

AD-A149 427 THE FEASIBILITY OF A MULTIPROCESSOR ARCHITECTURE FOR
REAL-TIME CONTOUR SURFACE DISPLAY GENERATION(U) NAVAL
POSTGRADUATE SCHOL MONTEREY CA M J ZYDA DEC 84
UNCLASSIFIED NP552-84-025 F/G 9/2

THE FEASIBILITY OF A MULTIPROCESSOR ARCHITECTURE FOR
REAL-TIME CONTOUR SURFACE DISPLAY GENERATION(U) NAVAL
POSTGRADUATE SCHOL MONTEREY CA M J ZYDA DEC 84
NPS52-84-025 F/G 9/2

1/1

UNCLASSIFIED

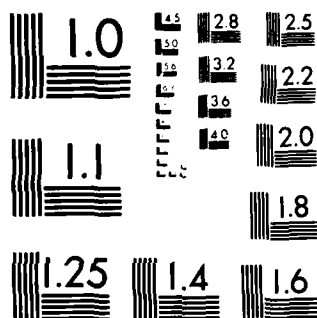
F/G 9/2

NL

END

FILMS

DTWC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

AD-A149 427

NPS52-84-025

NAVAL POSTGRADUATE SCHOOL

Monterey, California



SECRET
JAN 16 1985
A

THE FEASIBILITY OF A MULTIPROCESSOR ARCHITECTURE
FOR REAL-TIME CONTOUR SURFACE DISPLAY
GENERATION

Michael J. Zyda

December 1984

Approved for public release; distribution unlimited

Prepared for:

Chief of Naval Research
Arlington, VA 22217

85 01 10 025

ONE PAGE COPY

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Commodore R. H. Shumaker
Superintendent

D. A. Schrady
Provost

The work reported herein was supported in part by the Foundation Research Program of the Naval Postgraduate School with funds provided by the Chief of Naval Research.

Reproduction of all or part of this report is authorized.

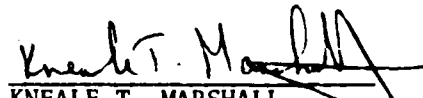
This report was prepared by:


MICHAEL J. ZYDA
Assistant Professor of
Computer Science

Reviewed by:


BRUCE J. MACLENNAN
Acting Chairman and Associate
Professor of Computer Science

Released by:

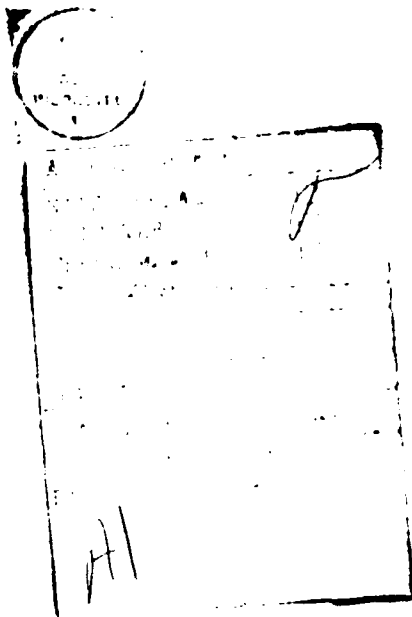

KNEALE T. MARSHALL
Dean of Information and
Policy Sciences

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NPS52-84-025	2. GOVT ACCESSION NO. AD-A149427	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) The Feasibility of a Multiprocessor Architecture for Real-Time Contour Surface Display Generation		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Michael J. Zyda		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, CA 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61152N; ER000-01-NP N0001485WP41005
11. CONTROLLING OFFICE NAME AND ADDRESS Chief of Naval Research Arlington, VA 22217		12. REPORT DATE December 1984
		13. NUMBER OF PAGES 27
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) contouring, contouring tree, contour surface display generation, real-time display generation		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) We present in this study the architectural specification and feasibility determination for a real-time contour surface display generator. We begin by selecting for architectural implementation a recently reported, highly decompos- able algorithm for contour surface display generation. We establish a piece of the total algorithm as the algorithm component. The algorithm component is that part of the algorithm that can be executed in parallel, independently from the computations performed on any other algorithm subpart. We propose an architecture for the algorithm component, and model that architecture in order to determine		

unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

the real-time capability of the algorithm. We then model the larger system of multiple algorithm component processors. This modeling effort is performed with respect to a particular application requiring real-time contour surface display generation. A VLSI feasibility computation is then performed on the proposed architecture. The study ends with a look at the impact of real-time contour surface display generation on the design of the graphics display system.



S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

The Feasibility of a Multiprocessor Architecture for Real-Time Contour Surface Display Generation ‡

Michael J. Zyda

Naval Postgraduate School,
Code 52, Dept. of Computer Science,
Monterey, California 93943

ABSTRACT

We present in this study the architectural specification and feasibility determination for a real-time contour surface display generator. We begin by selecting for architectural implementation a recently reported, highly decomposable algorithm for contour surface display generation. We establish a piece of the total algorithm as the algorithm component. The algorithm component is that part of the algorithm that can be executed in parallel, independently from the computations performed on any other algorithm subpart. We propose an architecture for the algorithm component, and model that architecture in order to determine the real-time capability of the algorithm. We then model the larger system of multiple algorithm component processors. This modeling effort is performed with respect to a particular application requiring real-time contour surface display generation. A VLSI feasibility computation is then performed on the proposed architecture. The study ends with a look at the impact of real-time contour surface display generation on the design of the graphics display system.

Categories and Subject Descriptors: I.3.1 [**Hardware Architecture**]: architectures, parallel processing, VLSI implementations; I.3.2 [**Graphics Systems**]: multiprocessing systems; I.3.3 [**Picture/Image Generation**]: surface visualization; I.3.5 [**Computational Geometry and Object Modeling**]: data structures, discrete planar contours, modeling molecules, surface approximation, surface generation, surface representation, surfaces, 3D graphics; I.3.6 [**Methodology and Techniques**]: contouring, interactive systems, parallel processing; I.3.7 [**Three-Dimensional Graphics and Realism**]: line drawings, line generation algorithms, real-time graphics, surface plotting, surface visualization, surfaces; I.3.m [**Miscellaneous**]: VLSI;

General Terms: Algorithms, architecture;

Additional Key Words and Phrases: contouring, contouring tree, contour surface display generation, real-time display generation;

‡ This work has been supported by the NPS Foundation Research Program.

