This is to certify that the

thesis entitled

A VLSI CHIP ARCHITECTURE FOR THE COMPUTATION OF

THE DIRECT KINEMATIC SOLUTION OF A ROBOTIC MANIPULATOR

presented by

STEVEN SIUTIT LEUNG

has been accepted towards fulfillment
of the requirements for

Masters degree in Elect. Engr.

Michael Shanblatt

Major professor

Michael Shanblatt

Date 4/12/85

A VLSI CHIP ARCHITECTURE FOR THE COMPUTATION OF THE DIRECT KINEMATIC SOLUTION OF A ROBOTIC MANIPULATOR

By

Steven Siutit Leung

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Electrical Engineering and System Science

1985

ABSTRACT

A VLSI CHIP ARCHITECTURE FOR THE COMPUTATION OF THE DIRECT KINEMATIC SOLUTION OF A ROBOTIC MANIPULATOR

By

Steven Siutit Leung

This thesis describes the architecture design and simulation of a VLSI chip dedicated to the computation of the Direct Kinematic Solution (DKS) of a robotic manipulator. The design features fixed-point calculation and on-chip generation of trigonometric functions. The calculation achieves the resolution required for industrial applications by augmenting the internal word size. Regularity considerations lead to the modification of the homogeneous transformation algorithm. Pipelining techniques applied to the inter- and intra-functional unit designs speed up the DKS calculation. Control signals assume a control store implementation, easing both the design and verification processes without sacrificing the completeness of the system-phase design tasks. A symbolic simulation approach verifies the correctness of the design. Transistor-count and cell-count confirm the feasibility of a semi-custom implementation approach with current technology. Results indicate that the DKS can be obtained in microseconds, which amounts to a speed improvement of three orders of magnitude compared to a similar algorithm implemented on a 16-bit microprocessor.

To

my parents and my remote lovers

ACKNOWLEDGEMENT

The author wishes to express his sincere appreciation to his major advisor, Dr. Michael A. Shanblatt, for his guidance, support and encouragement in the course of this research.

He also wishes to thank the committee members, Prof. P. D. Fisher, Prof. E. Goodman and Dr. R. L. Tummala, for their valuable comments in this work.

TABLE OF CONTENT

Page

LIST OF FIGURES

LIST OF TABLES

Chapter I

INTRODUCTION

In a computer-based robotics system with multiple degrees-of-freedom (DOF) movement, the position and orientation of the manipulator are servo-controlled through each individual joint. The parameters of position are conveniently expressed as joint angle variables in a link coordinate system. The position of the object to be manipulated, however, is usually expressed in the reference base (world) coordinate system. Given the joint angle vector θ, in the link coordinate system, the position and orientation of the manipulator in world coordinates can be obtained via the Direct Kinematic Solution (DKS).

A homogeneous transformation method has been developed to solve the direct kinematic problem effectively [1]. However, the successive matrix multiplications involved in the homogeneous transformation require a cumbersome amount of calculations. For similar computationally-bound problems, VLSI (Very Large Scale Integration) technology has shown great potential in improving system performance by implementing concurrent numerical algorithms directly in hardware. Furthermore, advances in CAD (Computer-Aided, or more recently, -Automated, Design) and CAE (Computer-Aided Engineering) systems for VLSI enable such an implementation to be carried out in a custom design approach with relatively low cost.

The purpose of this research project is to investigate aspects of a custom-designed VLSI chip architecture, tailored for the computation of DKS.

Specifically, application requirements for the DKS chip architecture are investigated in order to formulate the design criteria and a set of specifications describing the computational requirements. Based on the PUMA robotic manipulator, a set of simulation program modules is developed to study the effect of fixed-point calculations on the resolution of the resultant position and orientation.

Various methods for SIN/COS function generation are compared and the homogeneous transformation matrices are modified to reduce the control complexities. Based on the modified algorithm and the SIN/COS generation method, a set of functional units and storage elements are determined.

Additional considerations, such as the choice of the most suitable number representation, timing rules, and hardware constraints are discussed. The pipeline concept is extensively applied to the inter- and intra-functional unit designs to improve the speed of the DKS calculation. A detailed timing diagram is provided to illustrate the data flow in the machine and to facilitate the generation of control signals for the DKS calculation cycle. The flow of control is presented in the form of a program written in RTL (Register Transfer Language).

Control signals are identified and partitioned into fields to facilitate the specification and generation. Viewed as machine codes, they are generated through the manual compilation of the RTL program.

Verification of the architecture and control design is carried out by symbolic simulation. By comparing the simulation results with the expected DKS function, errors are discovered and corrected.

Finally, empirical and statistical data is collected from various sources to aid in the estimation of the total chip area requirement. The total DKS calculation time is evaluated from both simplified circuit models and published experimental data.

1.1 Problem Statement

Current research is being conducted in the field of robotic controls to improve the system performance by incorporating various dedicated computing hardware [2]. In the same spirit, the purpose of this research is to explore some specific algorithmic and architectural design alternatives for a single VLSI chip for dedicated high speed DKS computation.

1.2 Approach

The VLSI design process consists of a system design phase and a physical design phase. This research effort focuses exclusively on the first phase development. An overview of the research scheme is presented in Figure 1.1. Tasks in each step are identified and listed alongside while the output from each step is shown in *Italic*.

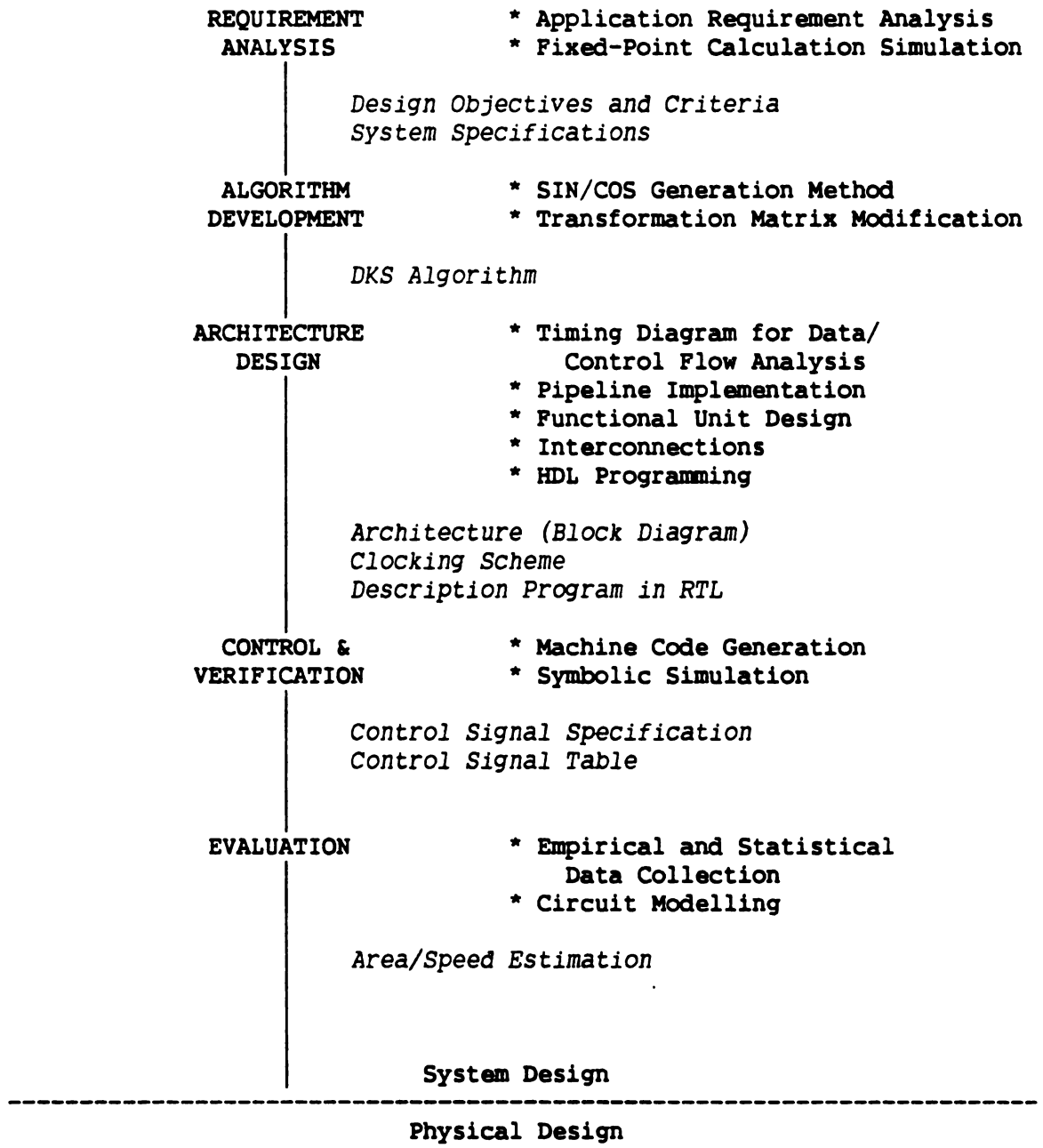


Figure 1.1. An overview of the design methodology with respect to the research scheme.

