


1988

# Design of a VLSI scan conversion processor for high performance 3-D graphics systems

Han-Uei Huang  
*Iowa State University*

Follow this and additional works at: <https://lib.dr.iastate.edu/rtd>

 Part of the [Computer Sciences Commons](#), and the [Electrical and Electronics Commons](#)

## Recommended Citation

Huang, Han-Uei, "Design of a VLSI scan conversion processor for high performance 3-D graphics systems " (1988). *Retrospective Theses and Dissertations*. 8849.  
<https://lib.dr.iastate.edu/rtd/8849>

This Dissertation is brought to you for free and open access by the Iowa State University Capstones, Theses and Dissertations at Iowa State University Digital Repository. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of Iowa State University Digital Repository. For more information, please contact [digirep@iastate.edu](mailto:digirep@iastate.edu).

## INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book. These are also available as one exposure on a standard 35mm slide or as a 17" x 23" black and white photographic print for an additional charge.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

# U·M·I

University Microfilms International  
A Bell & Howell Information Company  
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA  
313/761-4700 800/521-0600



**Order Number 8909151**

**Design of a VLSI scan conversion processor for high  
performance 3-D graphics systems**

**Huang, Han-Uei, Ph.D.**

**Iowa State University, 1988**

**U·M·I**

300 N. Zeeb Rd.  
Ann Arbor, MI 48106



Design of a VLSI scan conversion processor for  
high performance 3-D graphics systems

by

Han-Uei Huang

A Dissertation Submitted to the  
Graduate Faculty in Partial Fulfillment of the  
Requirements for the Degree of  
DOCTOR OF PHILOSOPHY

Department: Electrical Engineering and  
Computer Engineering  
Major: Computer Engineering

**Approved:**

Signature was redacted for privacy.

**In Charge of Major Work**

Signature was redacted for privacy.

**For the Major Department**

Signature was redacted for privacy.

**For the Graduate College**

Iowa State University  
Ames, Iowa

1988

## TABLE OF CONTENTS

	PAGE
1. INTRODUCTION . . . . .	1
1.1. Computer Image Generation Process . . . . .	1
1.1.1. Model . . . . .	1
1.1.2. Viewing Transformation . . . . .	2
1.1.3. Clipping . . . . .	3
1.1.4. Perspective Transformation . . . . .	3
1.1.5. Hidden Surface Elimination . . . . .	3
1.1.6. Pixel Painting . . . . .	4
1.1.7. Raster Display . . . . .	4
1.2. Performance Requirement . . . . .	4
2. PREVIOUS WORK . . . . .	6
2.1. 8-by-8 Display System . . . . .	6
2.2. Video RAMS (VDRAM) . . . . .	7
2.3. Scan-Line Access Memories (SLAM) . . . . .	8
2.4. The Integrated Display Controller . . . . .	9
2.5. Pixel Planes . . . . .	9
2.6. Super Buffer . . . . .	10
3. SCAN CONVERSION PROCESSOR ARCHITECTURE . . . . .	12
3.1. Scan Conversion Algorithm . . . . .	12
3.2. Hardware Implementation . . . . .	15
3.2.1. Command Generator . . . . .	15
3.2.2. Bounds-Checking Processor . . . . .	16
3.2.3. Scan Conversion Processor Operation . . . . .	17
3.3. The Structure of SCP . . . . .	20
3.4. Image Generation Process . . . . .	22
4. TIMING AND CONTROL . . . . .	24
4.1. Clocking Scheme . . . . .	24
4.2. Command Timing . . . . .	26
4.2.1. INIT (Initialization) . . . . .	26
4.2.2. NI (New Image) . . . . .	26
4.2.3. NP (New Polygon) . . . . .	27
4.2.4. ED (Edge Definition) . . . . .	27
4.2.5. ZDC (Z-depth Calculation) . . . . .	29
4.2.6. PRC, PGC, and PBC (Paint R, G, B Colors) . . . . .	31
4.3. Command Sequencer . . . . .	35
4.3.1. Command Register . . . . .	35
4.3.2. Sequencing Signals . . . . .	37
4.3.3. Counter . . . . .	38
4.3.4. Execution of Command NI . . . . .	38
4.4. Image Scan-Out . . . . .	39
4.4.1. Image Buffer Access Control . . . . .	39
4.4.2. Scan-Out Control . . . . .	40
4.4.3. Scan-Out Timing . . . . .	41
4.4.4. Memory Circuit . . . . .	42

4.5. A-BUS Transfer . . . . .	42
5. LAYOUT AND PERFORMANCE EVALUATION . . . . .	46
5.1. Design Environment . . . . .	46
5.1.1. CAD Tools . . . . .	46
5.1.2. Simulators . . . . .	46
5.1.3. Technology Used . . . . .	47
5.2. Layout Design . . . . .	48
5.3. Performance Evaluation . . . . .	49
5.3.1. Critical Signal Propagation Path . . . . .	51
5.3.2. Estimation of Gate Delay . . . . .	52
5.3.3. Estimated Throughput . . . . .	53
6. DISCUSSION AND FUTURE WORK . . . . .	56
7. BIBLIOGRAPHY . . . . .	58
8. ACKNOWLEDGEMENTS . . . . .	60



## LIST OF TABLES

	PAGE
TABLE 3.1. SCP Commands . . . . .	19
TABLE 4.1. Timing of Command INIT . . . . .	27
TABLE 4.2. Timing of Command NI . . . . .	28
TABLE 4.3. Timing for Command NP . . . . .	28
TABLE 4.4. Timing for Command ED . . . . .	30
TABLE 4.5. Timing for Command ZDC . . . . .	32
TABLE 4.6. Timing for Commands PRC, PGC and PBC . . . . .	34
TABLE 5.1. Guidelines for ignoring RC wiring delays . . . . .	50
TABLE 5.2. Influence of first-order scaling on MOS device characteristics . . . . .	50
TABLE 5.3. Processing Time of one Polygon . . . . .	54

## LIST OF FIGURES

	PAGE
FIGURE 1.1. Steps of image generation . . . . .	2
FIGURE 3.1. Diagram of a raster graphics system . . . . .	13
FIGURE 3.2. The position of a SCP in the screen area . . . . .	18
FIGURE 3.3. The structure of SCP processor . . . . .	21
FIGURE 3.4. Image generation process . . . . .	23
FIGURE 4.1. Clock Waveforms for $\phi$ and $\psi$ . . . . .	25
FIGURE 4.2. Block Diagram of Command Sequencer . . . . .	36
FIGURE 4.3. Buffer select circuit . . . . .	40
FIGURE 4.4. A block diagram of the global token chain . . . . .	43
FIGURE 4.5. A block diagram of the local token ring . . . . .	44
FIGURE 4.6. The 1-bit circuit example for A-BUS . . . . .	45

## 1. INTRODUCTION

An old adage says that "A picture is better than a thousand words" because a picture can carry much more information than a thousand words. Nowadays graphics capability has become an indispensable part of a computer system. However, the cost of a high performance graphics system that can render 3-D images in real time is still very high. A lot of research efforts have been made in the past twenty years and have resulted in many interesting high performance architectures for real time raster graphics. No single system as yet stands out as the best solution.

The objective of this research is to design a VLSI scan conversion chip to eliminate the bottleneck in the image generation process and at the same time lower the system cost.

### 1.1. Computer Image Generation Process

Figure 1.1 shows the sequence of operations required to generate a raster display out of the model data describing the scene [1,18]:

#### 1.1.1. Model

A computer graphics system contains a data base of object models [7]. An object model is a scene description

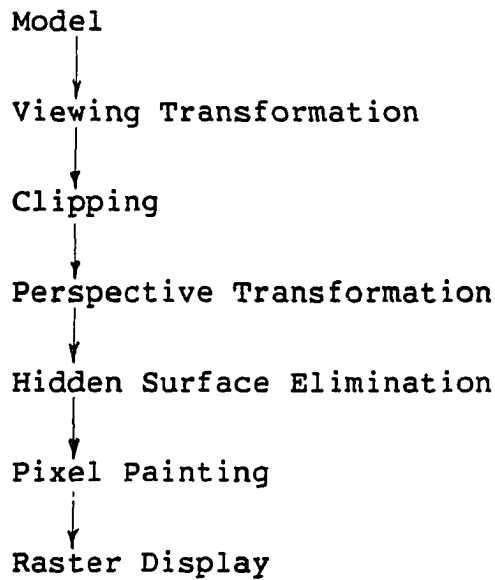


FIGURE 1.1. Steps of image generation

of one or more objects, each of which is described by a set of convex polygons that approximate its surfaces. Polygons are processed one by one in any order. Each polygon is described by a sequence of vertices whose  $x, y, z$  coordinates are in the "world" coordinate system. Associated with each vertex is a triple  $(R, G, B)$  which specifies the color intensity at that vertex.

### 1.1.2. Viewing Transformation

To generate an image, the application program combines basic objects into more complex objects [7]. This process involves the processing of polygons. The processing of a

polygon begins with the coordinates of its vertices being transformed into the specific position and direction. The involved transformations include translation, scaling, and rotation.

#### 1.1.3. Clipping

A portion of an object may be outside the field of view and invisible to the viewer, and hence must be clipped [7].

#### 1.1.4. Perspective Transformation

The human visual system has the characteristics that for two objects of same size but at different distance from the viewer, the farther one looks smaller [7]. To provide such realism, perspective transformation is performed. The exact color intensity at each vertex is also calculated based on the distance and direction to the light source. The output at this step is the scene description expressed in the coordinates of the display device.

#### 1.1.5. Hidden Surface Elimination

In the perspective transformation stage, several object points may be projected onto the same point on the screen. However, only the point nearest to the viewer is visible. The other object points are invisible and hence must be eliminated. This process is called hidden surface

elimination. In convention, the distance to the screen is called z-depth.

#### 1.1.6. Pixel Painting

After eliminating the hidden surfaces, the color intensity at each pixel is calculated and written into the image buffer.

#### 1.1.7. Raster Display

The color intensity data stored in the image buffer is scanned from left to right, and from top to bottom, and displayed on the screen.

### 1.2. Performance Requirement

Steps 1-4 of the image generation process are grouped as geometry processing while steps 5-6 are grouped as pixel processing. In the geometric transformation stage, the vertices of the graphical primitives are converted from the three-dimensional model in the display list  $(x,y,z)$  to their proper position on the screen  $(X_{\text{screen}}, Y_{\text{screen}})$ . The transformation formulas are:

$$\begin{aligned} X_{\text{screen}} &= (AX + BY + CZ + D) / W \\ Y_{\text{screen}} &= (EX + FY + GZ + H) / W \\ W &= IX + JY + KZ + L \end{aligned}$$

This transformation, which performs scaling, translation, rotation, and perspective transformation, involves 20 floating-point operations (9 additions, 9 multiplications, and 2 divisions) for each 3-D vertex.