1. Introduction

Contour surface display generation is one of the most frequently used graphics algorithms [Barry,1979], [Faber,1979], [Wright,1979], [Zyda,1984a], [Zyda,1984b], [Zyda,1984c], [Zyda,1983], [Zyda,1982], [Zyda,1981]. A contour surface display is a visual representation of a surface by the collection of lines formed when that surface is intersected by a set of parallel planes. The lines formed on each of those planes are called contours. A contour represents the set of points that belong to both the surface and the particular intersecting plane. Contour surface displays are used in X-ray crystallography, computer-aided tomography, and other applications for which grid data is collected. Contour surface display generation is generally depicted as a computationally slow operation whose output is sent to a plotter or film recorder. A number of papers have been written documenting "breakthroughs" that increase the speed of contour surface display generation. One author has reported that his contour surface display generation subroutine used one second of central processor time on NCAR's Control Data 7600 [Wright,1979]. Although a contour surface display generation program of this speed is useful for static situations, it is found to be lacking for interactive applications that generate a succession of contour surface displays in response to contour level changes read from a control dial. Such a program requires that a new contour surface display be generated, distributed, and displayed in real-time, typically one-thirtieth of a second for current display technology [Newman,1979].

One application in which real-time contour surface display generation is important is the determination of molecular structures from the electron density data generated by X-ray crystallography [Barry,1979]. Such an operation is executed interactively by using a computer graphics program that displays a Dreiding (stick) model of the molecule, inside a contour surface display of the corresponding region of the molecule's electron density grid. In addition to the graphics function, the computer program monitors a series of signals generated by the user, while the user is turning the various knobs on a control console [Zyda,1980]. The values read from these knobs are interpreted by the program as modifications to either the molecule or the surface display. Modifications to the molecule take the form of bond rotations or bond lengthenings. Modifications to

the contour surface display take the form of an increase or decrease of the contour level. The goal of this process is to produce the stick model of the molecule that best fits inside the given electron density data set. The user can determine whether or not the model fits the density grid by modifying the contour level, shrinking the contour surface to the molecule. Similarly, the user can expand the contour surface from the stick model for better visibility. This function requires that the hardware have the capability to rapidly change the contour display as its contour level changes.

We know from [Zyda,1984a] that the generation of a contour surface display, such as those required by the above application, cannot be accomplished in real-time using a conventional uniprocessor. This failure is due to the fact that contour surface display generation algorithms require many more instructions executed per second than can be provided by currently available uniprocessors. In the past, this limitation of the conventional processor has relegated such applications to either the non real-time environment (waiting a few minutes for each display), or to the equally unsatisfying environment of motion picture film. Because of this, this study looks for multiprocessor solutions to the real-time contour surface display generation problem. At the present time, efficient multiprocessor solutions generally mean VLSI solutions. Consequently, the multiprocessor architectures examined in this study are those implementable in the VLSI technologies.

2. Contouring Definitions

A contour surface is a visual display that represents all points in a particular region of three-space $\langle x,y,z \rangle$ which satisfy the relation $f(\langle x,y,z \rangle)=k$, where k is a constant known as the contour level. The function f represents a physical quantity which is defined over the three-dimensional volume of interest. The visual display created by this algorithm is the collection of lines that belong to the intersection of both the set of points that satisfy the relation $f(\langle x,y,z \rangle)=k$, and a set of regularly spaced parallel planes that pass through the region of three-space for which the relation is defined.

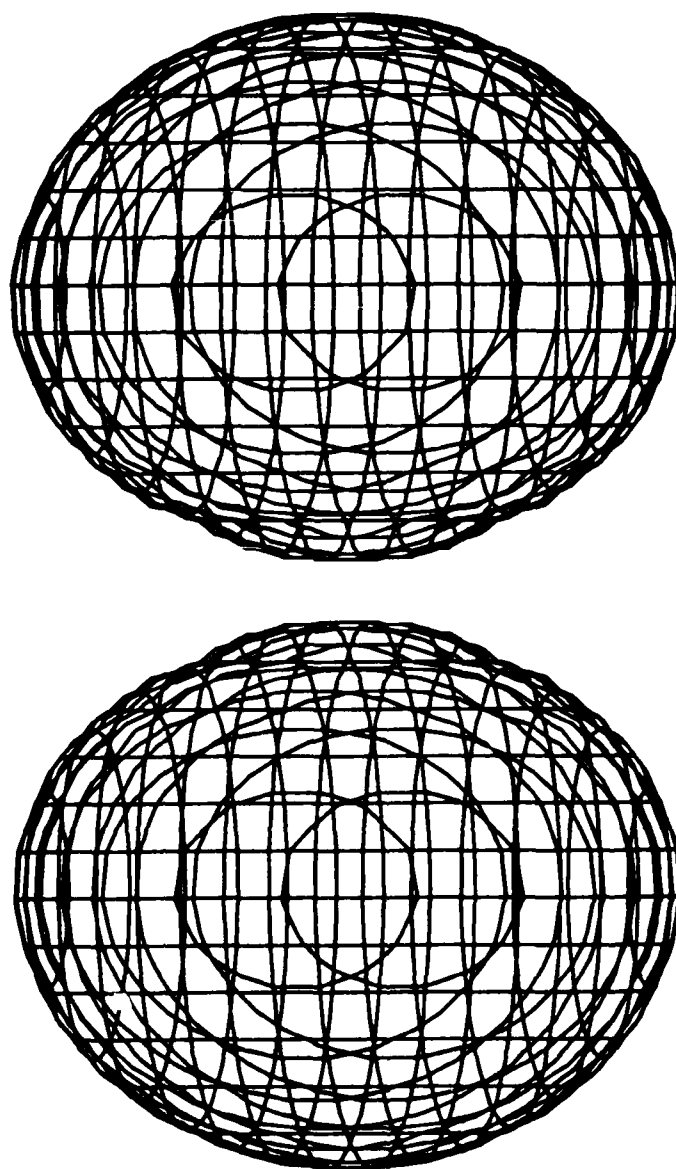


Figure 1
Contour Surface Display Generated from a Hydrogen Atom
Wavefunction Squared (3dxy orbital)

For this study, the function f is approximated by a discrete, three-dimensional grid created by sampling that function over the volume of interest. The three-dimensional grid contains a value at each of its defined points that corresponds to the physical quantity obtained from the function, i.e. the value associated with point (x_0, y_0, z_0) is v_0 , where $f(x_0, y_0, z_0) = v_0$. In order to minimise confusion, we will specify the value at a particular grid point (x, y, z) by $a(x, y, z)$, and will specify the value at a particular point (x, y, z) of the function by $f(x, y, z)$.