The high cost of the VLSI development places unprecedented importance upon the precise specification of the integrated system to be designed. The first step, therefore, is to study the requirements as carefully and comprehensively as possible. Questions to be addressed include the future utility of the proposed DKS chip, the I/O format, the interface requirements, the flexibility of adapting the chip to other robotic manipulators, the impact of the working environment on the choice of technologies, and the possibility of employing fixed-point calculations only.

Once the design goals and criteria are resolved, the design proceeds with algorithmic development. Alternatives to the SIN/COS generation methods are discussed. The transformation matrices are modified and decomposed to yield a higher degree of regularity and better hardware utilization. The final version of the DKS algorithm is expressed in a Pascal-like language.

The implementation of algorithms with VLSI technology gives the designer maximum freedom in making software/hardware choices. However, such freedom must be exerted judiciously with an understanding of the potentials as well as various constraints and limitations. These understandings are conveniently formulated as design rules or principles, such as those suggested by Mead and Conway [3], which are presented in Chapter 5. A two's complement number representation is chosen and this presents certain difficulties which must be resolved by the modifications of relevant functional units. Timing requirements are considered in sufficient detail leading to a decision to implement a

two-stage pipeline multiplier.

Ideally, the architecture should achieve performance of such a high degree that it will only be limited by the inherent data dependency. Accordingly, a detailed timing diagram is constructed to aid in the analysis of the data flow. The order of the original DKS calculation is manipulated in such a way that a minimum number of clock cycles is achieved for a given hardware-and-interconnect configuration. The sequence of events is translated directly from the timing diagram into an RTL program.

The control structure is further specified by defining the needed control signals, which are subsquently partitioned into eleven control fields. A table of machine code is compiled from the RTL program manually. The control signal table is then used to drive the data through a simulator (a symbolic simulation program) to verify the design. Simulation principles and program developments are described in Chapter 6.

In order to have a realistic picture of the feasibility of implementing the DKS on a single chip, transistor count and gate count are evaluated first and the total chip area is estimated from data collected on comparable transistor/gate count vs chip area relationships from various sources. Once the area, and hence the chip edge, are known, interconnection delay as well as delay in the combinational logic sections are estimated. Based on the estimated delays, the minimum clock period is evaluated from the worst case delay path.

Finally, to conclude this work, insights to the VLSI design process gained from the design of the DKS chip are sublimated to a higher level of understanding. Results of the evaluation are compared with the design objectives. Implications as well as problems for further research are identified.

Chapter II

BACKGROUND

2.1 The Direct Kinematic Solution of a Robotic Manipulator

The robot arm consists of a number of link-joint pairs, each providing one degree-of-freedom. It is convenient, for the purpose of control, to express the position of each individual rotational (or prismatic) link-joint pair by a single variable Θ_i (or d_i) with respect to its own link coordinate system. A unique 4X4 homogeneous transformation matrix A_i, which is a function of θ_i (or d_i), maps a vector in the link i^{th} coordinate system to the link $i\text{-}1^{th}$ coordinate system. Thus, for a six degree-of-freedom robot arm, given its six rotational joint variables $\Theta = (\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6)$, the joint space to Cartesian space mapping is obtained by the successive multiplication of the six homogeneous transformation matrices,

$$T = A_1 \cdot A_2 \cdot A_3 \cdot A_4 \cdot A_5 \cdot A_6 = [\ \mathbf{n}\ \mathbf{s}\ \mathbf{a}\ \mathbf{p}\ ]. \qquad (2.1)$$

The resultant homogeneous matrix T gives the orientation vectors of the wrist **n**, **s** and **a**, and the current arm position **p** which is defined as the vector from the origin of the base to the wrist, all in the world coordinate system. This joint space to Cartesian space mapping is known as the Direct Kinematic Solution (DKS) [4,5].

In the DKS calculation (eq. 2.1), each A_i has the form

$$A_i = \begin{bmatrix} \cos\theta_i & -\cos\alpha_i\sin\theta_i & \sin\alpha_i\sin\theta_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\alpha_i\cos\theta_i & -\sin\alpha_i\cos\theta_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

where α_i, a_i, and d_i are the parameters of the i^{th} link-joint pair. Even though the matrix A_i appears complicated, the parameter α_i of almost all of the existing robots only takes on values that result in $\sin\alpha_i$ or $\cos\alpha_i$ being either 0 or ± 1. As an example, the simplified PUMA robot arm transformation matrices together with their parameter values are shown in Figure 2.1 and Figure 2.2. The highly regular pattern and small number of elements for each A_i suggest the possible use of a dedicated computing structure to enhance computational throughput in the calculation of DKS. This can be obtained by incorporating pipeline and concurrent processing concepts with high speed VLSI hardware.

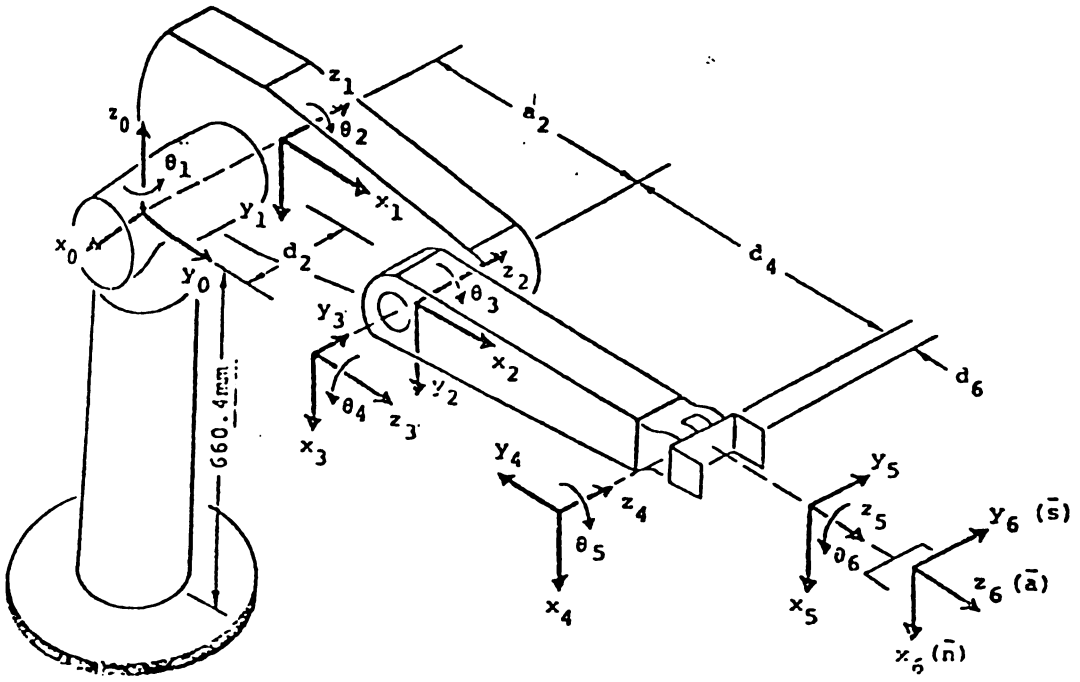
$$A_{i-1} = \begin{bmatrix} C\vartheta_i & -C\alpha_i S\vartheta_i & S\alpha_i S\vartheta_i & a_i C\vartheta_i \\ S\vartheta_i & C\alpha_i C\vartheta_i & -S\alpha_i C\vartheta_i & a_i S\vartheta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_0^1 = \begin{bmatrix} C_1 & 0 & -S_1 & 0 \\ S_1 & 0 & C_1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_1^2 = \begin{bmatrix} C_2 & -S_2 & 0 & a_2C_2 \\ S_2 & C_2 & 0 & a_2S_2 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_2^3 = \begin{bmatrix} C_3 & 0 & S_3 & a_3C_3 \\ S_3 & 0 & -C_3 & a_3S_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A_3^4 = \begin{bmatrix} C_4 & 0 & -S_4 & 0 \\ S_4 & 0 & C_4 & 0 \\ 0 & -1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_4^5 = \begin{bmatrix} C_5 & 0 & S_5 & 0 \\ S_5 & 0 & -C_5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad A_5^6 = \begin{bmatrix} C_6 & -S_6 & 0 & 0 \\ S_6 & C_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $C_i = \cos\vartheta_i$; $S_i = \sin\vartheta_i$

Figure 2.1. The transformation matrices of the PUMA arm [5].



PUMA Robot Arm Link Coordinate Parameters					
Joint i	ϑ_i	α_i	a_i	d_i	Range
1	90	-90	0	0	-160 to +160
2	0	0	431.8 mm	149.09 mm	-225 to 45
3	90	90	-20.32 mm	0	-45 to 225
4	0	-90	0	433.07 mm	-110 to 170
5	0	90	0	0	-100 to 100
6	0	0	0	56.25 mm	-266 to 266

Figure 2.2. The configuration of the PUMA robotic manipulator and its parameter values [5].

