing." When a segment crosses a plane, the engine finds the intersection point

FIGURE 2. Graphics geometry-subsystem data flow.

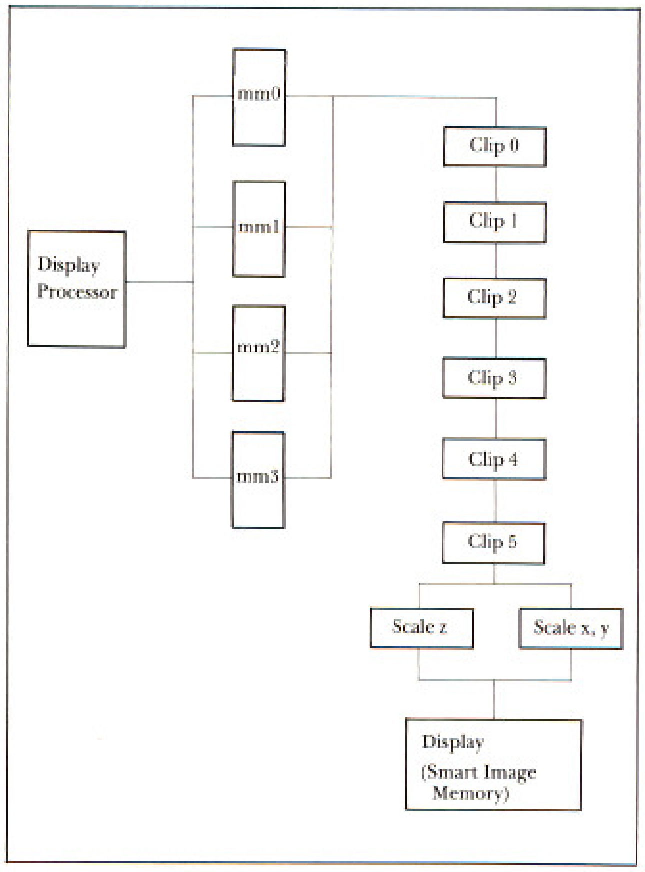
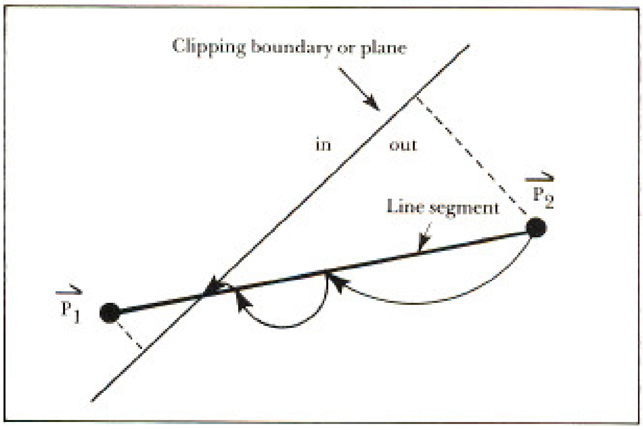


FIGURE 3. Logarithmic search for intersection point.



by a logarithmic search for the point (Figure 3). That is, on each microcycle each of the four units (the plane equation unit and the three coordinate units) computes the midpoint of its representation of the line. Choosing one of the endpoints as a reference point, it then determines if the midpoint is on the same side of the plane as this point. If so, then the reference point is updated by moving it to the midpoint. This is repeated until the precision of the mantissa is reached. The algorithm is very similar to that devised by Sproull and Sutherland (Sproull and Sutherland 1968).