A decomposable algorithm for contour surface display generation has been described in [Zyda, 1984b]. That algorithm is constructed from a two-dimensional contouring algorithm that is used to contour all the possible planar, orthogonal, two-dimensional grids of a larger three-dimensional grid. The two-dimensional contouring algorithm of that paper is comprised of components, called algorithm components, that operate on individual 2×2 subgrids of a larger two-dimensional grid. (Note: a 2×2 subgrid is defined to be that portion of the two-dimensional grid bounded by four adjacent grid points.) In the algorithm, the computations necessary for generating the contour lines for a single 2×2 subgrid are independent from those required for any other 2×2 subgrid. If we compute the contours corresponding to contour level k for all 2×2 subgrids of a two-dimensional grid, then we will have determined the complete set of contours for that grid. If we compute the contours corresponding to contour level k for all possible 2×2 subgrids of the larger three-dimensional grid, then we will have the complete contour surface display for that grid. The assemblage of the contours created by this process, i.e. the simultaneous display of all the contours created for all 2×2 subgrids of the larger three-dimensional grid, produces a "chicken-wire-like" contour surface display (see Figure 1). The full development of this algorithm can be found in [Zyda, 1984a], [Zyda, 1984b], and [Zyda, 1984c]. We will refer to the results of those studies, and consequently, will not cover the algorithm here in great detail.

3. Architectural Modeling

The architectural modeling necessary to determine if a VLSI multiprocessor for real-time contour surface display generation is feasible is accomplished in two steps. The first step is the modeling of the algorithm component, or 2×2 subgrid, level (see Figure 2). The purpose of this

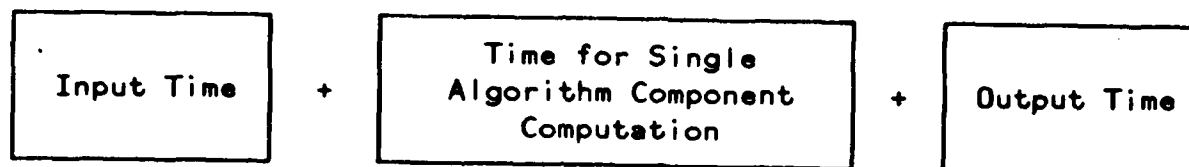


Figure 2
Algorithm Component Architectural Model

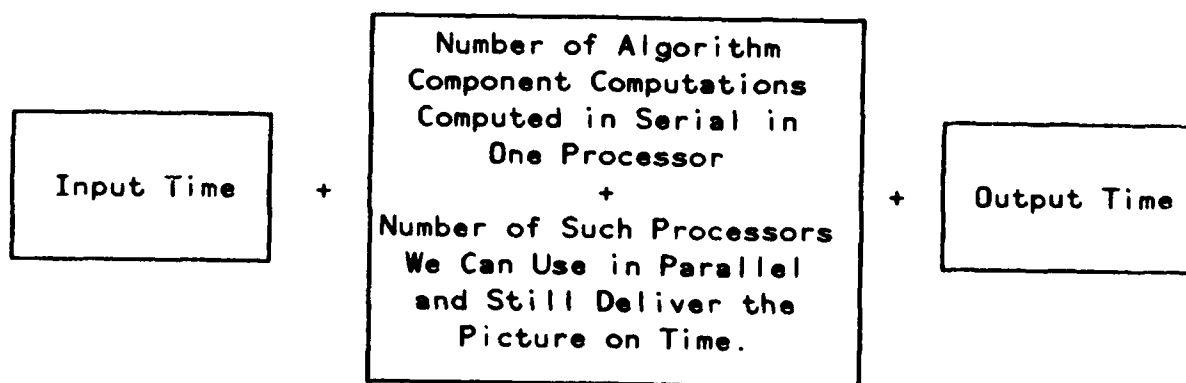


Figure 3
Total System Architectural Model

step is to determine if the amount of code specified for the algorithm component computation is executable in real-time. In this step, an implementation of the algorithm component is analyzed. The analysis is performed in the context of a processor whose characteristics are similar to those of a general purpose microprocessor, the MC68000. The model constructed is a register transfer model of the algorithm component. In this model, the memory references that are made for each instruction's operation and for each operand's retrieval during the execution of the algorithm component are counted and recorded. Since the number of memory references a program makes is proportional to its run time, we only have to multiply by the amount of time a memory reference requires in order to obtain a measure of the real-time capability of the algorithm component processor ([Zyda,1981], [Zyda,1982], [Zyda,1983], [Zyda,1984a], [Aho,1974], and [Fuller,1977]). The value used in this study is 250 nsec per memory reference. This value is the slowest access time indicated for dynamic RAM (DRAM), and ROM chips announced over the last year in the IEEE journals *Computer*, and *Micro* (see Figure 4). Since there are access times indicated that are less than half that value, i.e. 70 nsec, we are conservative in the choice of 250 nsec as the time required to complete a memory reference.

The second step in the architectural feasibility modeling is the modeling of the total system of algorithm component processors (see Figure 3). The purpose of this step is to determine the total number of processors we can use in parallel, the load (number of algorithm components) per processor, and the total real-time capability of that system, i.e. the size of the largest three-dimensional grid for which we are able to generate the contour surface display in real-time. This part of the modeling effort extends the algorithm component modeling results to a model of the total system architecture for the real-time contour surface display generator. With the structure and real-time capability of the algorithm component processor established, we determine the capabilities of a system utilizing multiple copies of that processor. The parameters of the complete system modeled are derived from the requirements of the applications. The parameters utilized include such factors as the total size of the inputs and outputs, and the total number of algorithm components (and hence, the total number of algorithm component processors).

DRAM Chip Characteristics

Manufacturer	Chip	Chip Size	Access Time	Reference
AMD	AM9128	16K DRAM	70ns	Micro, Feb. 83
Mostek	MK45H64	64K DRAM	80ns	Micro, Aug. 83
				100ns
				120ns

ROM Chip Characteristics

Manufacturer	Chip	Chip Size	Access Time	Reference
Signetics	23256A	256K ROM	200ns	Computer, Jun. 83
Synertek	SY23128/A	128K ROM	200ns	Computer, Jul. 83
	SY23256/A	256K ROM	200ns	
American	S23128A	128K ROM	250ns	Computer, Jul. 83
Microsystems				

Figure 4
 DRAM and ROM Chip Access Time taken from 1983 issues
 of Computer, and IEEE Micro

3.1. Architecture for the Algorithm Component

We begin the description of the architecture for the algorithm component with an overview diagram (see Figure 5). In that figure the important architectural pieces of the processor and their interconnections are depicted. The pieces shown are found in most processors. Instead of giving a detailed specification for those pieces, we will discuss only the important size parameters, and important uses for that hardware when they differ from the norm found in processors of the MC68000 class.