Usually a scene consists of 1000 polygons with an average size of 100 x 100 pixels [5]. To achieve real time performance, at least 30 images must be generated in one second. If each polygon contains four edges, then about 80,000 floating-point operations must be performed in 1/30 seconds or 2.4 flops per microsecond. Obviously this requirement cannot be satisfied by software nor by those conventional math coprocessors (e.g., Intel 80287 or 80387). However, there are several commercial math chips and digital signal processors (e.g., AMD 29327, TI TMS-32030) can complete one floating-point operation in less than 250 ns. Clipping takes much less time than transformations. Therefore, the geometrical transformation is no longer the performance bottleneck in a graphics system [5].

However, 10 million pixels per image must be processed in the pixel processing stage which means 300 million pixels per second must be computed, accessed, and modified. This implies that a scan conversion system must spend less than 3.3 ns per pixel on the average in order to perform in real time. It is clear that this performance requirement can be met only by a multiprocessor approach.

## 2. PREVIOUS WORK

Many graphics processing systems have been proposed in the past ten years. Some of them tried to eliminate the bottleneck of memory accessing, some tried to speed up the geometry processing and still others tried to solve the difficulty in satisfying the throughput requirement of scan conversion processing. Here we will focus on the scan conversion subsystem of those architectures.

### 2.1. 8-by-8 Display System

The 8-by-8 Display System is an approximation to the processor-per-pixel idea [4,17,18]. In general, the 8-by-8 Display System consists of a square matrix of 64 processors and allows the simultaneous accessing of 64 pixels. For example, in Sproul and Sutherland's system [17], internal shifters and special memory addressing circuits are provided to make the access independent of word boundaries in the memory. Pixel manipulation functions are included to process the 64 pixels thus accessed, mask them, overwrite them with new information, or combine them logically with other pixels. The resulting data can be stored into any other 8 by 8 square of pixels in the subsequent memory cycle.



All those proposed 8-by-8 Display System did not provide hardware support for hidden surface elimination and smooth 3-D shading, and hence cannot render high quality 3-D images.

## 2.2. Video RAMS (VDRAM)

An image buffer is an essential part of a high performance graphics system. Double buffering is necessary if the image is to appear to be changing smoothly. Otherwise, the observer would see glimpses of half completed images as one is erased and the next is drawn. The difficulty with the double buffer is that it needs twice the memory. A single-buffered system saves the memory but the processor must contend the memory bandwidth with the rasterizer.

Recently, a number of memory makers have begun to offer special memory chips designed to alleviate this problem. These chips are referred to as "Video RAMS" and are available from Texas Instrument, NEC, and Advanced Micro Device. Video RAMS are conventional RAMs with a shift register added. When a row of memory is to be displayed, the video RAM accesses the entire row and moves it into the shift register. The data can then be shifted out independently of the memory filling.

However, the Video RAMs can not make writing into memory faster and hence are inadequate for high performance image rendering.

### 2.3. Scan-Line Access Memories (SLAM)

The limitation of conventional memory chips is that only one pixel can be accessed at a time although a row represents one scan line. A special-purpose processor is added to Demetrescu's SLAM [5] chip so that large parts of a scan line can be modified without ever having to move the pixels off the chip. The SLAM chip executes commands that it receives over the 19 bits bus lines. SLAMs are capable of directly executing the horizontal line-fill commands that form the basis of the polygon-fill function.

The fill operation is accomplished through the use of four SLAM commands. The first command specifies the scan line that is to be filled (LOAD Y). The second command specifies a 16 bit pattern which will be used to fill pixels that will be modified. The third command specifies the  $X_{right}$  coordinate and directs the SLAM to read the selected scan line from the memory array. Finally, the fourth command specifies the  $X_{left}$  coordinate and directs the SLAM to modify all pixels in the range  $|X_{left}, X_{right}|$  and to write the scan line back into the memory array.

Like Video RAM, a shift register is incorporated in the SLAM to perform the similar function. A system of SLAM chips can modify from one to many thousands of pixels simultaneously. However, the SLAM chip does not support hidden surface elimination nor smooth shading.

#### 2.4. The Integrated Display Controller

The Integrated Display Controller (IDC) [18] allows the accessing and modifying of rectangles. The memory system consists of a special RAM and two address paths. These paths are composed of four m-of-n decoders, shift-registers and multiplexers. With this combination, it is possible to read and write any user defined pattern into a random rectangle.

The IDC can achieve significant performance in updating the image. Like SLAM, the IDC does not support smooth shading nor hidden surface elimination, and hence is not suitable for high quality 3-D image rendering applications.

#### 2.5. Pixel Planes

Henry Fuchs and John Poulton [8,9,15,16] incorporated a one-bit ALU to each pixel in their pixel-plane memory chip. The memory chips automatically perform, 1) Scan conversion (calculating the pixels that fall within a line segment,

convex polygon, or circle), 2) Visibility calculations based on the depth buffer algorithm, and 3) Pixel painting (either flat or Gouraud smooth shading). The above operations are performed by the variation of the calculation at each pixel, of the function

$$F(x,y) = A * x + B * y + C$$

where  $x$  and  $y$  are the coordinates of the pixel in the display space.

The preprocessor broadcasts the coefficients  $A$ ,  $B$  and  $C$  bit-by-bit to all the pixel-plane chips and all operations are performed bit-serially. Each chip has 64 pixels.

A full scale system (512 x 512 pixels) has been built at University of North Carolina at Chapel Hill using 2048 pixel plane chips. The system can process up to 25,000 four-sided polygons per second. Higher throughput is still desired.

## 2.6. Super Buffer

Super Buffer [10] is a systolic VLSI Graphics Engine for Real Time Raster Image Generation proposed at Cornell University. A prototype is under construction. The algorithm for generating the image is very similar to Cohen and Demetrescu's approach [18]. For the resolution of 512 x

512 pixels, only a 512 x 1 scan line buffer is needed. This system consists of one painting station and a raster graphics engine, i.e., super-buffer chip. Polygons are sorted according to their smallest y axis before being processed by the painting station. The painting station calculates the intersection of polygons with each scan-line, and broadcasts the x-coordinates of the intersection and the color to the super buffer.

There are 512 processors in the super buffer chip. Each processor has its own ID which is initialized during system initialization. Each processor compares the intersection x-coordinates with its own ID. If the ID is between the two x-coordinates, then the color is written into the buffer. This operation is finished in one clock cycle. After finishing this operation, the processor sends the color and the two x-coordinates to its right neighboring processor. The scan out of pixels is controlled by the shifting of a token. The proposed design does not provide the capability of smooth shading nor the hidden surface elimination.

### 3. SCAN CONVERSION PROCESSOR ARCHITECTURE

To solve the problem arising from smooth shading and hidden surface elimination, a new processor architecture has been invented which has been labeled as a scan conversion architecture (SCP). The SCP is design to support the raster graphics system for rapidly rendering 3-D objects and scenes. It includes on-chip memory for image buffering and z-depth, and a ALU to perform all the pixel processing tasks. The target raster graphics system which the SCP will support has a block diagram like Figure 3.1.

#### 3.1. Scan Conversion Algorithm

As mentioned in Chapter 1, a polygon consists of several edges. The edge equation is a linear function of the form

$$F(x,y) = A * X + B * Y + C \text{ ----- (1)}$$

where x and y are coordinates of a specific pixel.

The z-depth of any pixel can be expressed as a linear function of x and y coordinates similar to equation (1).

Gouraud smooth shading is the most widely used method for computing the R,G,B color intensities. This method can produce high quality color image. Using this method, the R,G,B color intensities can also be expressed as a linear

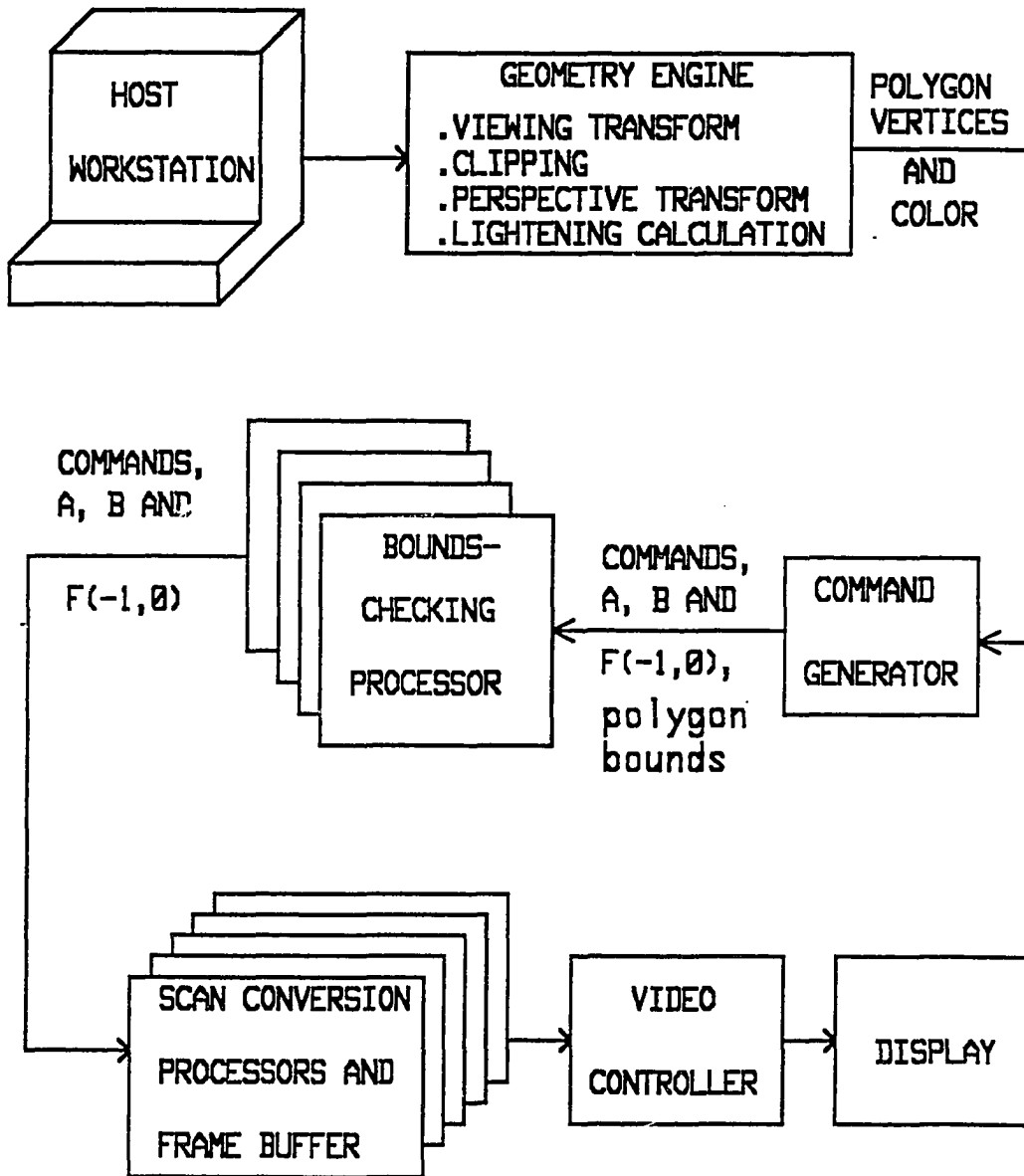


FIGURE 3.1. Diagram of a raster graphics system

function of  $x$  and  $y$  coordinates of a pixel similar to equation (1).