2.2 VLSI System Design

Design, in its most general sense, can be described as a process of successive mappings or transformations of specifications from one domain (or abstraction level) into another. In this aspect, VLSI design is no different from design in other disciplines. However, the fabrication of more than 100,000 transistors on one chip requires the use of special tools and techniques to make the design and verification process feasible.

2.2.1 Characteristics and Important Trends of VLSI System Design

In traditional engineering practice, design and manufacturing can be done almost independently of each other. This situation, however, has dramatically changed for VLSI design. "Integrated", in fact, means not only the integration of a large number of circuit components, but also implies the designer's integrated knowledge, from the choice of design alternatives to circuit layout to packaging [6]. Fortunately, the formulation of design rules and the development of powerful tools relieve the designer's burden by providing a well defined interface between the system designers and engineers working on physical implementation. Nevertheless, the complexities and interactions between and among all design levels have increased to the extent that it is

beyond human capabilities. As a result, design automation is no longer a luxury but a necessity.

Today, the process of physical design, including layout, wiring and timing analysis, has been largely automated [7]. Some state-of-the-art design automation systems are capable of placing and wiring over 37K logic gates on a chip, achieve 99% automatic physical design and 100% automated checking, and assure error-free functioning [8]. While improvement will certainly continue in this area, automated logic synthesis is gaining more and more attention. CAE systems aimed at automated translation of register transfer level description of digital systems to VLSI realization through gate arrays or standard cells, have been developed and claim varying degrees of success [9].

Amidst all the progress in VLSI technology, perhaps the single most important event is the appearance of the 'silicon compilers' which are about to revolutionize the VLSI design process. Silicon compilation attempts to reduce the entire physical design process of a VLSI system into a compilation process such that, at any point of time, most of the ultimate characteristics of the finished chip can be accurately estimated from available data [10,11]. This opens new opportunities for system designers to experiment with various algorithmic/architectural alternatives with much less effort and cost. This facilitates the making of more intelligent design decisions based on alternative analyses.

These developments have brought forth two major impacts on VLSI design. First, more emphasis has been shifted toward higher and higher levels of design. It is now recognized that the size and performance of VLSI chips are more influenced by higher level designs, such as task-realization algorithm design and architecture design, than logic design or layout [6,12]. The significance of this shift may be better understood from a historical perspective by comparing it with the shift from machine language to high-level language in the early days of computer programming [13]. In fact, the influence of the software engineering practice on the VLSI design process is pervasive as evidenced by the use of terms like "silicon compiler", and "top-down" and "bottom-up" approach.

In parallel with this upward shift, custom and semi-custom design approaches emerge as the mainstream of next-generation VLSI design. The custom IC market has grown from $54 million in 1979 to $1 billion in 1984 and is predicted to reach $5 billion by 1989 [11]. This trend can also be witnessed in the widely published objectives of the US Department of Defense's VHSIC program [14] and DARPA's announcement of a fast turnaround fabrication service available for commercial use. With the availability of silicon compilers and the formation of the silicon foundry [11,15], the ideal of a fast design turnaround time with relatively low cost is rapidly becoming a reality. It is under such auspices that the custom design approach is elected for the design of the DKS chip in this work.

2.2.2 The VLSI Design Space Concept

A significant wealth of knowledge about VLSI design has been accumulated in the past ten years as the industry grows. In order to gain a better understanding and insight of the design methodology, the domain of VLSI design may be expressed as a design space spanned by three axes - functional, structural, and geometrical representations - as shown in Figure 2.3. Elements in each axis represent the abstract level or refinement steps involved in the design process. VLSI design methodology, from this perspective, is visualized as a specific path traversing through the design space. A particular path taken by a certain school of thought, not surprisingly, more or less reflects a particular environment (usually expressed in needs and constraints of human, technological and financial resources) under which that particular methodology has evolved. (It is interesting to note that the term "level" suggests structured programming, and "representation," on the other hand, comes from the AI (Artificial Intelligence) vocabulary.)

In a more formal sense, design methodology, as suggested by Baller, can be defined as "a set of codified technique[s], broadly applicable, that facilitate the creation of designs that are functionally correct, qualitatively acceptable, are easily understood, and are easily modified" [16].

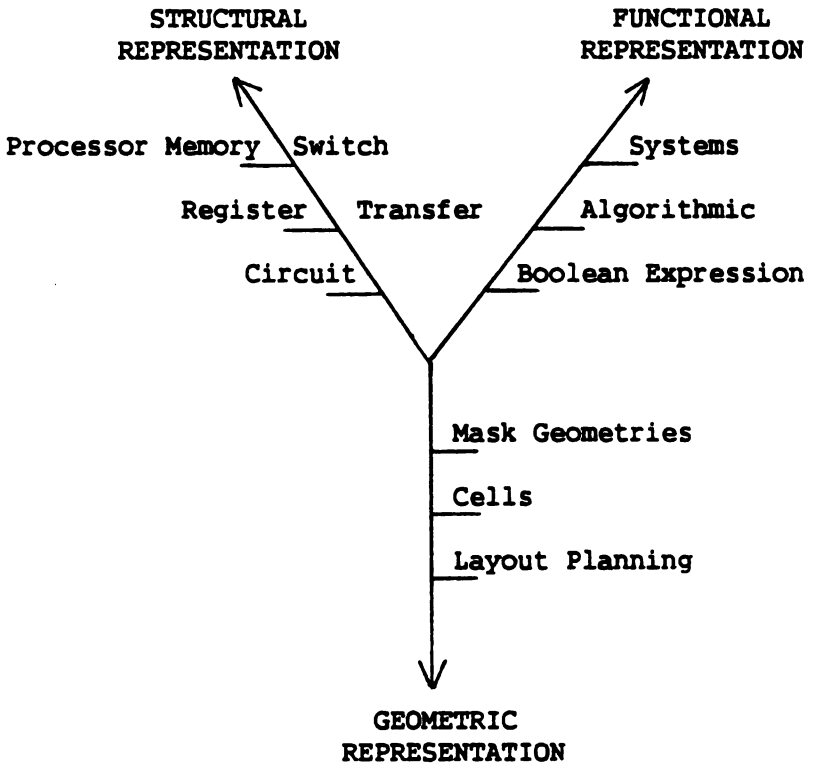


Figure 2.3. Triartite representation of the design and various design levels [17].

VLSI design methodology is still in a developmental stage. Therefore, instead of endorsing a particular approach, only the basic and common aspects of VLSI design methodology are discussed here. The presentation below follows the philosophy of Gajski and Kuhn [17].

Extreme complexity in VLSI design forces the designer to take a structured hierarchical approach. By hiding lower level details, the designer is thus capable of concentrating on the design at a specific abstract level. In spite of their wide differences, design levels in the entire VLSI design space fall into three categories of

representations: the functional, structural, and geometrical representations.

The functional representation is at the highest level of the design and may be captured on several sublevels. The most widely accepted of them are the systems, algorithmic, and Boolean expression (logic) levels. Between the functional and geometrical representations is the structural representation. It maps a functional representation onto a set of components and connections under constraints such as cost, area and timing. Sometimes structural representation may overlap with functional representation. Commonly used levels of structural representation are the processor memory switch, the register transfer (operator register bus), and the circuit level. The geometrical representation ignores, as much as possible, what the design is supposed to do and binds its structure in space (physical design) or to silicon (geometrical design). Geometric representation levels include layout planning with arbitrary size blocks, cells, and physical mask geometries.

This research effort focuses mainly on the design of the DKS chip at levels corresponding to the system, algorithmic, and register transfer levels.

Chapter III

SYSTEM SPECIFICATIONS

A lesson VLSI engineers learn from software engineering is the crucial role of requirement and specification definition in the design cycle. Study has shown that while requirement analysis occupies only 3% of the total cost, its errors are 100 times more expensive to correct than implementation errors. And unfortunately, over 40% of the errors observed during testing are due to incorrect or misinterpreted requirements or functional specification [18]. Therefore, at the beginning of this work, a literature search was undertaken to collect data on the application requirements in order to derive realistic design specifications.

3.1 Application Requirements

Robots can be classified as Cartesian, cylindrical, rotational or articulated depending on the number of prismatic/rotational link-joints. The kinematic problems of the former three classes are simple, but their rigid structures limit their use to only a certain class of applications. The articulated robot arm, on the other hand, mimics the actions of the human arm and is the most flexible. Consequently, the control is more complicated.

The function of the proposed DKS chip is to map the position and orientation of an articulated robotic manipulator from the joint space to the Cartesian space. The angular position of a joint can be measured by a potentiometer. More recently, two major kinds of optical shaft encoders (see Figure 3.1) have been developed. They utilize digital techniques and the measurement is more accurate. An absolute shaft encoder can provide a resolution of 0.04 degrees. The incremental shaft encoder can provide a somewhat lower resolution of 1/2500 turns (0.144 degrees) and requires more interface hardware. It has the advantage of encoding position and velocity information simultaneously and its cost is relatively low [19].