In order to detail the sizes of the hardware elements in the figure, we first describe the operations expected of the algorithm component processor. There are only four: (1) reset the entire system of algorithm component processors, (2) accept a 2×2 subgrid description into a particular algorithm component processor, (3) place the coordinates generated for a particular 2×2 subgrid onto the system bus, and (4) generate the contours for the 2×2 subgrid held in the algorithm component processor. The first operation, the reset operation for the entire system of algorithm component processors, is clearly required. Computing systems are never constructed without some mechanism for providing a known initial state of the hardware.

The second operation, that of accepting a subgrid definition into a particular algorithm component processor, has implications for both the size of the RAM of the processor, and for the performance of its external communication mechanism. For that operation, the algorithm component processor needs to be able to recognize when a subgrid definition is addressed to it, and then needs to be able to store that information into its RAM. From [Zyda,1984a], we find that the size of the input to the algorithm component processor is 24 bytes. If we assume 32-bit transfers to the algorithm component processor, this is a total of 6 references per 2×2 for the input operation, requiring an equivalent amount of RAM storage.

The third operation, that of placing the coordinates generated in a particular algorithm component processor onto the system bus, has implications similar to that of the input operation. For the output operation, the algorithm component processor needs to be able to recognize when it should deposit its coordinates onto the system bus, and needs to be able to provide RAM

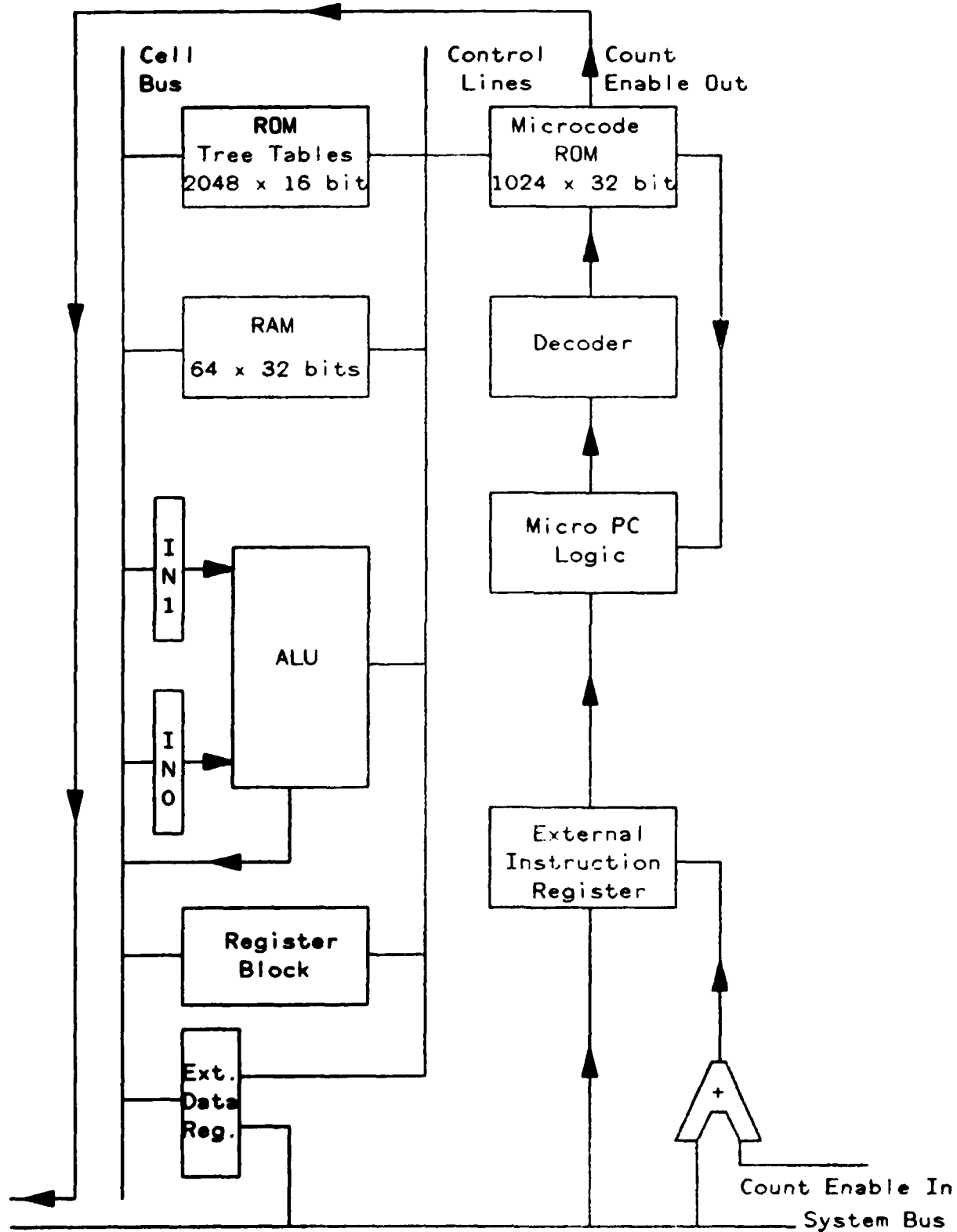


Figure 5
Block Diagram of the Algorithm Component Processor

storage for those output coordinates beforehand. From [Zyda,1984a], we know that the largest output that can be generated for a 2×2 subgrid is 6 coordinate and drawing instruction quadruples (78 bytes). If we count the byte indicating the number of coordinates output, we need to perform 20 32-bit transfers for the output operation, and need to provide an equivalent amount of RAM storage.

The fourth operation, that of generating the contours for the 2×2 whose definition is held in the algorithm component processor, effects the size of all the memories in the algorithm component processor. The algorithm in [Zyda,1984a] is based upon a data structure, the contouring tree, that is created for every 2×2 subgrid of the larger grid. A pre-order traversal procedure applied to each tree generates the coordinates and drawing instructions pertaining to the represented 2×2 subgrid for the selected contour level. If we use that algorithm, this means we need to provide space for the tree traversal list tables (2681 bytes), the algorithm component miscellaneous variables (45 bytes), and the code that performs the algorithm component computation (3080 bytes). (A comprehensive listing of all the data required in the algorithm component computation can be found in [Zyda,1984a], Figure 3.1.). The estimates for the input, output, tree traversal tables, and miscellaneous are derived directly from the data and data sizes required for the computation of the algorithm component. These values represent the space needed for registers, random-access and read-only memories. Rounding the microcode memory requirement, 3080 bytes, to a power of two, and assuming a horizontal microprogramming for the algorithm component processor, we find that a 1024 x 32-bit ROM is required. The tree traversal list memory requirement, 2681 bytes, becomes a 2048 x 16-bit ROM in the same fashion. The rest of the memory requirement for the algorithm component processor is shown in Figure 5 as a 64 x 32-bit RAM, which is used to hold the subgrid definitions, the coordinates generated, and any temporaries required to compute the algorithm component. We note at this point that the ROMs and RAMs specified are expected to consume the majority of the area on the VLSI chip.