Polygons are processed one by one in any order. The scan conversion process is divided into the following three steps [9]:

1. Determining whether a specific pixel is located inside the given polygon. By evaluating the function value of all the edge equations at each pixel, a pixel is said to be located inside the given polygon if all the calculated edge function values are less than zero.
2. Determining the visibility of all the pixels located inside the given polygon. When generating a new image, the  $z$ -depth value at each pixel is initialized to be 1111111111. The  $z$ -depth value is computed using linear equation and compared with its previous minimum value. If the computed value is smaller and the pixel is inside the given polygon then the previous minimum  $z$ -depth value is replaced by the new value.
3. Calculating the R,G,B color intensities for those visible pixels. The R,G,B color intensities are also calculated using linear equations.



Since all the performed operations are the evaluation of linear equations, they can be speeded up by using incremental addition. Further, we suggest a graphics system architecture that allows the simultaneous processing of many polygons to achieve higher throughput. The hardware implementation of this algorithm is described in the next section.

### 3.2. Hardware Implementation

The Host Workstation shown in Figure 3.1 maintains and manipulates a database of object descriptions. The Geometry Engine performs viewing transform, clipping, perspective transform, and lightening calculations. The Host Workstation can be the Geometry Engine. The screen is divided into several areas (e.g., 16) and each area is assigned to a Bounds-Checking Processor so that many polygons can be processed at the same time. The other blocks are detailed as below.

#### 3.2.1. Command Generator

Receiving polygon vertex coordinates and their associated information, the Command Generator calculates the coefficients and function values at location  $(-1,0)$  for edge, z-depth and R,G,B color intensities. Polygon bounds are also computed. Then the polygon bounds, commands and

associated information needed in polygon processing are broadcasted to all the Bounds-Checking Processors. The design of the Command Generator is not part of this research and will not be discussed in more detail.

### 3.2.2. Bounds-Checking Processor

When processing a polygon, the Command Generator first broadcasts the polygon bounds and then commands for edge definition, z-depth computation, and color intensity calculation to all Bounds-Checking Processors. The Bounds-Checking Processor checks the polygon bounds to determine whether the given polygon has any intersection with its controlled area. If YES, the commands and associated information following the polygon bounds will be either rebroadcasted immediately to its controlled SCPs or saved on its on-chip queue and be rebroadcasted later depending on whether its controlled SCPs are still busy. If NO, the commands and associated information will be discarded.

Since the processing time of one polygon can be determined beforehand, the Bounds-Checking Processor (BCP) knows whether its controlled SCPs are still busy. The design of the BCP is not part of this research and will not be described in more detail.

### 3.2.3. Scan Conversion Processor Operation

Each SCP is designed to handle 64 pixels. Assume  $(X_S, Y_S)$  is the coordinate of the upper left pixel on any SCP chip as shown in Figure 3.2. To facilitate the computation of the linear function, each SCP is assigned a processor ID  $(XID, YID)$  where

$$XID = X_S \text{ div } 8$$

$$YID = Y_S \text{ div } 8$$

Eight commands (see Table 3.1) are implemented which are adequate for the rendering of 3-D objects. For those commands which involve the computation of linear function, the SCP first computes the function value at  $(X_S-1, Y_S)$  using the following equation

$$F(X_S-1, Y_S) = F(-1, 0) + (XID * A + YID * B) * 8$$

where the multiplication by 8 is implemented by left shifting three bits.

The linear function value at any pixel can be calculated by incremental addition and will be detailed in the next chapter (either adding A to its left neighbor pixel's or adding B to its upper neighbor pixel's function value). Here, the function value and coefficients are integers.

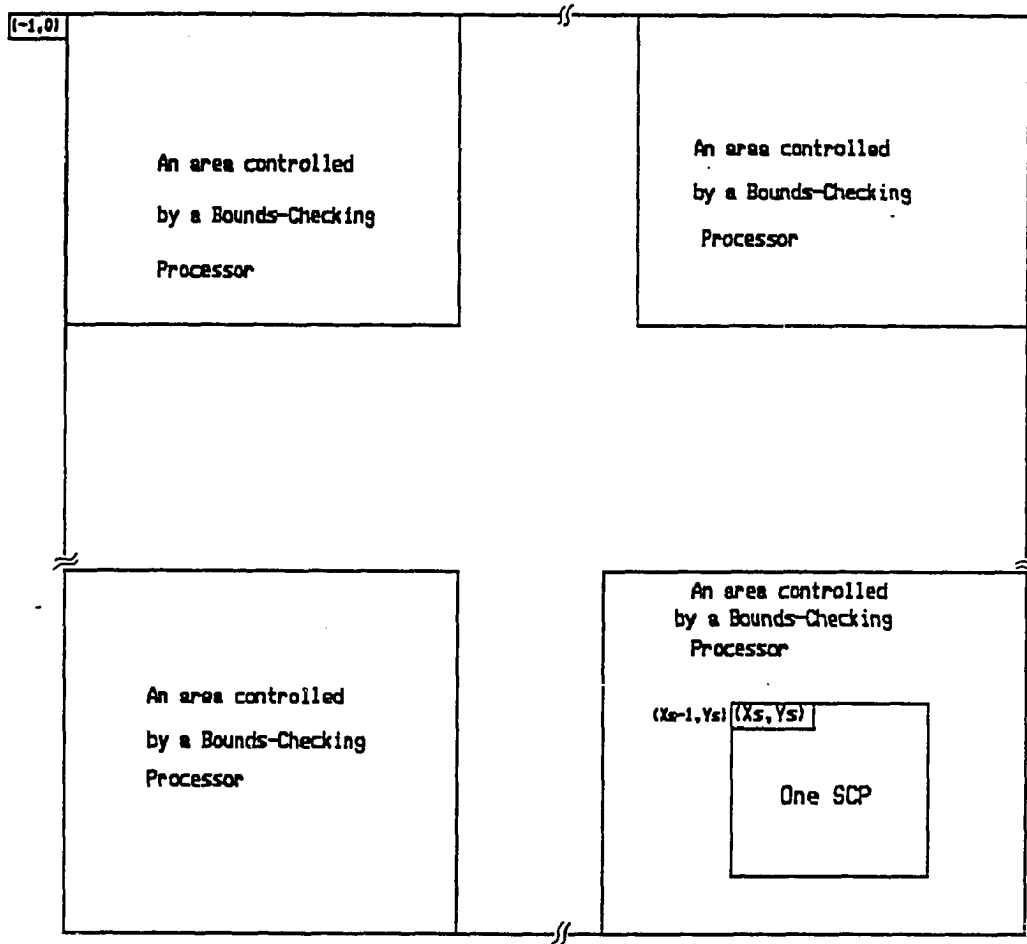


FIGURE 3.2. The position of a SCP in the screen area

TABLE 3.1. SCP Commands

commands	operation
INIT (initialization)	Initialization of XID and YID of each SCP
NI (new image)	all z-depth values are set to 1111111111. All color intensities are reset to 0. Image buffers are swapped.
NP (new polygon)	all enable flags are set to one.
ED (edge definition)	Computation of linear function values at 64 pixels. If any pixel's edge function value is less than zero, its flag will be reset.
ZDC (z-depth computation)	Computation of z-depth value of 64 pixels. If any pixel's new z-depth value is less than its previous minimum value and also located inside the current polygon, then its z-depth value will be updated. Otherwise, its enable flag is reset.
PRC (paint red color)	Calculation of red color intensity of 64 pixels. If any pixel's enable flag is set, the red color intensity will be written into the buffer memory.
PGC (paint green color)	Same as PRC except that green color is painted.
PBC (paint blue color)	Same as PRC except that blue color is painted.

### 3.3. The Structure of SCP

The block diagram of a SCP chip is shown in Figure 3.3. Commands are latched by the command register which consists of a 4 D-type master-slave flip-flops. The sequencer generates all the sequencing signals and pixel addresses for the SCP chip. An 1-of-64 decoder decodes the address signals of 64 pixels. Multiplication is performed by a 12 by 8 bits parallel multiplier based on modified booth algorithm [2] and its result is latched by a 20-bit D-type latch L1. ADD1 and ADD2 are 24 bit adders which use carry-lookahead technique to achieve high throughput.

XID and YID are 8-bit D-type latches which store the SCP processor ID. A and B are 12-bit D-type latches which hold coefficients A and B for any linear function. Both F1 and F2 are 24-bit D-type master-slave registers. The F1 latches the function value  $F(-1,0)$  and is one of ADD1's source and the temporary storage for ADD1's output. The F2 is used as one of ADD2's source and the temporary storage for ADD2's output.

There are two sets of color intensity buffers: (RB1,GB1,BB1) and (RB2,GB2,BB2). When one buffer is being updated, the other buffer is scanned out to refresh the screen display. Up to 256 intensity levels are supported for each color. The ZD is a 64 by 10 bits array which stores 64 pixel's z-depth values. The ENA is a 64 by 1 bit