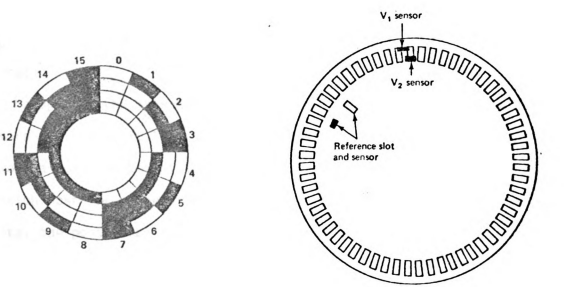


Figure 3.1. Absolute shaft encoder and incremental shaft encoder [19].

Surveys have shown that the positional resolution requirement of various applications range from 0.1-10.0mm for a working range of 1-2m [20]. Expressed in a relative terms, the base two logarithm of the ratio of the robot's maximum reach divided by its resolution is approximately 14. Repeatability requirements of 0.01-1mm are typical. The resolution requirement appears readily achievable using 16-bit processors [21].

Minicomputers are often used as the control computer in which information from sensors is transmitted to the CPU via a 16-bit bus. The immediate use of the DKS chip is to accept joint angle inputs and produce a real-time position and orientation in world coordinates. In the future, the DKS chip may be incorporated into iterative algorithms [22] to obtain the more useful Inverse Kinematic Solution (IKS). In either case, it is desirable to have the input port and output port separated.

To specify the orientation, only two out of three orientation vectors are needed [5,19]. Therefore, during the entire DKS calculation period, a total number of 15 I/O operations are required if the SIN/COS function can be generated on the chip. This includes six joint angle inputs, one positional and two orientational vector outputs. Compared with the minimum of 60 multiplications and 34 additions, the I/O obviously does not constitute a bottleneck. Therefore, a multiplexed scheme through separated I/O ports is assumed for I/O operations and its implementation will not be considered in detail here.

Throughout this project, the PUMA robot arm is chosen as a testbed to study the DKS chip architecture because of its popularity. Even though the general transformation matrix method can be applied to various arm configurations, differences in parameter values and types of variables (prismatic versus rotational) introduce tradeoffs between flexibility and time-area complexity of the control hardware. However, tradeoffs due to the difference of parameter values are simpler to handle and thus flexibility is given a higher priority in such a case.

3.2 Technology Assessment

Three particular state-of-the-art VLSI achievements have implications on the design of the DKS chip.

The first one is a 16-by-16 multiplier-accumulator chip by TRW. Using 1-μm CMOS technology, it can perform a multiply-accumulation operation in 90ns with maximum power consumption under 350 mW [23].

The second achievement is the 32-bit MC68020 microprocessor by Motorola. Using 2-μm HC-MOS technology, it contains roughly 200K transistors on a 375-by-350-mil chip. Operated at 16.67MHz, it dissipates only 1.5 watts [24].

The third one is the S83 "operating system on chip" by Gould AMI. Using 3-μm NMOS technology, it is actually a single chip integration of a Zilog Z80 microprocessor, a 64K ROM (containing its standard operating system), and the system interface logic [25].

Because of the high packing density, low power consumption and high noise immunity, which is particularly important in view of the generally harsh working environment of robotic operations, CMOS is probably the natural choice for the DKS chip. However, it is noted that the system design rules (see Chapter 5) are intentionally formulated to be independent of any particular IC technology. This enables the system designer to take advantage of the best technology available. Therefore, in this design, while CMOS technology is generally assumed, NMOS circuit models are also used because of availability. The conversion problem will be addressed during evaluation.

3.3 Fixed-Point Calculation

Since the joint angle input needs only 12 bits (maximum resolution of 0.04 degrees), and the resolution requirement of position ranges from 0.1-10mm for a working range of 1-2m, the I/O values are readily transferable through a 16-bit bus. It is of great advantage if the internal calculations can be carried out in fixed-point calculations since both the hardware design and control mechanism will be much simpler. Accordingly, Fortran simulation programs have been developed to study the round-off effects and ultimately determine the minimum number of fixed-point bits required by the resolution parameters.

A table look-up and interpolation algorithm is chosen for the SIN/COS generation (see Chapter 4). As a result, the position error may originate from three sources. The error due to the resolution limit of

the joint angles is intrinsic and cannot be eliminated. Consequently, all input angles in the simulations assume a 12-bit value. The second source of error is introduced in the sine function generation and is a function of table size as well as word size. The third error source is due to the accumulated round-off caused by the finite fixed-point internal word size of the arithmetic hardware.

The first simulation program assumes no internal word size limit so as to isolate the effect of table size as the subject of investigation. 1000 joint angle vectors are randomly generated as input. The range for each angle is checked and the 12-bit angle word size is enforced. The DKS results, calculated with various table sizes, are then compared with the correct values. The tabulated results are presented in Table 3.1.

Table 3.1. Error due to the interpolation table size.

	Table Size (Entries)			
Type	32	64	128	256
Orientation (ave)	0.0006	0.0001	0.0000	0.0000
Position (ave)	0.25(mm)	0.058(mm)	0.014(mm)	0.0034(mm)
Orientation (max)	0.0028	0.0006	0.0002	0.0000
Position (max)	1.58(mm)	0.334(mm)	0.084(mm)	0.019(mm)

The first simulation result shows that with table size above 128 entries, the accumulated error due to the sine function interpolation error is negligible. The result is encouraging, and therefore a second Fortran program is used to simulate the fixed-point DKS calculation for variable internal word sizes with table sizes of 128 and 256. The results are plotted in Figure 3.2.

The figure shows two alternatives to achieve the 0.1mm positional resolution. Either an 18-bit word with a table size of 256 or a 20-bit word with a table size of 128 is required. Although the 18-bit word alternative requires double the table size, the resultant increase in total chip area will be smaller than that of the latter which will increase roughly by 10%. Moreover, an increase of internal word size will also increase the delay time. Therefore, the 18-bit word with a table size of 256 is chosen.

3.4 Specifications

3.4.1 Design Objective and Criteria

The objective is to design a basic VLSI chip architecture dedicated to the fixed-point computation of the DKS for a six DOF articulated robotic manipulator, with computational speeds surpassing ordinary computational techniques. A major effort is devoted to the study of architectural alternatives for the computational requirements of DKS.

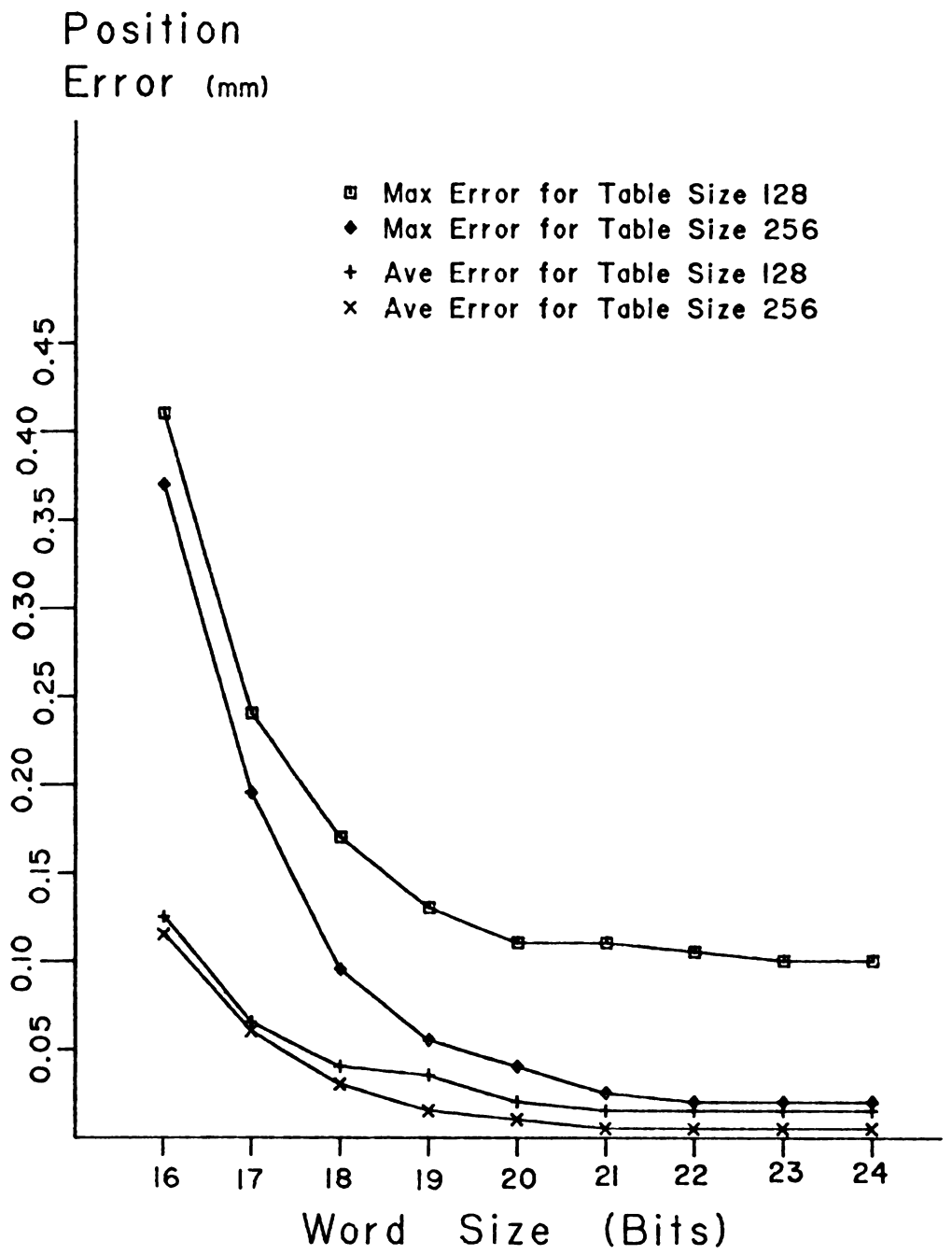


Figure 3.2. The accumulation error versus internal word size.