The final function to be accomplished is scaling, which is indicated in the bottom part of Figure 1. This scaling operation scales objects remaining after clipping from the window to a viewport, which is expressed in the integer coordinate system of the destination display. In three dimensions both perspective division and scaling to destination device coordinates simultaneously take place. In two dimensions only scaling is done. The perspective division/scaling operation requires two function units for each coordinate; thus one geometry engine works for two coordinates, since it has four function units. The operation is done by using one of the units to accomplish the division by arriving at the successive bits of the quotient one at a time, starting with the most significant bit. Rather than accumulating them, however, they are instead used immediately to determine whether to add the successive partial products of the viewport scale using the other function unit. For example in the equation

$$X = (x/w)*Vsx + Vcx,$$

which is the scaling operation for the x coordinate, one unit computes the quotient x/w one bit at a time, starting with the most significant bit, while the other unit uses these successive bits to govern the accumulation of the product of this quotient with Vsx. In this way the perspective division and the multiplication of the scaling operation are done simultaneously. Moreover, since the output device typically has no more than 12 bits of precision, this operation may stop after 12 microcycles. Then Vcx is added to the accumulated result. As in the clipper mode of operation, the algorithm used is very similar to that described in (Sproull and Sutherland 1968).

A Brief Design History

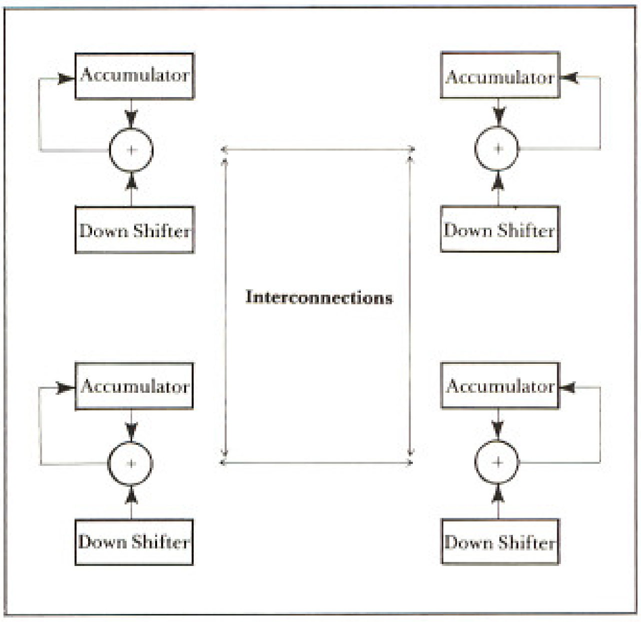
In July of 1979, Carver Mead of Caltech and Lynn Conway and several other people from the LSI Systems Area at the Xerox Palo Alto Research Center presented an intensive, three day course on designing nMOS circuits to some of the faculty and staff of Stanford. In search of a project, the author recalled that the Clipping-Divider, designed and built by Ivan Sutherland (Sproull and Sutherland 1968) in the days of small scale integration, was a relatively high performance device that operated with a clock cycle of approximately 200ns. It attained its high performance by having four units operate in parallel, each doing a clip operation somewhat like that described above. Unlike the present plane-at-a-time configuration, however, the units were tightly linked together in a configuration that simultaneously clipped lines to the four window boundaries. The plane-at-a-time notion is also due to Sutherland (Sutherland and Hodgman 1974). Figure 4 shows an approximate diagram of this organization. The initial project chosen was to implement the Clipping-Divider as a single nMOS chip.

After rediscovering the details of that design, a single bit-slice for the machine was designed and a layout was done using an interactive graphics layout system. This layout, using six micron design rules (lambda=three microns), was going to require a 9-mm-square chip. In addition, it had the unappealing feature that because of the architecture, there was quite a large amount of interconnect wiring within the bit slice. This wiring consumed as much area as the functional units of the bit-slice, making the bit-slice about twice as wide as it might otherwise be.

In addition to this, there was the annoying problem of transformations. A complete graphics system needs transformations as well as clipping and scaling, and the Clipping-Divider did only the latter functions. Another problem was that for a good user interface, the author felt that the coordinates supplied to the transformation unit should be floating-point.