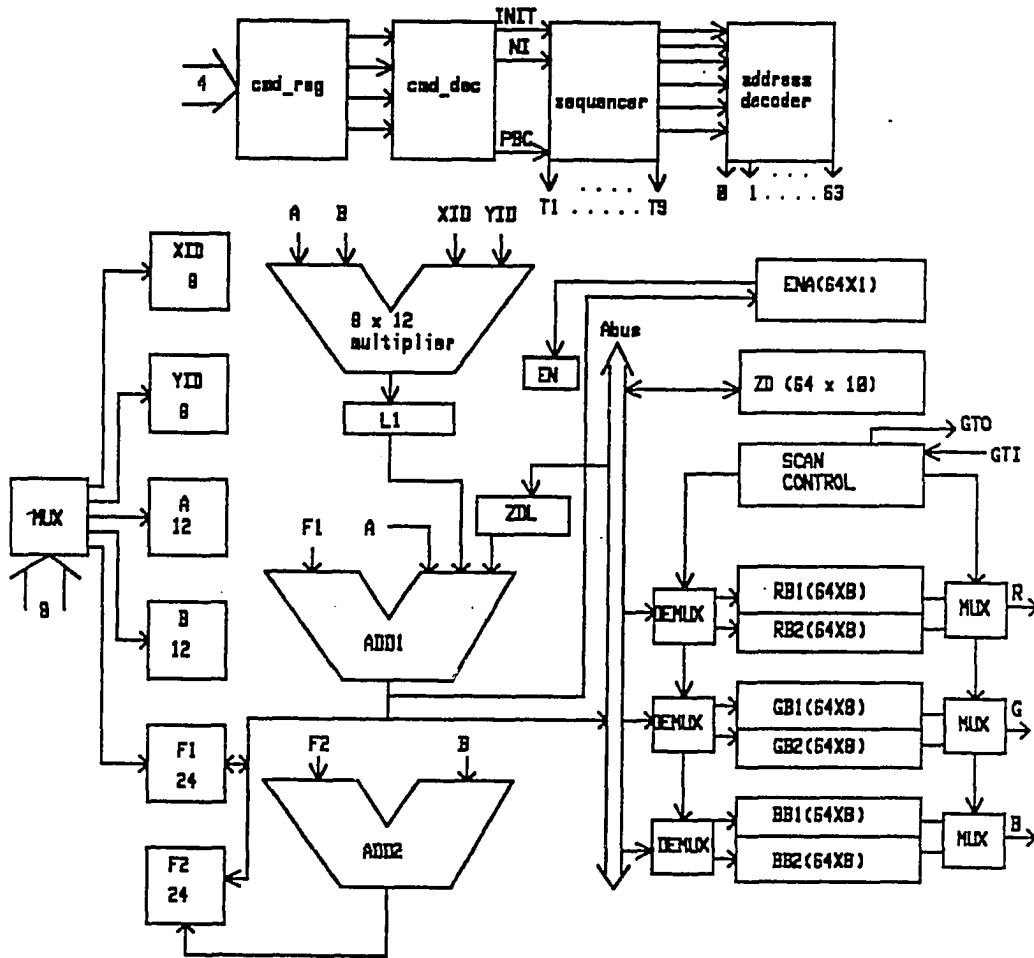


FIGURE 3.3. The structure of SCP processor

array which stores the enable flags of 64 pixels. The color intensity and z-depth value of any pixel can be updated only if its enable flag is set. The ENA is provided to keep track of those pixels located inside a given polygon and also to support hidden surface elimination.

The ZDL is a 10-bit D-type latch which is used as a temporary storage for the z-depth value of any pixel. The EL is a 1-bit D-type latch which is a temporary storage for any pixel's enable flag. A scan-out controller is incorporated to select one buffer to be updated and the other to be scanned out. The ADD1 accesses the ZD array and color buffers via the 10-bit A-BUS.

### 3.4 Image Generation Process

In a system that uses our design, the SCP must be reset after power-on. From then on, commands can be latched and executed. "INIT" will be the first command to be executed so that the processor ID of each SCP can be established. The image generation process in terms of command execution is shown in Figure 3.4.



1. Execute 1 "NI" command
2.  $I := \text{polygon number} - 1$
3. Execute 1 "NP" command
4. Execute as many "ED" commands as the edge number of the given polygon
5. Execute 1 "ZDC" command
6. Execute 1 "PRC" command
7. Execute 1 "PGC" command
8. Execute 1 "PBC" command
9. If  $I \neq 0$  GOTO step 3  
else stop.

FIGURE 3.4. Image generation process

#### 4. TIMING AND CONTROL

This chapter discusses the SCP clocking scheme, command timing, and the design of the command sequencer and the image scan-out controller.

##### 4.1. Clocking Scheme

The operation of the SCP chip is controlled by two independent non-overlapped 2-phase clocks ( $\phi$  and  $\psi$ ). The  $\phi$  clock controls the overall operation except the image scan-out which is controlled by the  $\psi$  clock. The basic idea behind the 2-phase clock is that one phase is used to drive each part of a command cycle. The timing diagram of  $\phi$  and  $\psi$  are shown in Figure 4.1.

It is very important that none of the phases of any clock overlaps. This is to ensure that one internal event is completed before the other is begun. These clock phases are used by the SCP to gate the control signals which are read from and written into internal latches. Non-overlapped time is essential for digital latches as it allows for the setup and hold times needed to reliably latch the data. In some designs this delay between phases is handled by circuitry on the chip [13]. For the SCP we chose to keep this constraint external to simplify the design. If the non-overlap delay is to be handled on chip, the clock skews

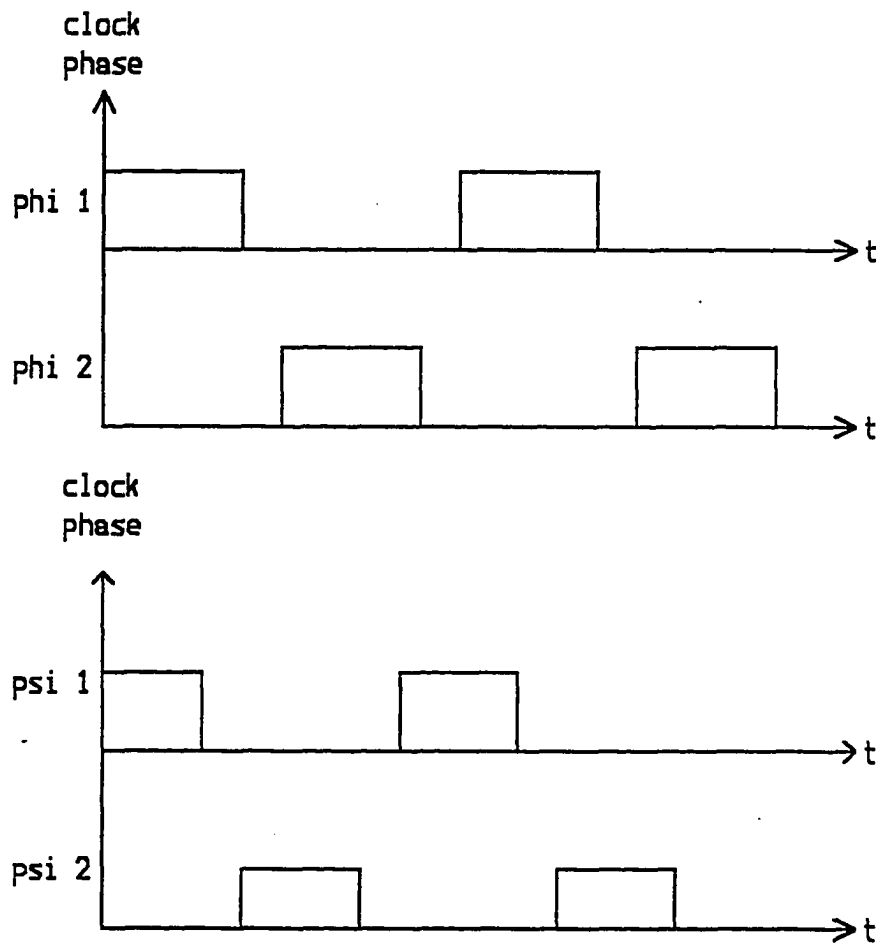


FIGURE 4.1. Clock Waveforms for  $\phi$  and  $\psi$

and time delays of certain critical signals need to be known and accounted for by on-chip clock circuits.

#### 4.2. Command Timing

The execution of each command can be divided from 1 to 138 steps. Each step is finished in one  $\phi$  clock cycle. The timing of each command is described in this section.

##### 4.2.1. INIT (Initialization)

During system initialization, a global token is generated and flows from one SCP chip to another SCP chip to control the latching of processor IDs. When the token is residing on one SCP chip, the corresponding processor ID must appear on the system bus and be latched. The timing of this command is shown in Table 4.1:

##### 4.2.2. NI (New Image)

This command is issued whenever a new image is going to be generated. The following operations are performed:

- The two image buffers are swapped.
- All the enable flags are set to 1.
- All the z-depth values are initialized to be  
1111111111.

TABLE 4.1. Timing of Command INIT

operation	clock cycle	clock phase
command latched	T0	$\phi 1$
global token shifts in	Tx	$\phi 1$
XID latched	Tx	$\phi 2$
YID latched	Tx+1	$\phi 2$
global token shifts out	Tx+2	$\phi 1$

- The RGB color intensity of each pixel in the buffer which is to be updated is initialized to be 0.

The timing of this command is shown in Table 4.2.

#### 4.2.3. NP (New Polygon)

The command NP is issued before a new polygon is processed. This command sets all the enable flags to one. The timing of this command is shown in Table 4.3:

#### 4.2.4. ED (Edge Definition)

The purpose of this command is to determine which pixel is located inside a given polygon. In order to simplify the function computation by using incremental addition, the image buffer are organized as an 8-by-8 array. The SCP first calculates the function value at  $(X_S-1, Y_S)$ , i.e.,

TABLE 4.2. Timing of Command NI

operation	clock cycle	clock phase
command latched	T0	$\phi 1$
two color intensity buffers swapped	T1	$\phi 1$
ZD(i) <- 1111111111, i = 0 to 63.	0 to 63	$\phi 2$
RBx(i) <- 0, i = 1 to 63 GBx(i) <- 0, x = 1 or 2 BBx(i) <- 0,	0 to 63	$\phi 2$

TABLE 4.3. Timing for Command NP

operation	clock cycle	clock phase
command latched	T0	$\phi 1$
ENA(i) <- 1, i = 0 to 63	T1	$\phi 2$

$F(X_S-1, Y_S)$ . Then function values of the first row pixels can be calculated by adding the coefficient A to their left neighboring pixels' function values. Before incremental addition is applied to calculate function values of the

second row pixels,  $F(X_S-1, Y_S+1)$  is calculated and saved at register F2. ADD2 is used to calculate  $F(X_S-1, Y_S+1)$ . Similarly, before calculating function values of pixels of  $i$ th row,  $F(X_S-1, Y_S+i-1)$  must be calculated. Each incremental addition takes one clock cycle.

The timing from latching the command ED to the processing of the first row is shown in Table 4.4. The timing of the processing of the remaining rows is identical with the first row.

After processing all edge equations of a given polygon, those pixels with their enable flags still set are located inside the polygon. As many as the edge number of command EDs must be executed when processing a polygon.