The following criteria, based on the application requirement analysis, have been established to guide the making of various design decisions:

* MOS technology with a feature size of 2μm or below and a clock of up to 20MHz are to be used.
* SIN/COS functions should be generated on chip to avoid possible I/O bottleneck.
* The architecture should be applicable to the whole class of six DOF articulated manipulators of rotational joints.
* Intra-functional unit pipelining is to be used and limited to two stages to reduce design complexities.
* Minimum cycle time and minimum interconnections are sought with the former having higher priority.

3.4.2.1 I/O and Functional Specifications

The input to the proposed DKS chip consists of six 12-bit joint angle values in fractional-turn representation [26]. Each is subject to a range limit as specified in Figure 2.2. The output consists of two orientation vectors, **n** and **a**, and the position vector **p**, all in two's complement representation.

The I/O formats are shown in Figure 3.3. I/O operations are multiplexed through separated I/O ports. The chip function is holistically defined by the matrix multiplications of eq. 3.1. The DKS is obtained by fixed-point calculation with an internal word size of 18

bits.

$$\begin{bmatrix} n & a & p \\ 0 & 0 & 1 \end{bmatrix} = A_1\ A_2\ A_3\ A_4\ A_5 \begin{bmatrix} C_6 & 0 & 0 \\ S_6 & 0 & 0 \\ 0 & 1 & d_6 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

3.4.2.2 Performance Specification

Since the major concern at this point is whether the DKS function can be realized on a single chip, only two basic performance requirements are specified here:

* Resource utilization higher than 90%;
* Total calculation time less than 10μs.

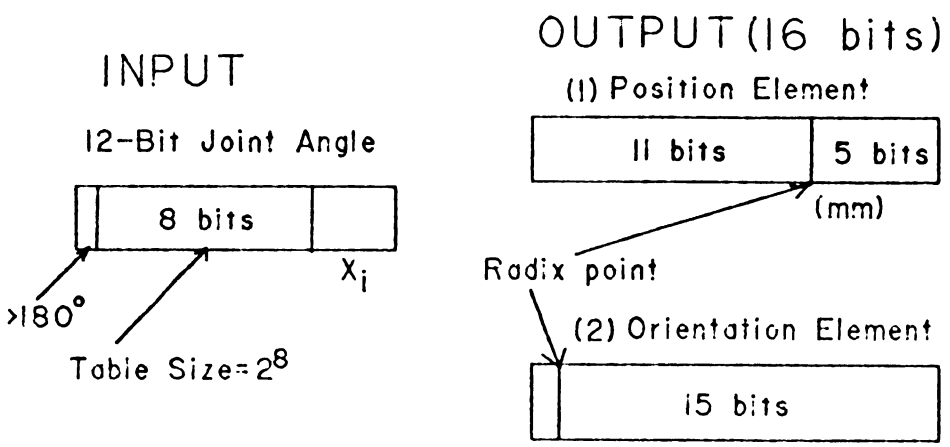


Figure 3.3. The I/O data formats.

In any bus oriented processor system, the performance is ultimately limited by the most precious resource - the bus bandwidth. Because of this limitation and the data dependency, 100% resource utilization is generally impossible. The performance specification figure is intended to provide a measure of how well the algorithm and the architecture match.

The second specification represents an absolute measure of performance of the proposed chip. In order to get a feeling for the DKS calculation speed, a DKS algorithm, similar to the one developed in Chapter 4, has been implemented on the Intel 8086 microprocessor. Experimental results show an average of 15ms total calculation time when run at 10MHz. Thus, the specification goal of the design in this project amounts to a speed improvement of three orders of magnitude.

Chapter IV

ALGORITHM DEVELOPMENT

4.1 Sine Function Generation

Each transformation matrix A_i in eq. 3.1 is a function of the joint angle θ_i for which both sine and cosine values are needed in the computation. Since the cosine can be obtained easily from the sine through trigonometric equivalence, only sine generation methods need be considered.

In general, there are four practical methods to generate the sine function. The oldest method of Taylor series expansion is notoriously slow and thus not considered here. The CORDIC (COordinate Rotation DIgital Computer) algorithm [27] has the advantage of large function capacities [28] but its hardware design is more involved. Moreover, there is no indication that the accuracy it will achieve is needed nor that it can be implemented as a small functional unit on a single DKS chip. Therefore, this implementation method is also not considered.

The remaining two candidates both employ ROM look-up techniques in their calculations. The first method, described by Muroga [29], is based on the principle of trigonometric sum of angles. It recursively divides the argument into a more significant part and a less significant part until the individual parts are small enough to address a ROM of reasonable size. The sine value is then obtained by summing up all the partial results. An example of a schematic block diagram based on this

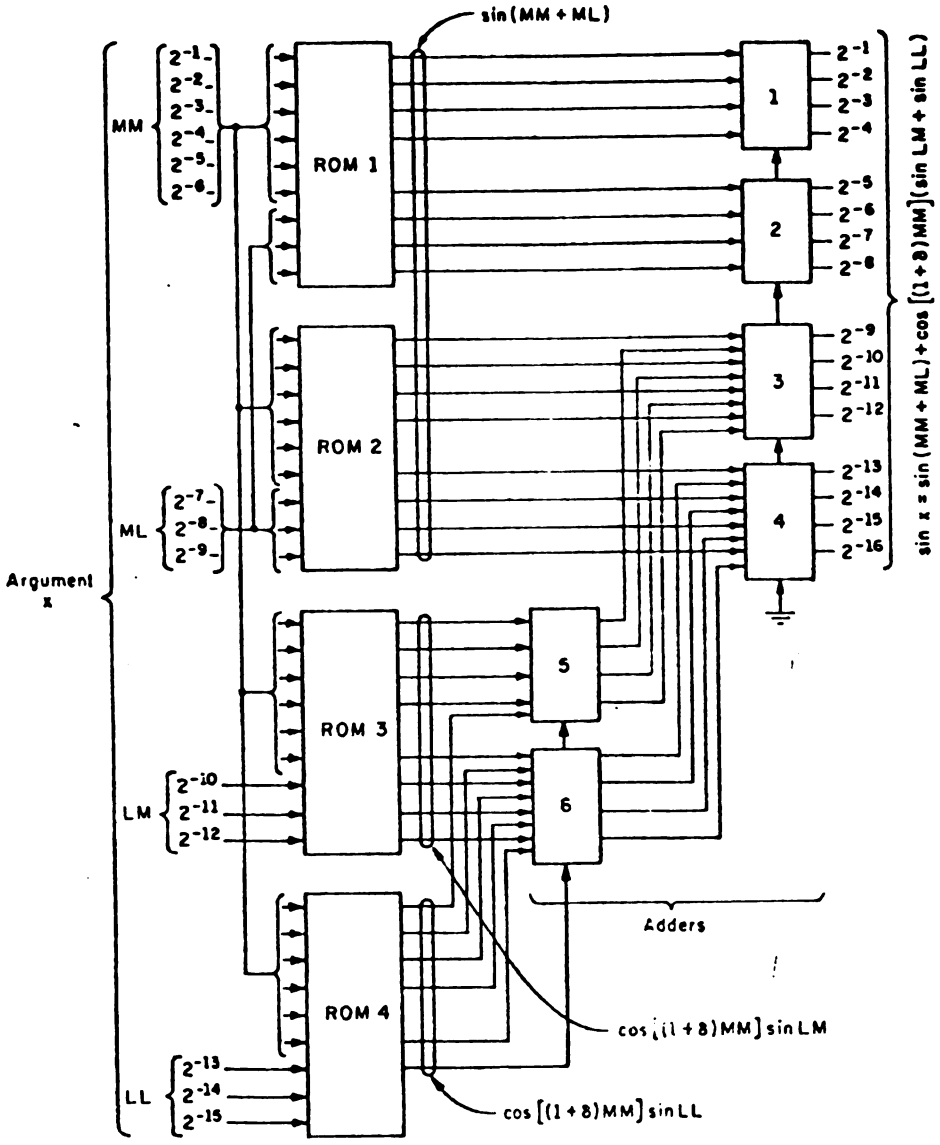


Figure 4.1. The schematic block diagram for sine function generation [29].

concept is shown in Figure 4.1. The design accepts a 16-bit angle input and outputs a 16-bit sine value.