3.2. Real-Time Capability of the Algorithm Component Processor

In order to determine if the amount of computation specified for the algorithm component is executable in real-time, one-thirtieth of a second, we need to put together a register transfer model of that algorithm and then to execute that model with the worst case inputs for the algorithm. As indicated above, a register transfer model counts the total number of memory references made by the algorithm component for both operation executions, and operand retrievals. There are four parts to the register transfer model of the algorithm component: (1) the input of the 2×2 subgrid to the algorithm component processor, (2) the output from the algorithm component processor, (3) the tree construction (traversal list indexing), and (4) the contour generation (traversal list usage). We obtain the memory reference counts for all four parts from [Zyda,1984a]. We find a total of 2676 memory references are required for (1) the input to the algorithm component processor (6 memory references), (2) the output from the algorithm component processor (20 memory references), (3) the tree construction, or traversal list indexing, for the 2×2 subgrid (602 memory references), and (4) the contour generation from the trees generated, or traversal lists indexed, (2048 memory references). We note that the fourth part of the register transfer model is obtained from the subgrid that generates the maximum number of coordinate and drawing instruction quadruples, 6 quadruples per 2×2 . We also note that for typical applications, the average number of coordinate and drawing instruction quadruples generated for the set of 2×2 subgrids that generate coordinates at all is 2.54 quadruples per 2×2 [Zyda,1984a]. At 250 nsec per reference, 2676 memory references require about 669 microseconds -- clearly under the one-thirtieth of a second (33,333 microseconds) goal we set for the algorithm component processor. In fact, given one-thirtieth of a second, we can accomplish about 50 algorithm component computations in serial.

4. Larger System of Multiple Algorithm Component Processors

The first issue of importance that must be covered when considering the design of the larger system is the issue of how operations and data are communicated. Figure 6 contains a view of the proposed interconnection scheme for the algorithm component processors. In that figure, each

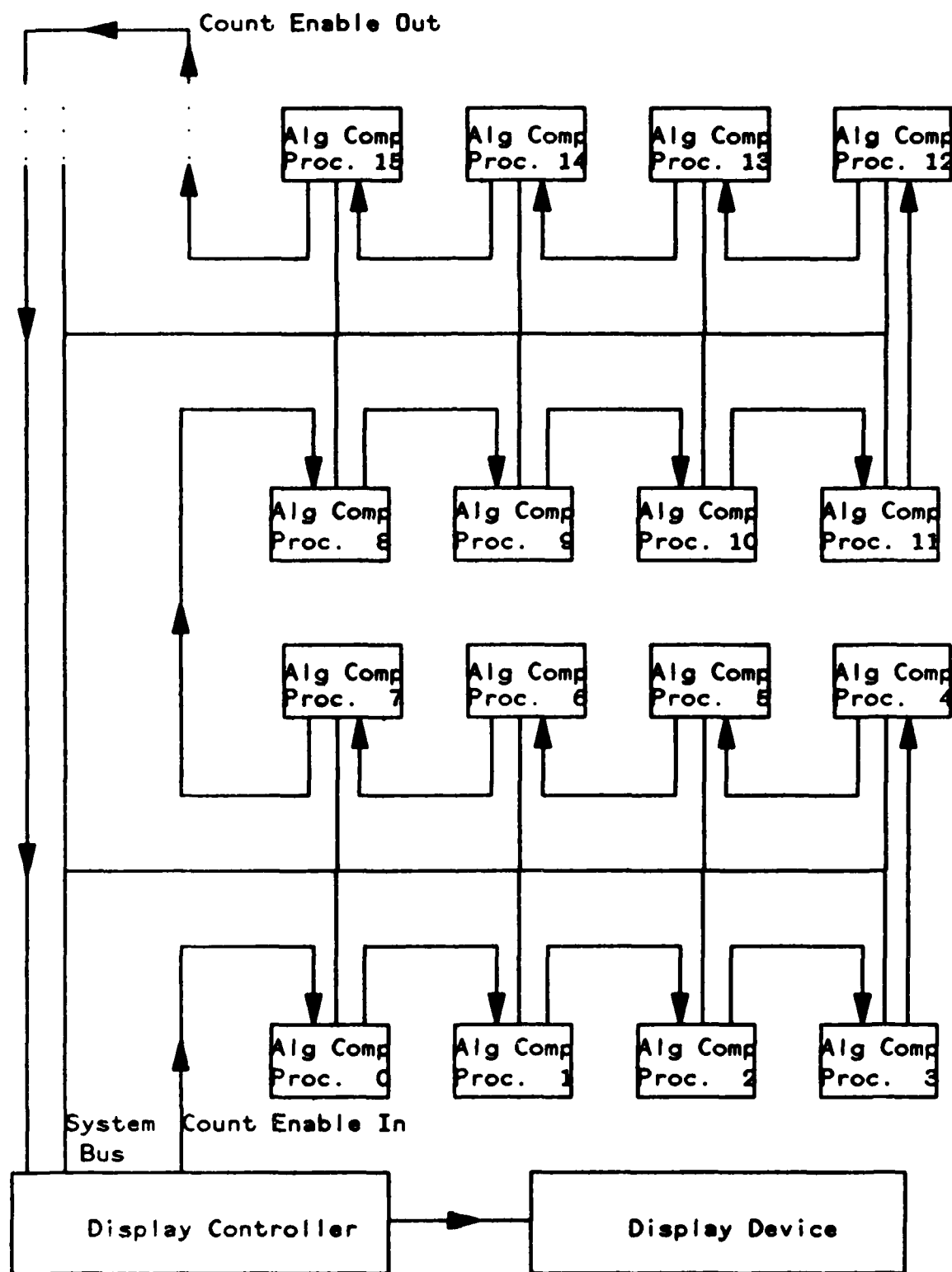


Figure 6
Multiple Algorithm Component Processor Interconnections

processor is depicted as being connected to a system bus, and a serial control line called the count-enable line. As indicated in Figure 5, the system bus provides both data and instructions to the algorithm component processor. It also provides the pathway for data output back to the display controller. Not so clear in that figure is the function of the count-enable line. The count-enable line is a one bit control line that runs in a daisy-chain fashion from one algorithm component processor to the next. Its function is to provide a processor addressed capability for operations indicated to the larger system of processors. Its effect is to serialise the execution of processor addressed operations such as data input and output. This is accomplished in the following manner. Each algorithm component processor uses the logical OR of the global control line contained in the system bus and the count-enable line to determine if it should gate in the instruction currently presented on the system bus. A signal on the global control line indicates a global operation, and means that all processors of the system should perform the specified operation. Global operations are used to initiate the highly parallel computations of the algorithm component. A signal on the count-enable line for an algorithm component processor indicates a processor addressed operation, and means that the instruction and any following data on the system bus are addressed to that specific processor. Once an algorithm component processor has gated in a processor addressed instruction and its data, it then sets the count-enable out line high. The setting of the count-enable out line to high indicates to the next processor in the chain that it should gate in the instruction and data next on the system bus.