#### 4.2.5. ZDC (Z-depth Calculation)

The timing of the command 'ZDC' from latching the command to the processing of the first row is shown in Table 4.5. The calculation of the z-depth of one pixel takes two clock cycles. During the first clock cycle, the current minimum z-depth and the enable flag of current pixel are read. The new z-depth is also computed in the first cycle. During the second cycle, the new z-depth is compared with the current minimum z-depth value. If the new value is smaller and the enable flag is set, the new z-depth becomes the new minimum z-depth of the current pixel. The timing of

TABLE 4.4. Timing for Command ED

	operation	clock cycle	clock phase
	command latched coef A(0:5) latched coef A(6:11) latched coef B(0:5) latched coef B(6:11) latched F(0:7) latched into F1(0:7) F(8:15) latched into F1(8:15) F(16:23) latched into F1(16:23) L1 ← A * XID	T0 T1 T2 T3 T4 T5 T6 T7 T7	φ1 φ2 φ2 φ2 φ2 φ2 φ2 φ2 φ2
	compute F1 + L1 F1 ← F1 + L1 L1 ← B * YID	T8 T8 T8	φ1 φ2 φ2
	compute F1 + L1 F1 ← F1 + L1 F2 ← F1 + L1 MEM_ENA ← 1	T9 T9 T9 T9	φ1 φ2 φ2 φ2
pixel #	operation	count	phase
0	compute F1 + A, update N compute F2 + B ENA(0) ← ENA(0) and N F1 ← F1 + A, F2 ← F2 + B	0 0 0 0	φ1 φ1 φ2 φ2
1	compute F1 + A, update N F1 ← F1 + A ENA(1) ← ENA(1) and N	1 1 1	φ1 φ2 φ2
2	compute F1 + A, update N F1 ← F1 + A ENA(2) ← ENA(2) and N	2 2 2	φ1 φ2 φ2
3	compute F1 + A, update N F1 ← F1 + A ENA(3) ← ENA(3) and N	3 3 3	φ1 φ2 φ2
4	compute F1 + A, update N F1 ← F1 + A ENA(4) ← ENA(4) and N	4 4 4	φ1 φ2 φ2



TABLE 4.4 (continued)

5	compute F1 + A, update N flag F1 ← F1 + A ENA(5) ← ENA(5) and N	5 5 5	φ1 φ2 φ2
6	compute F1 + A, update N F1 ← F1 + A ENA(6) ← ENA(6) and N	6 6 6	φ1 φ2 φ2
7	compute F1 + A, update N F1 ← F2, ENA(7) ← ENA(7) and N	7 7 7	φ1 φ2 φ2

the processing of the remaining rows is identical with the first row.

#### 4.2.6. PRC, PGC, and PBC (Paint R, G, B Colors)

The timing of these commands are identical. The timing of the command PRC from latching the command to the processing of the first row is shown in Table 4.6. The processing of one pixel takes one clock cycle. The calculation of the new color intensity and the reading of the enable flag is performed during the φ1. If the enable flag is set, the color intensity will be written into the buffer during the φ2. The timing of the processing of the remaining rows are identical with the first row.

TABLE 4.5. Timing for Command ZDC

pixel #	operation	clock	phase
	command latched coef A(0:5) latched coef A(6:11) latched coef B(0:5) latched coef B(6:11) latched F(0:7) latched into F1(0:7) F(8:15) latched into F1(8:15) F(16:23) latched into F1(16:23) L1 ← A * XID	T0 T1 T2 T3 T4 T5 T6 T7 T7	φ1 φ2 φ2 φ2 φ2 φ2 φ2 φ2 φ2
	compute F1 + L1 F1 ← F1 + L1 L1 ← B * YID	T8 T8 T8	φ1 φ2 φ2
	compute F1 + L1 F1 ← F1 + L1 F2 ← F1 + L1 MEM_ENA ← 1	T9 T9 T9 T9	φ1 φ2 φ2 φ2
0	compute F1 + A, read ENA(0) compute F2 + B F1 ← F1 + A, read ZD(0) F2 ← F2 + B compute F1 - ZD(0), update N flag write F1 to ZD(0) if (N and ENA(0)) = 1 ENA(0) ← ENA(0) and N flag	S1 S1 S1 S1 S2 S2 S2	φ1 φ1 φ2 φ2 φ1 φ2 φ2
1	compute F1 + A, read ENA(1) F1 ← F1 + A, read ZD(1) compute F1 - ZD(1), update N flag write F1 to ZD(1) if (N and ENA(1)) = 1 ENA(1) ← ENA(1) and N flag	S1 S1 S2 S2 S2	φ1 φ2 φ1 φ2 φ2
2	compute F1 + A, read ENA(2) F1 ← F1 + A, read ZD(2) compute F1 - ZD(2), update N flag write F1 to ZD(2) if (N and ENA(2)) = 1 ENA(2) ← ENA(2) and N flag	S1 S1 S2 S2 S2	φ1 φ2 φ1 φ2 φ2

TABLE 4.5 (continued)

3	compute F1 + A, read ENA(3) F1 ← F1 + A, read ZD(3) compute F1 - ZD(3), update N flag write F1 to ZD(3) if (N and ENA(3)) = 1 ENA(3) ← ENA(3) and N flag	S1 S1 S2 S2 S2	$\phi 1$ $\phi 2$ $\phi 1$ $\phi 2$ $\phi 2$
4	compute F1 + A, read ENA(4) F1 ← F1 + A, read ZD(4) compute F1 - ZD(4), update N flag write F1 to ZD(4) if (N and ENA(4)) = 1 ENA(4) ← ENA(4) and N flag	S1 S1 S2 S2 S2	$\phi 1$ $\phi 2$ $\phi 1$ $\phi 2$ $\phi 2$
5	compute F1 + A, read ENA(5) F1 ← F1 + A, read ZD(5) compute F1 - ZD(5), update N flag write F1 to ZD(5) if (N and ENA(5)) = 1 ENA(5) ← ENA(5) and N flag	S1 S1 S2 S2 S2	$\phi 1$ $\phi 2$ $\phi 1$ $\phi 2$ $\phi 2$
6	compute F1 + A, read ENA(6) F1 ← F1 + A, read ZD(6) compute F1 - ZD(6), update N flag write F1 to ZD(6) if (N and ENA(6)) = 1 ENA(6) ← ENA(6) and N flag	S1 S1 S2 S2 S2	$\phi 1$ $\phi 2$ $\phi 1$ $\phi 2$ $\phi 2$
7	compute F1 + A, read ENA(7) F1 ← F1 + A, read ZD(7) compute F1 - ZD(7), update N flag write F1 to ZD(7) if (N and ENA(7)) = 1 F1 ← F2 ENA(7) ← ENA(7) and N flag	S1 S1 S2 S2 S2 S2	$\phi 1$ $\phi 2$ $\phi 1$ $\phi 2$ $\phi 2$ $\phi 2$

TABLE 4.6. Timing for Commands PRC, PGC and PBC

pixel #	operation	clock cycle	clock phase
	command latched coef A(0:5) latched coef A(6:11) latched coef B(0:5) latched coef B(6:11) latched F(0:7) latched into F1(0:7) F(8:15) latched into F1(8:15) F(16:23) latched into F1(16:23) L1 <- A * XID compute F1 + L1 F1 <- F1 + L1 L1 <- B * YID	T0 T1 T2 T3 T4 T5 T6 T7 T7 T8 T8 T8	$\phi 1$ $\phi 2$ $\phi 2$ $\phi 2$ $\phi 2$ $\phi 2$ $\phi 2$ $\phi 2$ $\phi 2$ $\phi 1$ $\phi 2$ $\phi 2$
	compute F1 + L1 F1 <- F1 + L1 F2 <- F1 + L1 MEM_ENA <- 1	T9 T9 T9 T9	$\phi 1$ $\phi 2$ $\phi 2$ $\phi 2$
	operation	clock cycle	clock phase
0	compute F1 + A, read ENA(0) compute F2 + B F1 <- F1 + A write F1 + A to RB(0) if ENA(0) = 1 F2 <- F2 + B	0 0 0 0 0	$\phi 1$ $\phi 1$ $\phi 2$ $\phi 2$ $\phi 2$
1	compute F1 + A, read ENA(1) F1 <- F1 + A write F1 + A to RB(1) if ENA(1) = 1	1 1 1	$\phi 1$ $\phi 2$ $\phi 2$
2	compute F1 + A, read ENA(2) F1 <- F1 + A write F1 + A to RB(2) if ENA(2) = 1	2 2 2	$\phi 1$ $\phi 2$ $\phi 2$
3	compute F1 + A, read ENA(3) F1 <- F1 + A write F1 + A to RB(3) if ENA(3)	3 3 3	$\phi 1$ $\phi 2$ $\phi 2$

TABLE 4.6 (continued)

4	compute F1 + A, read ENA(4) F1 ← F1 + A write F1 + A to RB(4) if ENA(4) = 1	4 4 4	φ1 φ2 φ2
5	compute F1 + A, read ENA(5) F1 ← F1 + A write F1 + A to RB(5) if ENA(5) = 1	5 5 5	φ1 φ2 φ2
6	compute F1 + A, read ENA(6) F1 ← F1 + A write F1 + A to RB(6) if ENA(6) = 1	6 6 6	φ1 φ2 φ2
7	compute F1 + A, read ENA(7) F1 ← F2 write F1 + A to RB(7) if ENA(7) = 1	7 7 7	φ1 φ2 φ2

### 4.3. Command Sequencer

The Command Sequencer is the control center of the SCP chip. The block diagram of the Command Sequencer is shown in Figure 4.2. Command Register, Command Decoder and the pixel address generator are also shown.

#### 4.3.1. Command Register

Commands are latched during φ1. After the Command Register is reset, the first command can be latched by the Command Register. Before the current command is completed, no new command should be latched. New command can be