In contrast, the second method proposed by Ruoff [26] is based on linear interpolation as shown in Figure 4.2. In this method, the domain of the argument is divided into intervals of 2^n which become the table size. The endpoints of each interval are calculated and adjusted so that the interpolation error is spread evenly over each interval. The upper n bits (not including the most significant bit) of the argument are used to address a precalculated table. The lower part is then used to interpolate the sine value using the endpoint value and the interpolation difference of that interval.

A comparison of the two described methods is shown in Table 4.1. While the first method requires more ROM area, its only external need is an adder. Since both methods require two-level calculations, the first method appears faster as multiplication tends to be slower than addition. However, an on-chip multiplier is necessary to speedup the DKS calculation. The speed of the second method can approach the first if pipelining is implemented within the multiplier, and between the multiplier and adder. Thus, with much less ROM area, the second method is preferred.

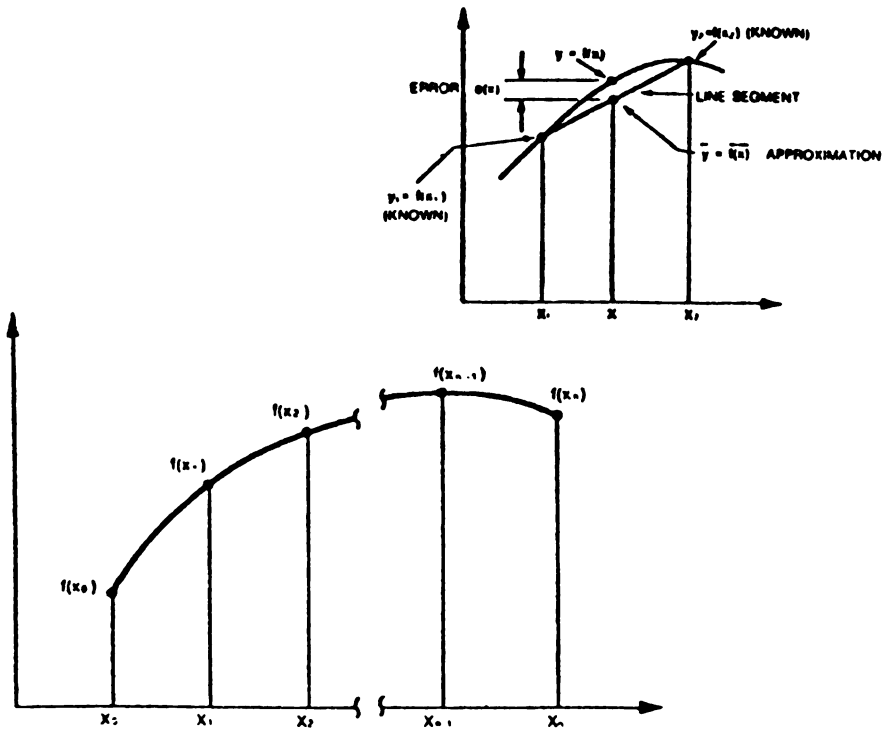


Figure 4.2. Sine function by linear interpolation [26].

Table 4.1. A comparison of two sine generation methods.

Method	ROM (Blocks)	Multiply (Operations)	Add (Operations)	Levels	Accuracy
Muroga	4	0	2	2	2^{-14}
Ruoff	1	1	1	2	2^{-14} *

* Table size = 128 entries

4.2 Modifications on the Homogeneous Transformation Matrices

A concensus among VLSI designers is that a good VLSI architecture should have the following properties [30,31]:

1. It should be implementable by only a few different types of simple cells.
2. It should have simple and regular data and control paths so that the cells can be connected by a network with local and regular interconnections. Long distance or irregular communications must be minimized.
3. It should use extensive pipelining and multiprocessing. In this way, a large number of cells are active at one time so that the overall computational rate is high.

The realization of the above properties is ultimately constrained by the data dependency inherent in the task to be implemented. However, within this constraint, the design of the algorithm determines the eventual performance of the chip. The demand placed on the algorithm can probably be summarized into a single goal - regularity.

4.2.1 Regularity of the Matrix Entries

Each transformation matrix in eq. 3.1 can be partitioned into four submatrices as

$$T = \left[\begin{array}{c|c} R_{3X3} & p_{3X1} \\ \hline f_{1X3} & 1X1 \end{array}\right] = \left[\begin{array}{c|c} \begin{array}{c}\text{Rotation}\\ \text{Matrix}\end{array} & \begin{array}{c}\text{Position}\\ \text{Vector}\end{array} \\ \hline \begin{array}{c}\text{Perspective}\\ \text{Transf.}\end{array} & \text{Scaling} \end{array}\right] . \qquad (4.1)$$

For robotic applications, the perspective transformation and scaling are always 0 and 1 respectively. The rotation matrix is already well structured, with the first column always $[\ C_i\ S_i\ 0\ ]^T$ and the other two columns $[\ 0\ 0\ \pm 1\ ]^T$ and $\pm[-S_i\ C_i\ 0\ ]^T$ arranged in different orders. However, the position vectors in A_2 and A_3 have elements of $a_{2(3)}C_{2(3)}$ and $a_{2(3)}S_{2(3)}$ which cause the entries to be irregular. Note that A_3 can be decomposed as a product of a rotation matrix and a pure translation matrix as

$$A_3 = \begin{bmatrix} C_3 & 0 & S_3 & a_3C_3 \\ S_3 & 0 & -C_3 & a_3S_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} C_3 & 0 & S_3 & 0 \\ S_3 & 0 & -C_3 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (4.2)$$

The translation matrix merely adds the element a_3 to the first element of the position vector of A_4. Likewise, A_2 can be decomposed which results in a_2 being added to the left-component matrix of eq. 4.2. With this modification, all the transformation matrices now have the same basic structure. The redefined transformation matrices A_2 to A_4 are shown in Figure 4.3.

$$A_2 = \begin{bmatrix} C_2 & -S_2 & 0 & 0 \\ S_2 & C_2 & 0 & 0 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_3 = \begin{bmatrix} C_3 & 0 & S_3 & a_2 \\ S_3 & 0 & -C_3 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad A_4 = \begin{bmatrix} C_4 & 0 & -S_4 & a_3 \\ S_4 & 0 & C_4 & 0 \\ 0 & -1 & 0 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 4.3. The three modified transformation matrices.

4.2.2 Decomposition of the 4X4 Homogeneous Transformation Matrix

Even though systolic array processors designed for 4-by-4 matrix multiplications have been proposed [32], they are not yet practical for single chip implementation [33]. Moreover, most of the entries of each matrix in the DKS calculation are either 0 or 1 such that the array processor approach may be overkill. Since only three vectors are actually involved in the calculations, each intermediate matrix multiplication can be decomposed to a 3-by-3 matrix multiplication and a vector addition as

$$\left[\begin{array}{c:c} R & \mathbf{p}' \\ \hdashline \mathbf{0} & 1 \end{array}\right] [\, \mathbf{n}\ \mathbf{a}\ \mathbf{p}\,] = [\, R\cdot\mathbf{n}\ R\cdot\mathbf{a}\ R\cdot\mathbf{p}+\mathbf{p}'\,]. \tag{4.3}$$

In ordinary matrix multiplication, the resultant column can be expanded to

$$A\cdot\mathbf{b} = \begin{array}{l} a_{11}\cdot b_1 + a_{12}\cdot b_2 + a_{13}\cdot b_3 \\ a_{21}\cdot b_1 + a_{22}\cdot b_2 + a_{23}\cdot b_3 \\ a_{31}\cdot b_1 + a_{32}\cdot b_2 + a_{33}\cdot b_3 \end{array} \tag{4.4}$$

Because of the special structure of the rotation matrix, the resultant column is actually either

$$\begin{matrix} a_{11}\cdot b_1 + a_{13}\cdot b_3 \\ a_{21}\cdot b_1 + a_{23}\cdot b_3 \\ \pm b_2 \end{matrix} \quad \text{or} \quad \begin{matrix} a_{11}\cdot b_1 + a_{12}\cdot b_2 \\ a_{11}\cdot b_1 + a_{22}\cdot b_2 \\ \pm b_3 \end{matrix} \; .$$

From the expansion above, it is easy to see that twelve multiplications and six additions are needed for each matrix multiplication. Two additional multiplications for SIN/COS must be carried out first. This implies that a functional unit set of more than two multipliers and one adder will result in low hardware utilization. However, the performance of a two-multiplier hardware unit can be approximated by that of a two-stage pipelined multiplier with half of the area and a much simpler interconnection requirement. Therefore, in this design, only the architectural alternatives for a one-multiplier-one-adder configuration are considered.