It should be noted at this point that other processor interconnection schemes such as multiple buses for parallel data output have not been considered in this study. The reason for this limitation is that the currently available display devices to which the output is directed, only have a single, 8 to 32 bit wide pathway for display list modification. The design of a display device with multiple, parallel pathways for display list modification is outside the scope of this study.

In order to complete our description of the communication mechanism for the system of multiple algorithm component processors, we need to estimate the widths of (1) the system bus data and control lines, (2) the count-enable lines, (3) the external instruction register, and (4) the

external data register. The system bus and count-enable lines sizes are the most important because they extend across VLSI chip boundaries, and hence require package pins. The count-enable lines require two bits, one into and one out of each algorithm component processor. This requires two pins on the VLSI chip. The system bus specification is more difficult in that we have both data and control line widths to specify. The width of the data portion of the system bus is chosen to be 32 bits. This figure is based upon the number of pins we expect to be able to spare on the VLSI chip, and upon the fact that we assume a 32-bit processor, and 32-bit transfers in our register transfer models. The control portion of the system bus has the following components: (1) global/processor addressed bit (1 bit), (2) instruction bits (3 bits), (3) data transfer control lines (6 bits), and (4) miscellaneous control lines (6 bits). The sizes indicated for the data transfer and miscellaneous control lines are taken from the bus designs for similarly sized processors and are not exact [Hayes,1978]. The values quoted only serve as an estimate on the number of control signals expected. Consequently, the total estimate for the control portion of the system bus is 16 bits for a bus total of 48 bits. Adding the two pins for the count-enable lines, this means a minimum of 50 pins on the VLSI chip. This is somewhat under the current package limit of 64 pins, and allows room for additional pin requirements.

The final size specifications we need to make with respect to the system's communication mechanism are those of the external instruction register, and the external data register. The external instruction register needs only three bits, based upon the fact that there are only four operations we expect to signal the algorithm component processor. The purpose of the external instruction register is to hold a signaled instruction until the control portion of the algorithm component processor is finished with its previous operation and ready to execute a new one. The external data register is used to transfer data to/from the algorithm component processor from/to the data portion of the system bus. It is 32 bits wide to match the data width of the system bus.

4.1. Modeling the Larger System of Algorithm Component Processors

The purpose of the model for the larger system of algorithm component processors is to answer the question of exactly how many algorithm component computations can be executed in

parallel in one-thirtieth of a second, with the only limitation being that the coordinates and drawing instructions must be delivered within that same time period. For this model, we assume an infinite capability for processors. We also assume that to obtain the highest processor utilisation, the individual processor may be responsible for multiple, serial algorithm component computations. The timing values for this step are obtained by extending the register transfer model developed for the algorithm component processor.

In order to determine the number of maximal algorithm component computations we can execute in parallel, we compose a model of that system:

$$\text{Real-Time Available} = \text{Input Time} + \text{Computation Time} + \text{Output Time}$$

The model forms a simple linear equation, with the real-time available on one side and the input, output, and computation times on the other. For this model, we make the following assumptions:

(1) the amount of real-time available is 33.333×10^{-3} seconds, (2) all of the algorithm component computations occur in parallel, so only one maximal computation is added to the model's equation (2650 references @250 nsec/reference), (3) the only input is the single 32 bit new contour level, distributed to all processors via a global command (1 reference @250 nsec/reference), (4) the size of the output from each algorithm component computation is of average size (2.54 coordinates and drawing instruction quadruples, or 9 references, for each 2×2 subgrid that generates coordinates [Zyda,1984a]). The model has the following equation:

$$33.333 \text{ msec} = 1 \text{ ref @250 nsec per ref} + 2650 \text{ refs @250 nsec per ref} + X(9 \text{ refs @250 nsec per ref})$$

The variable X stands for the maximum number of algorithm component computations that the modeled system can handle. Solving for X, we find that we can compute in parallel, in one-thirtieth of a second, 14,520 algorithm component computations, generating a total of 36,880 coordinate and drawing instruction quadruples. Again, this requires some 14,520 processors, each operating in parallel.

5. Further Applications Details

Once we have an idea of approximately how many algorithm component computations we can perform in one-thirtieth of a second, we then need to further examine the particular real-time application in order to determine if we are able to handle the expected maximum input data grid. Using the molecular modeling program presented above as the typical application, we find that the largest three-dimensional grid of interest is a cube of 30 units on each side [Barry,1979]. As discussed, a contour surface display is created for a three-dimensional grid by generating the coordinates and drawing instructions for all possible orthogonal two-dimensional grids of that larger grid. For the 30 x 30 x 30 grid, this is 90 30 x 30 grids. Specifying this in total 2 x 2 subgrids, this is 75,690 2 x 2s that must be computed in one-thirtieth of a second. From our architectural discussion, we found that we have the capability for generating coordinates from 14,520 2 x 2 subgrids in one-thirtieth of a second. Given that this is considerably under the total number of 2 x 2s, there are several questions for which we must provide answers. One question is what is the maximum number of 2 x 2s (of the 75,690 total) for which we expect to generate coordinates and drawing instructions for the application of interest? To answer this, we refer to studies of the application and see that the maximum observed percentage of 2 x 2s that generate coordinates is 13 percent, or around 9900 2 x 2s [Zyda,1984a]. The number of coordinates generated for that system, the maximum number for our application's purposes, is 25,150 coordinate and drawing instruction quadruples. Clearly this is within the capabilities shown for the system of algorithm component processors.