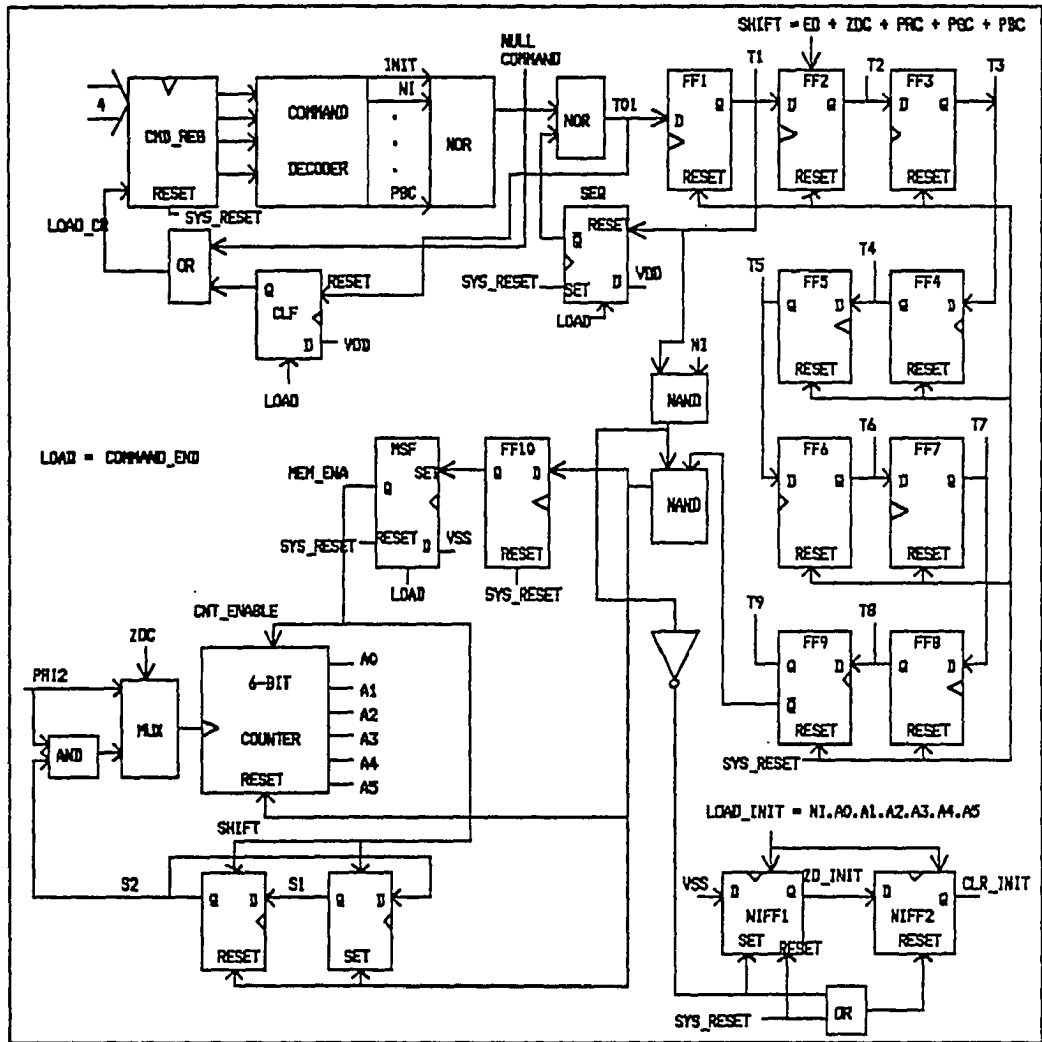


FIGURE 4.2. Block Diagram of Command Sequencer

latched only if either the control signal 'NULL COMMAND' or the flip-flop CLF's Q-output is asserted. When the Command Register is reset, the NULL COMMAND signal will be asserted. Whenever a new command is latched, the signal NULL COMMAND is pulled low and the flip-flop CLF is reset before the  $\phi_1$  of the next clock cycle and inhibits the latching of the next command. At the completion of the current command, high voltage will be loaded into the flip-flop CLF and allows the latching of the next command.

#### 4.3.2. Sequencing Signals

T1, T2, ... and T9 are the sequencing signals. T1 is asserted during the clock cycle following the command latching. T2 is asserted in the next clock cycle, etc. At the end of T9, the control signal MEM\_ENA will be asserted and allows the processing of the 64 pixels. Special care must be taken to the generation of T1, T2, ... and T9. When a new command is latched, the signal NULL COMMAND is pulled low and hence T1 will be generated in the next clock cycle. T1 should not be asserted in the following clock cycles. This is ensured by the flip-flop SEQ. The SEQ is reset after T1 is asserted, and hence set the input to the flip-flop FF1 to low until the command is completed. High voltage is loaded into SEQ at the command completion and hence allows the generation of T1 for the next command.

#### 4.3.3. Counter

Pixel addresses are generated by a 6-bit counter. Because it takes two clock cycles to process each pixel's z-depth, the counter increments every two clock cycles during the execution of command ZD. For other commands including NI, ED, PRC, PGC, and PBC, the counter increases in every clock cycle. To select the right clock rate, two flip-flops are incorporated for this purpose. In the execution of command ZD, the counter increases only when S2 is asserted. In addition, the counter can count only when the signal MEM\_ENA is asserted. The control signal MEM\_ENA is asserted at the end of T1 for command NI while at the end of T9 for commands ED, PRC, PGC, and PBC.

#### 4.3.4. Execution of Command NI

From Table 4.2 we can see that z-depth values are initialized before color intensities. Two flip-flops NIFF1 and NIFF2 are incorporated to ensure this processing order. First the signal ZD\_INIT is asserted for 64 clock cycles to allow the initialization of the z-depth. Then the signal CLR\_INIT is also asserted for 64 clock cycles to allow the initialization of color intensities.



#### 4.4. Image Scan-Out

There are two sets of color intensity buffers:

1. (RB1, GB1, BB1)
2. (RB2, GB2, BB2)

While one set is being updated, the other set can be scanned out to refresh the display. The roles of two sets are interchanged by the command 'NI'. The command 'NI' is broadcasted to all SCPs during the interval of vertical fly-back.

##### 4.4.1. Image Buffer Access Control

The ALU unit can write but not read the image buffers whereas the Scan-Out Controller can read but not write the image buffers. The read and write of the image buffers are controlled by following control signals:

- WB1 : If WB1 is true, the buffer 1 can be written into.
- WB2 : If WB2 is true, the buffer 2 can be written into.
- SC1 : If SC1 is true, the buffer 1 can be scanned out.
- SC2 : If SC2 is true, the buffer 2 can be scanned out.

The above signals are related by:

$$WB1 = \overline{SC1} = \overline{WB2} = SC2$$

and generated by the circuit shown in Figure 4.3. FF1 is set and FF2 is reset when the graphics system is initialized. Whenever command NI is executed, Q1 and Q2 are swapped and hence the roles of the two sets of buffers are interchanged.

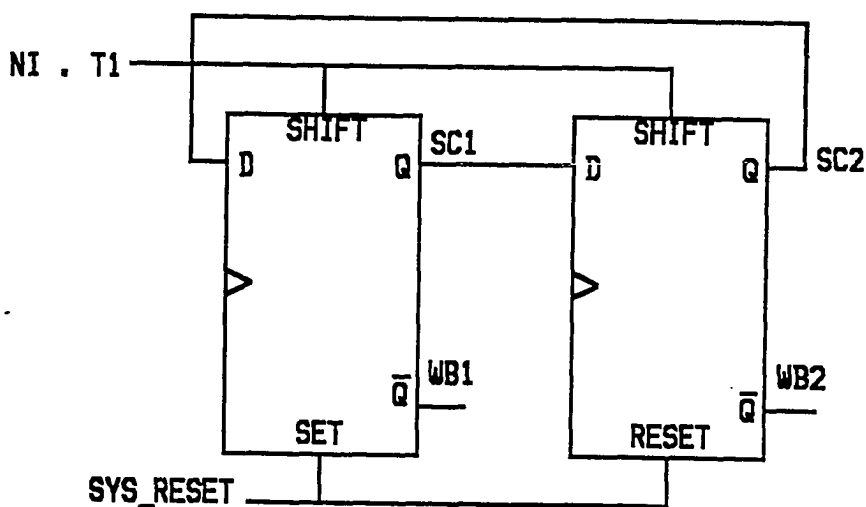


FIGURE 4.3. Buffer select circuit

#### 4.4.2. Scan-Out Control

The scan-out operation is controlled by a global token chain and a local token ring. The global token chain is an 8-bit shift register while the local token ring is a 64-bit

shift register with the output of the 63-th flip-flop connected to the input of the 0-th flip-flop. Pixels can be scanned out only when the global token resides on the SCP chip. The pixel to be scanned out is selected by the local token. The inclusion of the local token eliminates the need of external address pins for pixel selection.

#### 4.4.3. Scan-Out Timing

The global token is latched into the first flip-flop of the global token chain during  $\psi_1$ , and shifted to the second flip-flop during the second  $\psi_1$ , and to the third flip-flop during the third  $\psi_1$ , etc. The global token stays at one SCP for 8  $\psi$  clock cycles at a time, and thus allows 8 pixels to be scanned out.

A local token is generated during system initialization and latched onto the 0-th flip-flop of the local token ring. The local token is shifted from one flip-flop to the next flip-flop at the end of  $\psi_1$  when global token is on the chip. The local token shifts 8 flip-flops during the global token residency. In this way, pixels can be scanned out in a correct order.

The block diagrams of the global token chain and the local token ring are shown in Figure 4.4 and Figure 4.5 respectively. The latching of processor ID is also controlled by the global token. However, the global token

stays at one SCP for only 2  $\phi$  clock cycles for the latching of processor ID. As shown in Figure 4.4, two flip-flops are incorporated to satisfy this requirement. During system initialization, these two flip-flops are reset. When the global token resides on one SCP, the XID will be latched if the signal L\_XID is asserted and the YID will be latched if the signal L\_YID is asserted. The global token follows the right branch during the execution of command INIT so that the processor ID can be latched. In other situations, the global token follows the left branch.

#### 4.4.4. Memory Circuit

The 4-transistor static memory designed by Lyon and Schneider [12] is used as the basic memory cell of the color intensity and z-depth buffers. According to Lyon and Schneider [12], this design can achieve the access time between 25 and 50 ns. Lyon and Schneider have achieved the access time of 50 ns using the 3.5  $\mu\text{m}$  CMOS technology.

#### 4.5. A-BUS Transfer

All the data transfers between the ALU and the Z-depth and image buffers are via the A-BUS. The A-BUS is pre-charged [13] to high during  $\phi_1$  and all the data transfers are performed during  $\phi_2$ . The 1-bit circuit example of the A-BUS is shown in Figure 4.6.