In a discussion of the problem with Jim Rowson of Caltech, it was suggested that, for simplicity, the clipping be done one plane at a time. Because of the (unreasonable) desire to put everything on one IC chip, this was initially not considered desirable. Since the author was aware of the Reentrant Polygon Clipper of Sutherland and Hodgman (Sutherland and Hodgman 1974), however, it was clear that this scheme had the advantage that it would work for polygons as well as lines. That is, by pipelining the clipping operation, keeping a small amount of additional information and

FIGURE 4. Rough diagram of original clipping divider.

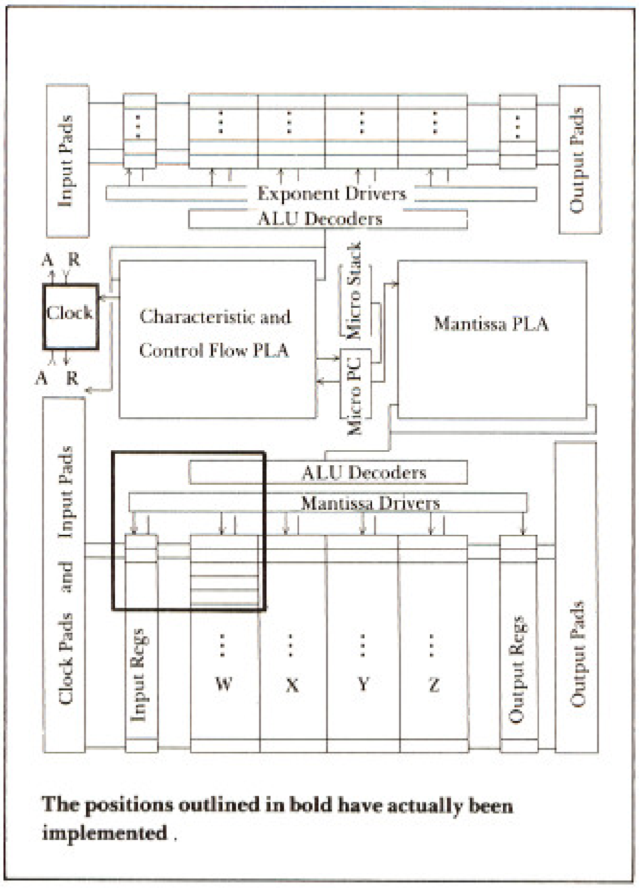


clipping to one plane at a time, it is possible to clip polygons and lines. The original Clipping-Divider algorithm worked only with lines.

There was still a desire to keep everything on one chip. The next major step came with the realization that a system like the Clipping-Divider actually does a multiplication in the scaling mode. Recall from the divide/scale operation described in the previous section that one of two units accumulates the partial products of a multiplication while the other generates bits of a quotient. However, two of the units are required in this mode because one is doing a division. It was suddenly realized that by making a simple bit slice with one register that shifts up in addition to the one that shifts down in the clipper mode, one unit could do multiplication as well as the division implicit in the clipping mode. Moreover, since a single plane clipper requires four function units, as described above, and since four function units can also simultaneously multiply four numbers, it was evident that it was possible to have a single chip that could do a four-component vector dot product as well as a single plane clipping operation.

The final step to enable a single-chip implementation came when it was realized that both an exponent and a mantissa would conveniently fit on either side of the control part of the unit, *i.e.*, the Programmable Logic Array, as shown in Figure 5. With one exception, the

FIGURE 5. Plan view of the Geometry Engine.