4.3 The Final Algorithm

Based on the decomposed matrix multiplication, the final DKS algorithm is presented below. The SIN/COS generation and the pseudo-matrix-multiplication described earlier will be given a more detailed treatment in the next chapter and hence are simply presented as two procedures here.

```
Procedure DKS (jangle, var world : register)

const d6=56.6*2**7; a3=-20.5*2**7; d4=432*2**7;
      a2=432*2**7; d2=149.5*2**7;
type register=integer<0:17>;
var jangle : array [1..6] of register;
    world : array [1..3,1..3] of register;
    buffer : array [1..3] of register;
    posvec : array [1..5,1..3] of register;
    creg, sreg : register;

Begin
  posvec[1..5,1..3]:=0;
  posvec[3,1]:=a3; posvec[4,3]:=d4;
  posvec[2,3]:=d2; posvec[3,1]:=a2;
  world[1..3,1..3]:=0; world[2,3]:=1*2**17; world[3,3]:=d6;
  Table-look-up-and-interpolation (jangle,6,creg,sreg);
  world[1,1]:=creg; world[2,1]:=sreg;
  for I:=5 to 1 do
    Begin
      Table-look-up-and-interpolation (jangle,I,creg,sreg);
      for J:=1 to 3 do
        Begin
          buffer[1..3]:=0;
          Pseudo-Matrix-Multiply(I,buffer,creg,sreg,world[J,1..3]);
          if J<>3 then world[J,1..3]:=buffer[1..3]
        End;
      world[3,1..3]:=buffer[1..3]+posvec[I,1..3]
    End
End.
```

Chapter V

ARCHITECTURE DESIGN

The architecture of a computational system can be characterized in many ways depending on the designer's view as well as the design level. A view proposed by Dasgupta [34] is particularly useful for the design of the DKS chip. In this view, an architecture is collectively defined by a specification of the *functional* capacities of its physical *components*, the logical structure of their *interconnections*, the nature of the information (data) *flow*, and the *control* structure and mechanism.

The design decisions on system components, such as functional units or storage elements, are rooted in the needs of how data are to be processed. On the other hand, the interconnection between these building blocks depends mainly on how the data flow. The strategy adopted in this work is to determine the structure of the functional units first. Considerations on timing are explained in detail and several clocking scheme alternatives are discussed. A detailed timing diagram is constructed to analyze the data flow of the DKS algorithm. The data flow is then manipulated in such a way that maximum resource utilization is achieved. The architecture is modified as necessary and presented in block diagram form. Once the architecture of the DKS chip is defined, the DKS function is realized by a logical sequence of events activated by signals from the control logic. This sequence of events is expressed in the form of a program written in RTL (Register Transfer

Language).

5.1 VLSI System Design Rules

The following three rules from Mead and Conway [3] are adequate to guarantee the correct functioning of a bus-oriented chip design and are rigidly followed in the design of the DKS chip.

Rule 1: The system is driven by a two-phase, non-overlapped clock; phase one inputs from phase two outputs and vice-versa.

Rule 2: Functional units are timeless C/L (Combinational Logic); operations on data are separated by latches or registers which are controlled by the two-phase clock.

Rule 3: A standard bus scheme is used; data transfer through the bus occurs during phase one, and the bus is precharged during phase two.

5.2 Architecture Development I

5.2.1 Functional Units

As will be seen, the choice of functional units, the mode of calculation, and the sine function implementation are actually very interdependent on each other. The mode of calculation is considered first.

The elementary arithmetic operations needed for the DKS calculation are multiplication, addition, subtraction and unitary complementation. Several factors appear to favor the mode of sign-magnitude integer calculation. First, the multiplier design is relatively simple as the sign of the result is just the XOR (eXclusive OR) of the signs of the two operands. Furthermore, the most significant bit of the sine argument indicates whether the angle is greater than or equal to 180 degrees and can be used directly as the sign bit of the result. And above all, the unitary complement operation can be carried out separately without occupying the bus and adder with a small increase in hardware. The fatal disadvantage, however, is that the adder design is cumbersome and the speed is slow because of the need for post-adjustment [35].

Another alternative is a sign-complement mode such as the two's complement number representation. Additionally, several two's complement multiplier designs are available in the literature. However, in the two's complement number system, the unitary complement has to be carried out through the adder. Thus, a double penalty will be imposed on this operation. This is because the operation will not only require the bus and adder for a clock cycle, but it also requires that the multiplier-adder pipeline be emptied and results in low functional unit utilization.

A closer look at the unitary complementation in the DKS reveals that there are actually two different sources of this operation. One comes from the "pseudo-matrix-multiplication", as described in Chapter 4, where the third row of the resultant column (b_2 or b_3) may require complementation depending on the relevant element of the rotation matrix. Since five successive matrix multiplications are performed to obtain the DKS, the complement operation can actually be combined in the previous or next matrix multiplication depending on which is more convenient. Therefore, by carefully manipulating the relevant additions and subtractions in the neighboring matrix operations, the complement operation due to this source can be eliminated.

The second source of the complement operation comes from the process of sine function generation. In order to reduce the interpolation table size, only the interval values from 0 to 180 degrees are stored. Therefore, when the argument is greater than 180 degrees, the result needs to be complemented. With the table look-up and interpolation method, the sine value is obtained by

$$\text{SIN}(x) = T(x_U) + x_L \cdot D(x_U), \tag{5.1}$$

where x_U and x_L denote the upper 8 bits and lower 3 bits of the argument respectively, and $T(x_U)$ and $D(x_U)$ are table values addressed by x_U. Obviously, -SIN(x) can be obtained by $-T(x_U)-(x_L \cdot D(x_U))$. In two's complement operations, -a is implemented by a'+1, where a' denotes the bitwise complement of a. While a two's complement adder can only

execute a+b or a-b and not -a-b in one cycle, it is easy to modify the adder so that it can also perform a'-b. The result may lose one digit's resolution, but for a table size of 256, the worst maximum interpolation error is about 2^{-15} (0.000019). The truncation of 1 in the last digit of an 18-bit word is equivalent to a loss in resolution of 2^{-17} and should be acceptable, especially since this occurs only when calculating -SIN(x). In short, with a small penalty on resolution, a two's complement implementation will be faster, the design effort will be relatively simple, and is thus preferred. Figures 5.1 and 5.2 show the schematic block diagrams of a Baugh-Wooley two's complement array multiplier and a two's complement adder.

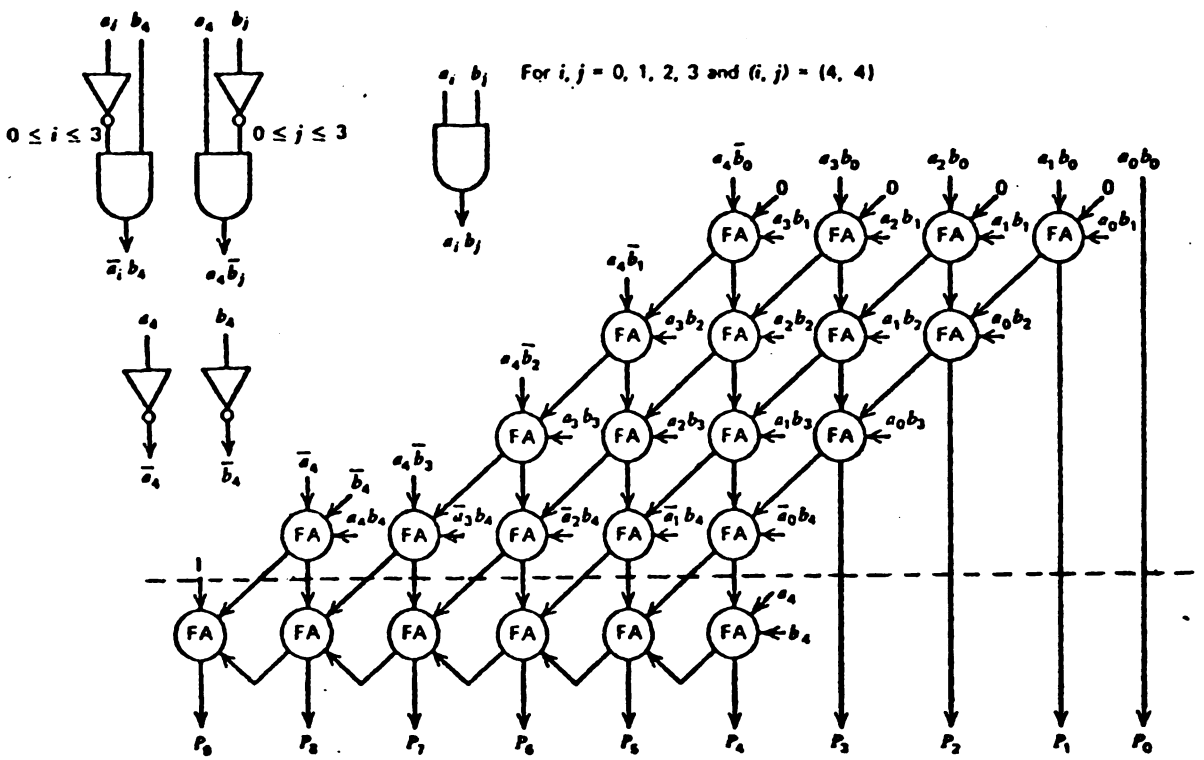


Figure 5.1. The block diagram of a 5-by-5 Baugh-Wooley two's complement array multiplier [35].