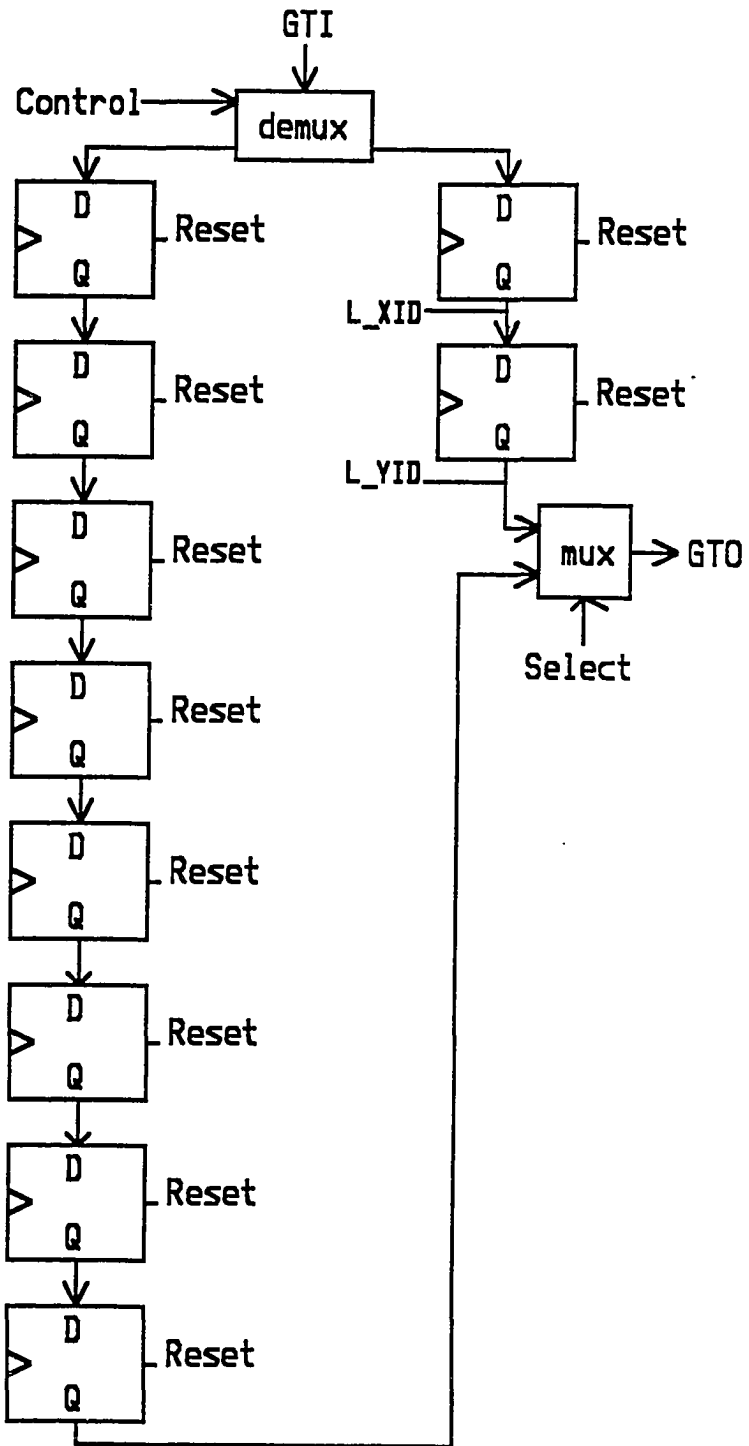
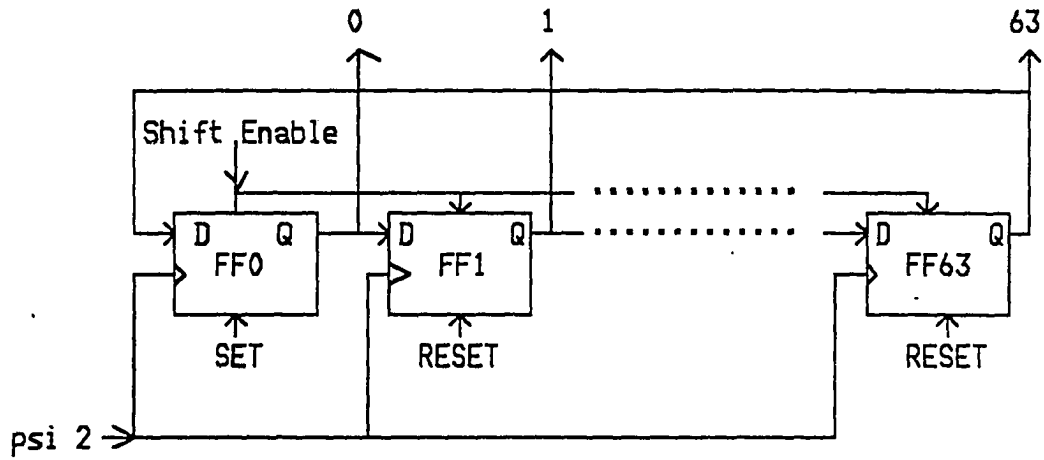


FIGURE 4.4. A block diagram of the global token chain



SET = SYS\_RESET

RESET = SYS\_RESET

Shift Enable is TRUE when Global Token is on chip

FIGURE 4.5. A block diagram of the local token ring

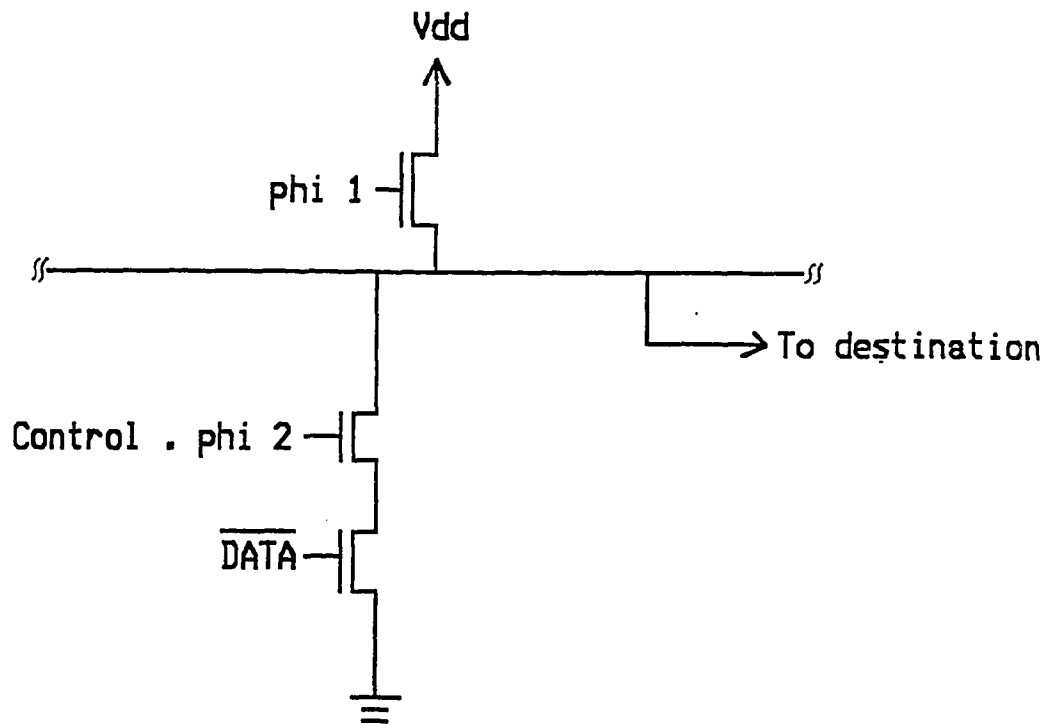


FIGURE 4.6. The 1-bit circuit example for A-BUS

## 5. LAYOUT AND PERFORMANCE EVALUATION

### 5.1. Design Environment

#### 5.1.1. CAD Tools

The CAD tool available to us is the 'ELECTRIC' from Schlumberger. The ELECTRIC provides a graphics editor, a one dimensional compactor, a design rule checker, a river router and interfaces to a couple of switch level simulators and the SPICE. The ELECTRIC supports several different technologies including NMOS and CMOS. Due to the time limit, we only used the ELECTRIC to do the mask layout.

#### 5.1.2. Simulators

The ELECTRIC provides interfaces to several simulators including ESIM, RSIM, RNL, MOSSIM, and SPICE. ESIM, RSIM, RNL and MOSSIM are switch level simulators [6] widely used in universities. These switch level simulators can perform timing simulation for a circuit with up to 50,000 transistors in reasonable computer time. The SPICE program is suitable for the timing simulation for a circuit with not more than several hundred devices because it uses very detailed models to calculate the circuit responses. The ELECTRIC can generate input for these simulators. However, none of them are available to us. Without these simulators we can only verify the correctness of our design by



observation. Our performance evaluation is based on estimation.

### 5.1.3. Technology Used

We decided to use Grieswold's CMOS technology [11] to design the SCP chip. Grieswold's CMOS technology uses  $4\ \mu\text{m}$  design rule, i.e., the minimum line width is  $4\ \mu\text{m}$  and the  $\lambda$  value is  $2\ \mu\text{m}$ . Mead and Conway [13] proposed the use of a single parameter  $\lambda$  to express the design rule in which  $\lambda$  is roughly equal to half of the minimum line width. One layer of metal and one layer of polysilicon are provided for interconnection. The one layer metal and one layer polysilicon limit the flexibility of interconnection, and also increase the chip size significantly due to the fact that more than 60 percent of the chip area are used for interconnection [3,14]. The state-of-the-art CMOS technology uses  $1.2\ \mu\text{m}$  line width and provides at least two layers of metal and one layer of polysilicon for interconnection. We made such a choice only because the ELECTRIC does not provide enough documentation and Grieswold's CMOS technology is the only CMOS technology that works fine for us.

## 5.2. Layout Design

About five months was spent on the layout. The SCP chip uses about 30,000 transistors. The chip size is about  $8000 \times 4000\lambda$  which is at the upper bound of the chip size. In our estimation, the chip size can be reduced from 15% to 20% if the second layer of metal is available for interconnection using the same technology. The chip size can be reduced further by 30% if a 2-D compactor similar to ZORRO [6] is available to us. We expect 16 SCPs can be put on one chip if the  $1 \mu\text{m}$  technology is used. The SCP chip consists of four major cells, i.e., ALU, SEQUENCER, SCAN-OUT CONTROLLER and memory buffer.

The ALU consists of 1 multiplier, two carry lookahead adders ADD1 and ADD2, and registers XID, YID, A, B, F1, F2, ZDL, and EL. The multiplier is on the top of the ALU. The ADD1 is in the middle and the ADD2 is at the bottom of the ALU. XID and YID are on the top of the multiplier. A and B are at the left of the multiplier. F1 and ZDL are located between the multiplier and the adder ADD1. F2 is located between ADD1 and ADD2.

The SEQUENCER consists of the command register, the command decoder, a 9-bit shift register, a 6-bit counter, the address decoder and some other circuits.

The memory buffer consists of Z-depth buffer, RB1, RB2, GB1, GB2, BB1, and BB2 from top to bottom. The read-write interface and the word line driver circuit are also in this cell.

The SCAN-OUT CONTROLLER consists of the global token chain and the local token ring.

### 5.3. Performance Evaluation

In this section we will estimate the highest clock rate under which our SCP chip can operate. In doing the estimation, we make following assumptions:

- 1  $\mu\text{m}$  CMOS technology is used.
- Two layers of metal and one layer of polysilicon are available for interconnection.
- All the control signals and power are distributed via metal layers so that the RC wiring delay are minimized.
- The polysilicon layer is only used for local signal distribution and signal crossover.

We do not estimate the performance of the SCP chip based on the technology which we used mainly because the Grieswold's CMOS technology is already out-of-date. According to Weste and Eshraghian [19], the RC wire delay can be ignored if the length of signal path in any layer is

less than the value indicated in Table 5.1. Also, the influence of first-order scaling on MOS devices is shown in Table 5.2 where  $a$  is the scaling factor ( $a > 1$ ).

TABLE 5.1. Guidelines for ignoring RC wiring delays

LAYER	MAXIMUM LENGTH
Metal	$20,000\lambda$
Silicide	$2,000\lambda$
Polysilicon	$200\lambda$
Diffusion	$20\lambda$

TABLE 5.2. Influence of first-order scaling on MOS device characteristics

PARAMETER	SCALING FACTOR
supply voltage	$1/a$
parasitic capacitance	$1/a$
gate delay	$1/a^2$
DC power dissipation	$1/a^2$
dynamic power dissipation	$1/a^2$

There are two steps in our method:

1. Identifying the critical signal propagation path.
2. Estimating the propagation delay and gate delay.

### 5.3.1. Critical Signal Propagation Path

After reviewing the timing of eight commands and the layout, we conclude that following gates and signal paths are critical:

- **Memory buffer:** All the memory components including z-depth, 2 sets of color intensity buffers must be accessed within one  $\phi$  clock phase. The longest word line is about  $2900\lambda$ .
- **Carry lookahead adder ADD1:** The 24-bit carry lookahead adder ADD1 is critical because it must perform one addition in one  $\phi$  clock phase. The register A is one of ADD1's sources and its distance to ADD1 is about  $1800\lambda$ .
- **ENA array:** All enable flags must be accessed in one  $\phi$  clock cycle.