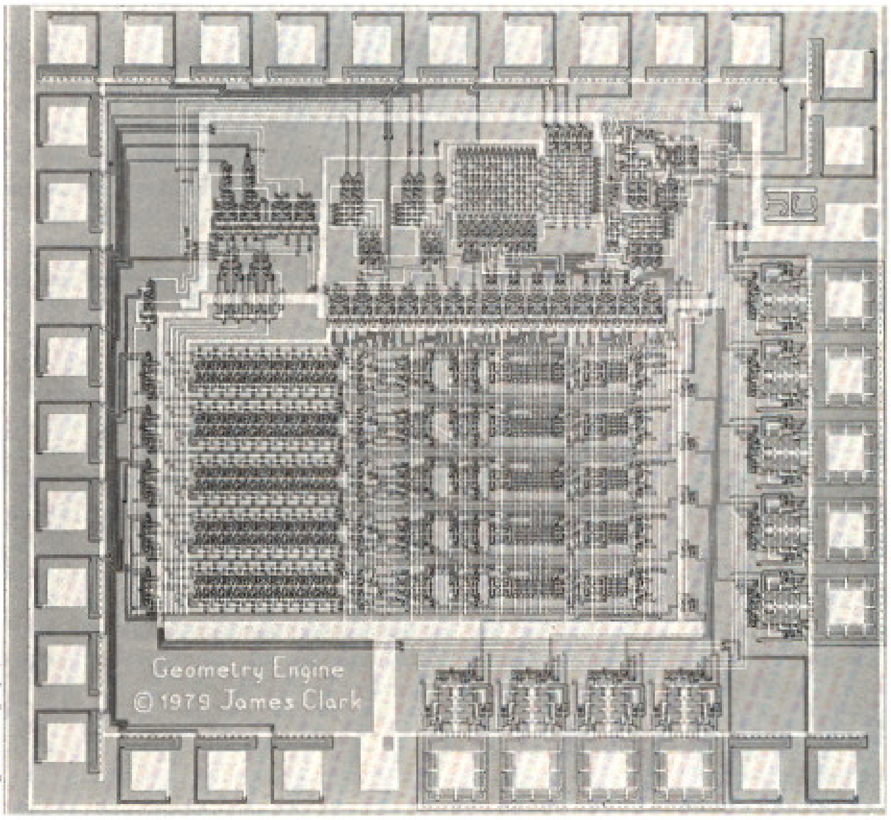
basic architecture was determined at this time.

The exception was that no provision was made for a matrix stack. A matrix stack allows for structured pictures. It is used to push away a current transformation matrix before going to a "picture subroutine," in much the same way that arguments are saved in going to a recursive subroutine in a regular program. It provides a picture hierarchy capability. It was originally thought that the matrix stack would be too space consuming to put on the chip and that it really belonged on a separate stack chip. When the time had come to consider the issue more seriously, however, it was discovered that putting the stack on a separate chip caused a lot of communication problems. Observe from Figure 2 that the organization has a clean, pipeline flow as shown, without an external matrix stack. To include a matrix stack chip so that matrices can be transferred from the *mm* chips on a Push command and back to the *mm* chips on a Pop command interrupts this flow and requires more complicated bus communication paths.

In puzzling over this need for a stack, it was realized that the best place for it, to simplify communication, was in the bit slice. Provision had already been made in the bit-slice for several extra storage registers in addition to the three already mentioned (Accumulator(A), Up Shifter(U),and Down Shifter(D)). That is, because as a polygon clipper the function units needed several save-point registers, the bit-slice at this time had two additional storage registers, S1 and S2. After actually doing a layout of a variant of the stack cell given in Mead and Conway, it was evident that about nine stack cells took only slightly more space than the two register cells; this was largely due to the need for fewer control lines. Also, since the top two elements of the stack were still useful for the temporary save-point registers, it was clear that the stack belonged with the bit-slice. In other words, the bit-slice became a microrepresentation of the entire system.