Once we know that not all 2 x 2 subgrids of the 75,690 generate coordinates, a second question we need to answer is how do we handle 2 x 2s that do not generate coordinates? One possibility is to double up the algorithm component processors with 2 x 2s of non-overlapping grid point value range. Non-overlapping 2 x 2s never generate coordinates for the same contour level. If we keep track of the ranges for each 2 x 2, and the processor range in each algorithm component processor, we have a method for examining and computing coordinates for all 75,690 2 x 2s in roughly the same amount of time it takes to perform the calculation on only those 2 x 2s

that generate coordinates. The only question is can we find enough non-overlapping 2×2 s in the typical problem to allow this solution? The answer is certainly we can. From studies of the value ranges of the grids we expect to encounter, we find that for a system of 75,690 2×2 s the maximum number of non-overlapping partitions is about 16,000. This is an average of five 2×2 subgrids per partition, with an observed maximum of fifteen 2×2 subgrids in a single partition. Extrapolating these figures to the architecture, we find a requirement of 16,000 algorithm component processors, with a storage capacity of 15 2×2 subgrids in each processor.

A system comprised of 16,000 processors as the above is bound to have a great number of idle processors. One possibility to reduce this expense is to consider a system of multiple algorithm component computations being performed in serial by a single processor. The model for such a system is easily composed from the data computed and derived for the highly parallel system. We will skip the preliminary considerations and model the system with the following assumptions. The input subgrids are already loaded into each algorithm component processor. The output from the total system of algorithm component processors is of average size, i.e. 2.54 coordinate and drawing instruction quadruples are generated from 9900 2×2 subgrids, for a total of 25,146 quadruples, or 89,100 memory references. The output is 32 bits wide, again due to the design of the display processor. In one-thirtieth of a second, there are 133,333 memory references using the figure of 250 nsec per memory reference. Subtracting the total number of memory references required for the output from the total number of memory references in one-thirtieth of a second, we find that 44,233 memory references are available for the computation of multiple subgrids in a single algorithm component processor. Dividing the total available computation time by the maximum amount of time an algorithm component processor could spend on a single algorithm component computation, 2650 memory references, we find that each algorithm component processor can compute the display for 16 subgrids in serial, with the system still being able to deliver the output in real-time. Dividing the total number of subgrids considered for our applications, 75,690 subgrids, by 16, we find that we need 4731 algorithm component processors.

Referring back to the discussion of our ability to coalesce the 75,690 subgrids into 16,000

partitions, each partition containing a maximum of 15 subgrids of non-overlapping grid values, we find that we really only have a requirement for 16,000 subgrid computations. If we design each algorithm component processor to hold 16 of these partitions, i.e. each processor has the capability for 15 times 16 subgrids, then we really only need 1000 processors. The only differences from the algorithm component processor previously described are (1) a larger RAM for the extra subgrid definitions (2048 x 32 bits), (2) a larger microcode ROM for the value range acceptance mechanism (1024 x 32 bits), and (3) a wider instruction portion of the system bus (by 1 bit).

6. VLSI Feasibility for the Contour Surface Display Generator

The above discussion has left us with an outline of the architecture necessary for real-time contour surface display generation. An important factor to consider at this point is the actual feasibility of implementing such a system in the VLSI technology. For this feasibility determination, we need to compute a value for the hardware complexity. The chief components of this complexity are the total number of transistors required, and the total number of VLSI chips. Once these values are obtained, we can then make a statement as to the feasibility of actually constructing the real-time contour surface display generator.

From the architectural specification, we can compute a value for the circuit complexity if we make some fairly simple assumptions. The first assumption is that if we obtain a circuit complexity for the algorithm component processor, then all we have to do to get the total system complexity is multiply by the total number of processors required. The second assumption is that the complexity of the algorithm component processor is less than or equal to the complexity of a known microprocessor, say perhaps the MC68000 used in our evaluation of the algorithm component's real-time capability. One paper, [Frank,1981], provides a comparison of the Motorola MC68000 and the Zilog Z8000 with figures for the total number of transistors. For the MC68000, the total transistor count is approximately 68,000, with 50,000 of those transistors being in the microcode ROMs and PLAs and the remaining 18,000 being in the registers and random logic. For the Z8000, the total transistor count is specified as 17,500. Consequently, a good estimate for the circuit complexity of a processor such as the one we propose for the algorithm

(1) RAM space -- (2 devices/bit)		
2048 x 32 bits	=	131,072 devices
(2) ROM space -- (1 device/bit)	,	
-- tree tables		
2048 x 16 bits	=	32,768 devices
-- microcode		
1024 x 32 bits	=	32,768 devices
(3) Rest of processor space --		
-- ALU		
-- register block		
-- control section		
-- external registers		
-- data and control buses		
	=	18,000 devices
Device total	=	214,608 devices (215K devices)

Figure 7
Algorithm Component Processor's Circuit Complexity Estimate

component processor is 18,000 devices, not counting the RAM space, or the ROM space.

Figure 7 is a short table showing the breakdown of the algorithm component processor into pieces of similar circuit complexity. Using figures of two devices per bit for the random access memory (DRAM), and one device per bit for the read-only memory (ROM), we find that 195K devices are required for the storage alone. Adding that value onto the 18K devices that form the rest of the algorithm component processor, we note that the total number of devices the processor requires is on the order 215K. From the literature, we note that two million device VLSI chips are already being produced in the research lab [Micronews,1984], with ten million device VLSI chips promised in the time period ranging from the year 1985 to the year 2001 [Uhr,1984]. This means 9 algorithm component processors per chip at the two million devices per chip level, and 48 algorithm component processors per chip at the ten million devices per chip level. For the 1000 processors needed for the contour surface display generator, this means a total system size in the range of 112 to 21 VLSI chips.

7. Conclusions

This study has focused on the architectural specification and feasibility determination of the real-time contour surface display generator. The conclusions we draw are that yes, we can put together such a multiprocessor. Once we have made such an assessment, we then need to consider the next steps in this research effort. Two directions come to mind, the second following directly from the first. The first direction concerns the details of how the real-time contour surface display generator is interfaced to a display system. The importance of this research direction becomes evident if we compute a value for the output data rate of the contour surface display generator. In Figure 6, the output is shown to be destined for a display device, with that output passing through a display controller. The assumption for that data transfer has been that it is accomplished via a DMA transfer mechanism of 32 bits width similar in operation to that of the DEC Unibus. Assuming that the output display is of average size, 89,100 32-bit memory references, this is a data rate of 10.7 megabytes per second. The delivery of data to the display system at the rate of 10.7 megabytes per second is somewhat faster than current display system

technology allows. Compounded with this problem, is the fact that besides being able to deliver the picture within the given time constraints, we also need to maintain the functionality of the display system. This means that if we add the contour surface display generator to a display system that we cannot reduce or eliminate the display system's capability for real-time display rotation, scaling, translation, clipping, and other assorted, real-time operations. The full specification of the architectural changes required for the display system by the contour surface display generator are left as an area for further study.