The multiplier looks like a critical signal propagation path. The gate delay of the multiplier is equal to the sum of delays of 5 4-bit carry lookahead adders, 1 half adder and 2 full adders. This delay is just a little bit longer than the 24-bit carry lookahead adder. We conclude that the multiplier gate delay is not critical because we can allow 2 clock cycles to perform the multiplication and only deteriorate the performance by 2.5 percent.

The 24-bit carry lookahead adder ADD2 is not critical because it performs addition every 8 clock cycles. The register B is one of ADD2's source. The distance from register B to ADD2 is about  $2400\lambda$  which will cause only negligible RC wiring delay. The content of register B does not change during the interval of execution of any specific command.

Inside the SEQUENCER, all interconnection wires are not longer than  $2000\lambda$  and hence cause only negligible RC wiring delay. The 6-bit counter is not critical because its gate delay is the sum of two 2-input NOR gates and two inverters which is much less than the delay of a carry lookahead adder.

### 5.3.2. Estimation of Gate Delay

As we mentioned in Chapter 4, the access time of the static memory cell is 50 ns using the  $3.5 \mu\text{m}$  CMOS technology. We expect the access time can be reduced to 12.2 ns if the  $1 \mu\text{m}$  CMOS technology is used because the scaling factor is 3.5. The propagation delay of the decoder circuit is included because we use the same decoder as Lyon's design [12].

To estimate the gate delay of the 24-bit carry lookahead adder ADD1, we compare it with the AMD29000 RISC microprocessor from Advanced Micro Devices Corporation. The

AMD29000 uses 1.2  $\mu\text{m}$  CMOS technology and can perform 32-bit addition within 40 ns. We expect the SCP can perform 1 24-bit addition in less than 25 ns using the 1  $\mu\text{m}$  CMOS technology.

The ENA array consists of D-type master-slave flip-flops. It is accessed via a read and a write bus lines. Its access time is expected to be similar to the memory buffer.

### 5.3.3. Estimated Throughput

From the analysis of the previous section, we find that the longest gate delay is less than 25 ns. Since our design uses the 2-phase clocking scheme, each operation must be finished in one clock phase. The highest clock rate under which the SCP can operate will be 20 MHz.

With the highest clock rate determined we can estimate the throughput of the Raster Graphics system that uses our SCP chips. The throughput is expressed in terms of the number of polygons that can be processed per second. Our performance estimation is based on following assumptions:

- The screen is divided into 16 areas so that 16 polygons can be processed simultaneously.
- All polygons are 4-sided so that 4 ED commands are processed for each polygon.

The processing time of one polygon is shown in Table 5.3:

TABLE 5.3. Processing Time of one Polygon

OPERATION	# OF CLOCK CYCLES
1 NP command	2
4 ED commands	296
1 ZDC command	138
1 PRC command	74
1 PGC command	74
1 PBC command	74
TOTAL TIME	658

Among these 658 clock cycles, 68 cycles are spent on the latching of commands and information needed in the command execution. Comparing these two numbers, we are sure that many polygons can be processed simultaneously.

Since 16 polygons can be processed at the same time the maximum throughput is estimated to be 486,322 polygons per second. This throughput is much higher than what is needed in rendering the 3-D images in real time. Usually, this peak throughput can not be achieved due to the fact that one polygon can have intersection with several subareas on the screen. The actual throughput will be application dependent and can only be obtained by simulation.

Our SCP chip is designed to support the resolution of up to 2048 x 2048 pixels. At the resolution of 1024 x 1024 pixels, 16,384 SCPs are needed. Since 16 SCPs can be put on one chip using 1  $\mu\text{m}$  CMOS technology, only 1024 SCP chips are



needed. At the current technology, one printed circuit board can hold from 120 to 150 IC chips. Seven to 9 printed circuit boards will be needed to hold the SCP chips in the graphics system. The Raster Graphics System that uses our SCP chips needs only two new custom-designed chips. The system cost is not high.

## 6. DISCUSSION AND FUTURE WORK

In this dissertation research, a Scan Conversion Processor to eliminate the bottleneck in the image generation process was invented and designed. According to our estimation, the Raster Graphics System that uses our SCP chips will be able to process up to 480,000 polygons per second if the 1  $\mu$ m CMOS technology is used. Although only 8 commands are implemented, they are enough for the 3-D image rendering. We can add a few more commands to enhance the capability of our SCP chip. Due to the fact that adding a new command will increase the device count, one important consideration in adding new commands to the SCP is that the device count increase must be kept to the minimum; otherwise, the chip size will increase significantly. If the chip size increases to a degree that reduces the number of SCPs that can be put on one chip, the system cost will be doubled or even quadrupled.

Since our SCP chip can perform the hidden surface elimination and smooth shading in hardware, we expect the software overhead on the hidden surface elimination and smooth shading can be reduced to the minimum. The software on the geometry transformation is not affected. The software to adapt the Raster Graphics System to our SCP design is not significant because not many new transformations need to be done.

Our SCP chip uses polygons to approximate the surface of all possible geometry. Some curved surfaces are not best represented by polygons, for example, spheres and cones. According to Fuchs and Goldfeather [8], the z-depth and color intensity of a sphere can also be represented by linear equations if the quadratic term of  $x^2+y^2$  is stored in the memory beforehand. However, that will increase the device count by about 5200. Some application such as the molecule modelling will generate images mainly consisting of spheres. It may be worthwhile to add such a capability for this application.

Our future work will be to explore the possible capability enhancements to the SCP chip under the constraint that it evaluates only the linear functions.

As we mentioned in chapter 3, there is a Bounds Checking Processor in our target Raster Graphics System. This Processor will consist of one or more comparators, a queue of some size to store the incoming commands and information, and a counter to keep track of whether its controlled SCPs are busy. The size of the queue need to be decided so that the chip size is minimized and at the same time the throughput is maximized. We plan to continue on the design of this processor.

## 7. BIBLIOGRAPHY

1. G. D. Abram and H. Fuchs. VLSI Architecture for Computer Graphics. Pp. 189-204 in Advances in Computer Graphics I. Berlin, West Germany: Springer Verlag, 1986.
2. Marco Annaratone. Digital CMOS Circuit Design. Norwell, Massachusetts: Kluwer Academic Publishers, 1987.
3. M. Burstein. "Channel Routing." Chapter 4 of Layout Design and Verification. Amsterdam, Holland: North Holland, 1986.
4. J. Clark and M. Hannah. "Distributed Processing in a High-performance Smart Image Memory." VLSI Design 1, No. 3 (4th quarter 1980): 59-68.
5. Stefan Demetrescu. Moving Pictures. Byte 10 (Nov. 1985): 207-217.
6. Wolfgang Fichtner and Martin Morf. VLSI CAD Tools and Applications. Norwell, Massachusetts: Kluwer Academic Publishers, 1987.
7. J. D. Foley and Van Dam. Fundamentals of Interactive Computer Graphics. Reading, Massachusetts: Addison Wesley, 1982.
8. H. Fuchs, J. Goldfeather, J. P. Hultquist, S. Spach, J. D. Austin, F. P. Brooks, J. G. Eyles and J. Poulton. Fast Spheres, Shadows, Textures, Transparencies, and Image Enhancements in Pixel-Planes. In Advances in Computer Graphics. Berlin, West Germany: Springer Verlag, 1986.
9. H. Fuchs, J. Poulton, A. Paeth and A. Bell. "Developing Pixel Planes, A Smart Memory-Based Raster Graphics System." Pp. 207-217 in Proceedings, Conference on Advanced Research in VLSI, 1982.
10. N. Gharachorloo and Christopher Pottle. "Super-Buffer: A Systolic VLSI Graphics Engine for Real Time Raster Image Generation." Pp. 285-305 in 1985 Chapel Hill Conference on VLSI. Rockville, Maryland: Computer Science Press, 1985.

11. Thomas W. Grieswold. "Portable Design Rules for Bulk CMOS." VLSI Design 3, No. 5 (September/October, 1982): 62-67.
12. R. F. Lyon and R.R. Schneider. CMOS Static Memory with a New Four-Transistor Memory Cell. Pp. 111-132 in Proceedings of the 1987 Stanford Conference on Advanced Research in VLSI, 1987.
13. Carver Mead and Lynn Conway. Introduction to VLSI System. Reading, Massachusetts: Addison Wesley, 1980.
14. T. Ohtsuki. "Maze-Running and Line-Search Algorithms." Chapter 3 of Layout Design and Verification. Amsterdam, Holland: North Holland, 1986.
15. J. Poulton, H. Fuchs, J. Austin, J. Eyles and Trey Greer. "Building a 512 x 512 Pixel-Plane System." Pp. 57-74 in Proceedings of the 1987 Stanford Conference on Advanced Research in VLSI, 1987.
16. J. Poulton, H. Fuchs, J. D. Austin, J. G. Eyles, J. Heinecke, C. H. Hsieh, J. Goldfeather, J. P. Hultquist and S. Spach. "Pixel-Planes: Building a VLSI-Based Graphics System." Pp. 35-60 in 1985 Chapel Hill Conference on VLSI. Rockville, Maryland: Computer Science Press, 1985.
17. R. F. Sproul, Ivan E. Sutherland, Alistair Thompson, Satish Gupta and Charles Minter. "The 8 by 8 Display." ACM Transactions on Graphics, 2, No.1 (January 1983): 32-56.
18. W. Strasser. "VLSI-Oriented Graphics System Design." In Advances in Computer Graphics II. Berlin, West Germany: Springer Verlag, 1986.
19. Neil Weste and Kamran Eshraghian. Principles of CMOS VLSI Design: A Systems Perspective. Reading, Massachusetts: Addison Wesley, 1985.

## 8. ACKNOWLEDGEMENTS

I would like to express my gratitude to my major professor, Dr. Arthur Pohm, for his encouragement and guidance during this research. His technical advice and judgement were invaluable for my work.

Special thanks are extended to Dr. Terry Smay, Dr. Harrington Brearley, Dr. V. Vittal, Dr. A. E. Oldehoeft and Dr. J. A. Davis for giving suggestions and guidance as my graduate committee. I also like to express my appreciation to Dr. Charles Wright for his help in acquiring the VLSI design software 'ELECTRIC' which is very valuable in my research.

I am grateful for the financial support I received from the Department of Electrical Engineering and Computer Engineering at Iowa State University.

I wish to thank my parents for their love and encouragement. My father always taught me the value of education and honesty. Finally, I am grateful to my wife Su-Jane for her continued faith and help throughout the writing of this thesis.