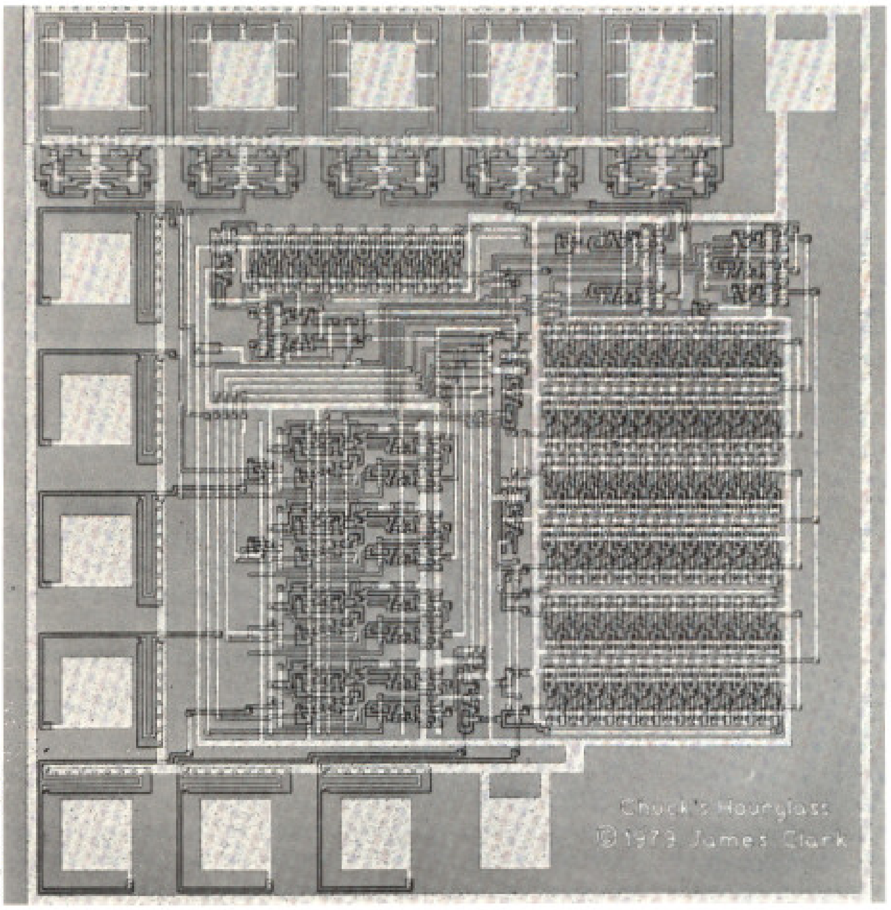
The ALU in the bit-slice is very similar to the one described in Mead and Conway (1980). Likewise, the stack cell is like that given in the same text except that power, ground, and control signals run the length of the stack rather than across it. Each of the registers, which are in the middle of the slice, is simply a pair of inverters with regenerative feedback on phase 1 of the clock. The symmetric U and D registers also have a bit of extra circuitry to communicate their value in the slice to the appropriate adjacent slice. As in the OM2, there is a single, precharged polysilicon bus running through the slice to communicate information between the slice and the Input/Output pads. The slice is "regular" since it adjoins to another copy simply by placing them side by side.

Figure 5 shows a plan view of the entire chip. The parts outlined with bold lines have been implemented as two separate projects on the multiuniversity, multiproject chip set, MPC79 (Conway, Bell and Newell 1980). The small rectangles appearing in this figure are copies of the principal bit-slice discussed above. The two projects



Melgar Photography

FIGURE 6. Feasibility of the Geometry Engine design was proven with this five-bit-wide slice of the data path.



Melgar Photography

FIGURE 7. Self-timed clock generation chip for the Geometry Engine.

implemented are a five-bit version of the main function unit and the pipeline clock discussed below. A photograph of the project with the five-bit slice is shown in Figure 6. This project worked as planned.

System Timing

Concurrent with all of these structural decisions were decisions about how the system was to communicate temporally. After reading the chapter in Mead and Conway on System Timing by Seitz, it seemed clear that a self-timed system would be the most resilient to changes of scale.

The timing methodology, which Seitz refers to as self-timed logic, was employed in the design for two reasons. The first was to avoid the clock-skew problems associated with distributing a clock over a very large circuit. Although clock distribution is not a problem with four-micron dimensions, as fabrication technology moves toward one-micron dimensions, clock skew becomes a problem. By using a self-generated clock on each of the 12 original chips, the scaled system will have 12 separately timed subsystems. Moreover, it will automatically run approximately four times as fast as a result of the decreased scale.