Once we have answered the questions with respect to the contour surface display generator's impact on the design of the display system, the second research direction is to examine other graphics algorithms for implementation in VLSI. If we then perform the same study of the interfacing of those special purpose display generators with the display system, we can see if there are any general principles we can establish. It is not until this question is answered in the general case, that we can actually begin the systematic implementation in VLSI of special purpose, real-time display generators.

8. References

1. Aho, Alfred V., Hopcroft, John E., and Ullman, Jeffrey D. *The Design and Analysis of Computer Algorithms*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1974, Chapters 1-5.
2. Barry, C.D. and Sucher, J. H. "Interactive Real-Time Contouring of Density Maps," American Crystallographic Association Winter Meeting, Honolulu, March 1979, Poster Session
3. Faber, D.H., Rutten-Keulemans, E.W.M., and Altona, C. "Computer Plotting of Contour Maps: An Improved Method," *Computers & Chemistry*, Vol. 3, pp. 51-55, Great Britain: Pergamon Press Ltd., 1979.
4. Frank, Edward H. and Sproull, Robert F. "Testing and Debugging Custom Integrated Circuits," *Computing Surveys*, Vol. 13, No. 4 (December 1981), p. 425.
5. Fuller, S.H. et. al., "A Collection of Papers on CM*: A Multi-Microprocessor Computer System," Technical Report, Pittsburg: Carnegie-Mellon University, Department of Computer Science, February 1977.
6. Hayes, J.P. *Computer Architecture and Organization*, New York: McGraw-Hill Book Company, 1978, p. 408-418.
7. Micronews. "IBM Experimental Million Bit Memory Chip," *IEEE Micro*, Vol. 4, No. 4 (August 1984), p. 119.
8. Newman, William H., and Sproull, Robert F. *Principles of Interactive Graphics*. Second Edition. New York: McGraw-Hill, 1979.
9. Uhr, Leonard. *Algorithm-Structured Computer Arrays and Networks*, Orlando, Florida: Academic Press, 1984.
10. Wright, Thomas and Humbrecht, John "ISOSRF -- An Algorithm for Plotting Iso-Valued Surfaces of a Function of Three Variables," *Computer Graphics: A Quarterly Report of SIGGRAPH-ACM*, Vol. 13, No. 2 (August 1979), pp. 182-189.

11. Zyda, Michael J. *Algorithm Directed Architectures for Real-Time Surface Display Generation*, D.Sc. Dissertation, Dept. of Computer Science, Washington Univ, St. Louis, Missouri, January 1984a.
12. Zyda, Michael J. "A Decomposable Algorithm for Contour Surface Display Generation," Technical Report NPS52-84-011, Monterey, California: Department of Computer Science, Naval Postgraduate School, August 1984b.
13. Zyda, Michael J. "Real-Time Contour Surface Display Generation," Technical Report NPS52-84-013, Monterey, California: Department of Computer Science, Naval Postgraduate School, September 1984c.
14. Zyda, Michael J. "A Contour Display Generation Algorithm for VLSI Implementation," *Selected Reprints on VLSI Technologies and Computer Graphics*, Compiled by Henry Fuchs, p. 459, Silver Spring, Maryland: IEEE Computer Society Press, 1983.
15. Zyda, Michael J. "A Contour Display Generation Algorithm for VLSI Implementation," *Computer Graphics: A Quarterly Report of SIGGRAPH-ACM*, Vol. 16, No. 3 (July 1982), p. 135.
16. Zyda, Michael J. "Multiprocessor Considerations in the Design of a Real-Time Contour Display Generator," Technical Memorandum 42, St. Louis: Department of Computer Science, Washington University, December 1981.
17. Zyda, Michael J. "Joystick Driven Display Rotation and Control Console Management," Technical Memorandum 24, St. Louis: Department of Computer Science, Washington University, November 1980.

Distribution List for Papers Written by Michael J. Zyda

- (1) Dr. Henry Fuchs,
208 New West Hall (035A),
University of North Carolina,
Chapel Hill, NC 27514
- (2) Dr. Kent R. Wilson,
University of California, San Diego
B-014,
Dept. of Chemistry,
La Jolla, CA 92093
- (3) Dr. Guy L. Tribble, III
Apple Computer,
20525 Mariani Ave,
Cupertino, CA 95014
- (4) Dr. Victor Lesser,
University of Massachusetts, Amherst
Dept. of Computer and Information Science,
Amherst, MA 01003
- (5) Dr. Gunther Schrack,
Dept. of Electrical Engineering,
University of British Columbia,
Vancouver, B.C., Canada V6T 1W5
- (6) Dr. R. Daniel Bergeron,
Dept. of Computer Science,
University of New Hampshire,
Durham, NH 03824
- (7) Dr. Ed Wegman,
Division Head,
Mathematical Sciences Division,
Office of Naval Research,
800 N. Quincy Street,
Arlington, VA 22217
- (8) Dr. Gregory B. Smith,
ATT Information Systems,
190 River Road,
Summit, NJ 07901

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943	3
Office of Research Administration Code 012A Naval Postgraduate School Monterey, CA 93943	1
Chairman, Code 52ML Department of Computer Science Naval Postgraduate School Monterey, CA 93943	40
Assistant Professor Michael J. Zyda, Code 52Zy Department of Computer Science Naval Postgraduate School Monterey, CA 93943	40
Dr. Henry Fuchs, 208 New West Hall (035A), University of North Carolina, Chapel Hill, NC 27514	1
Dr. Kent R. Wilson University of California, San Diego B-014 Dept. of Chemistry La Jolla, CA 92093	1
Dr. Guy L. Tribble, III Apple Computer 20525 Mariani Ave. Cupertino, CA 95014	1

Dr. Victor Lesser
University of Massachusetts, Amherst
Dept. of Computer and Information Science
Amherst, MA 01003

1

Dr. Gunther Schrack
Dept. of Electrical Engineering
University of British Columbia
Vancouver, B.C., Canada V6T 1W5

1

Dr. R. Daniel Bergeron
Dept. of Computer Science
University of New Hampshire
Durham, NH 03824

1

Dr. Ed Wegman
Division Head
Mathematical Sciences Division
Office of Naval Research
800 N. Quincy Street
Arlington, VA 22217

1

Dr. Gregory B. Smith
ATT Information Systems
190 River Road
Summit, NJ 07901

1

END

FILMED

2-85

DTIC