The other reason for using the self-timed convention was to avoid synchronization failure. By employing a separate clock that is tightly coupled with the control of each chip, the clock can be synchronously stopped when the unit is idle, waiting for input or waiting for output to be taken, and it can be asynchronously started by the unit on which it is waiting when the unit is ready. This avoids synchronization failure and allows the clock cycle to be varied according to the needs of the particular micro-instruction. Figure 7 shows a photograph of the clock chip. This chip also worked entirely as planned. Only the most complicated part of the design remained: writing the microcode for the PLA control.

Microprogramming

The Geometry Engine is a quasi general-purpose, four-component vector function unit. It gets its power as a result of its architectural simplicity; almost all of its complexity is in the microcode that drives it. This microcode is a representation of the logic equations for its finite state machine, which will be implemented in a Programmable Logic Array (PLA). Writing this microcode and making minor additions to the principal bit-slice to accommodate the requirements of the microcode has taken approximately 50 percent of the total design time.

The state of design tools in the VLSI design business is still relatively crude. The experience of this project has shown that when one has a very homogeneous architecture so that the layout problem for the main function unit is minimized, the most needed design tool is a mechanism for simulating the microcode and thence automatically generating a PLA layout from this simulated code. Although this need has been recognized by others, there has been no well-structured language available with a suitable PLA generating postprocessor.

In response to this need, John Hennessey of the Stanford Computer Systems Laboratory, has written a Pascal-based language for this purpose, and the author has added the PLA generation postprocessor (Hennessey 1980). This language has two modes of operation, simulation and PLA generation. In the simulation mode, the user defines with Pascal procedures an algorithmic statement of the function of the "environment" that is being driven by the logic equations of the micro-architecture. All PLAs are named, giving their layout orientations and naming their input and output signals with their polarities; for additional convenience, the signals may be labeled as "pipelined," thereby relieving the microprogrammer of worries related to pipelining. Then the microprogram is written in a very simple, clean form. The microcode is translated to Pascal procedures, loaded with the environment procedures and the result is then simulated. After satisfactory simulation of the microcode, the user then runs the system in the PLA generation mode and the CIF (Caltech Intermediate Form) for the PLA layout is output. This is then merged with the layouts of the functions units being driven by the microcode and they are wired together.

Summary

High performance nMOS circuits are possible with a very high degree of parallelism. Finding the correct architecture to exploit the parallelism is of course the difficult problem. The flexibility of structuring one's own building blocks to implement a specially conceived architecture is one of the key features of the design possibilities made readily available through the excellent, pragmatic book by Mead and Conway. This book and courses taught from it are encouraging a rapid evolution of sharable design tools at several major universities and corporations. As a result, a time is quickly approaching when a small number of people rather than very large teams can design, simulate, create the layout for, and readily implement new architectures.

Acknowledgments
The author would like to thank Lynn Conway of Xerox for making available the resources of her LSI Systems Area, enabling the author to learn the material quickly. Martin Newell was extremely helpful as a source of encouragement and insight.

References

Conway, L.A.; Bell, A.G.; and Newell, M.E. January 1980. "MPC79: A Demonstration-Operation of a prototype Remote-Entry Fast-Turnaround, VLSI Implementation System." Paper read at the Conference on Advanced Research in Integrated Circuits, M.I.T.

Hennessey, J. 1980. "Microprogram Simulation/PLA Compilation." Paper read at Stanford Computer Forum.

Mead, C.A. and Conway, L.A. 1980. *Introduction to VLSI Systems*. Boston: Addison Wesley.

Sproull, R.F. and Sutherland, I.E. 1968. "A Clipping Divider", In *Proceedings of the FJCC 1968*. Thompson Books, Washington, D.C.

Sutherland, I.E. and Hodgman, G.W. January 1974. "Reentrant Polygon Clipping", *CACM, vol. 17, no. 1.*