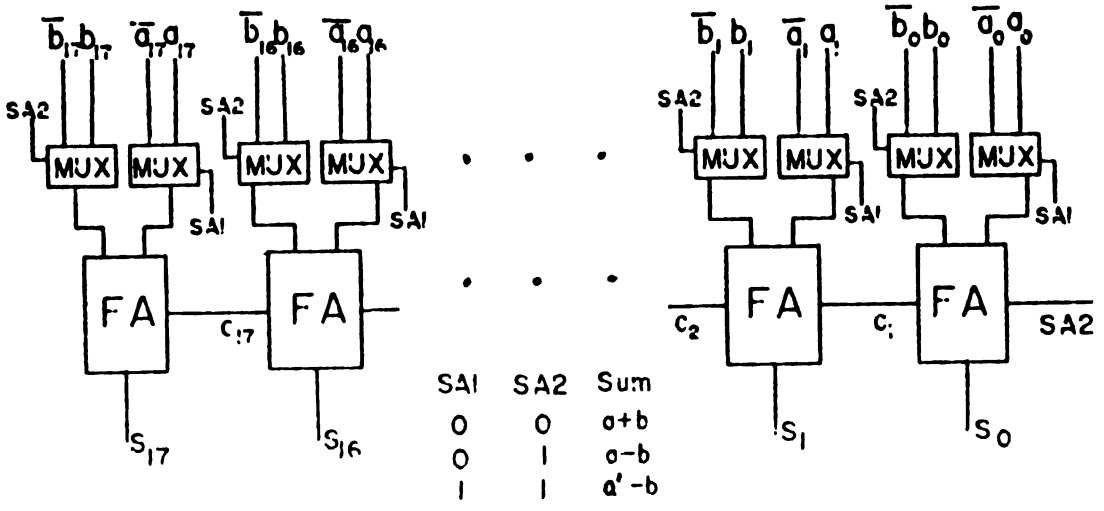


Figure 5.2. The block diagram of a two's complement adder used in the DKS chip.

5.2.2 Fixed-Point Calculation Implementation

Three data formats, as shown in Figure 3.3 exist in the DKS calculation. The integer values of the orientation vector element and the position vector element are their real values multiplied by 2^{17} and 2^{7} respectively, In other words, an implicit exponent of either 2^{-17} or 2^{-7} is implied for each integer value. To assure a correct solution, it is necessary to maintain the consistency of these implicit exponent values. According to the DKS algorithm developed in Chapter 4, the matrix multiplication of $A_t W$ is carried out by multiplying the rotation submatrix of A_t with each column vector of W, and the exponent value of the result is the same as that of the W element. Since the elements of the rotation submatrix are in the orientation vector format, the exponent value of the W element can be easily preserved by truncating

the 17 least significant bits and the most significant bit of the 18-by-18 two's complement integer multiplier output. This is illustrated in Figure 5.3. Since the implicit exponents are always the same for the two operands for the addition operations, no adjustment is required. The multiplication in the interpolation of the sine value will be addressed in next section.

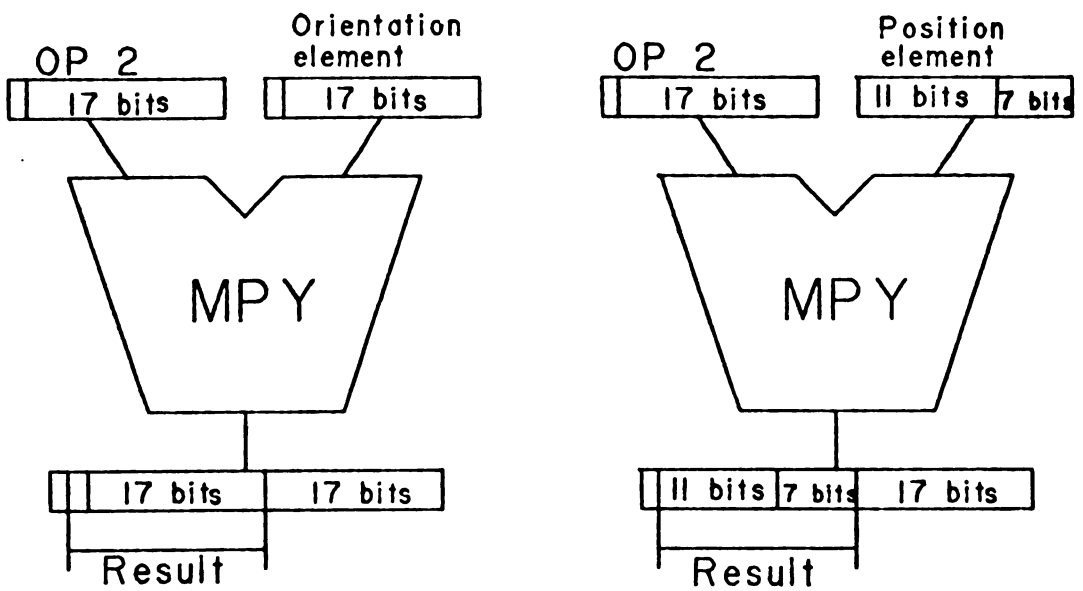


Figure 5.3. Illustration of the radix point position in the multiplication.

5.2.3 SIN/COS Implementation

The sine function needed in the DKS is approximated by the linear interpolation between the endpoint values of the interval in which the angle lies. In order to reduce the approximation error, the maximum error in each interval is calculated first. The endpoints are then adjusted such that the maximum error of each interval will be reduced to half of its original error as shown in Figure 5.4.

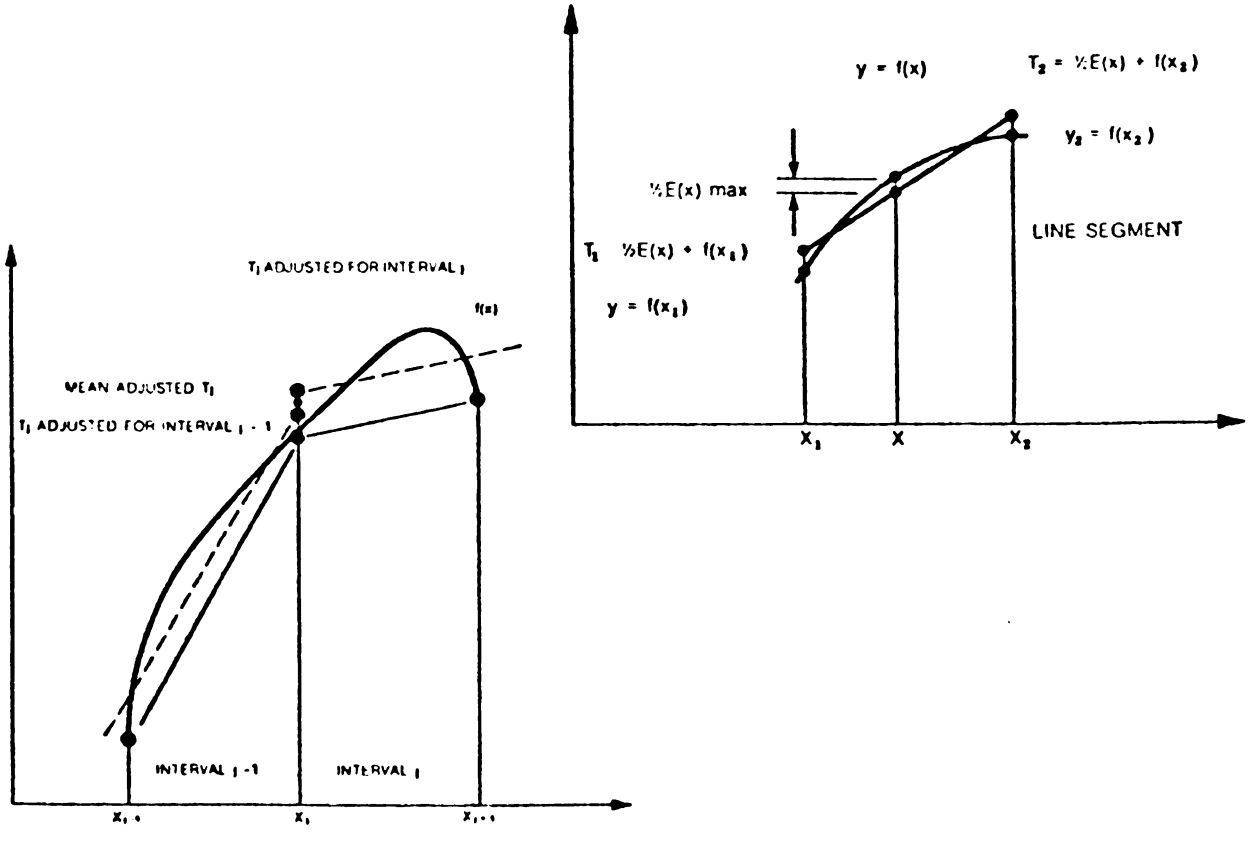


Figure 5.4. The table value adjustment scheme [26].

The interpolation formula is

$$f(x) = f(x_1) + [(x-x_1)/(x_2-x_1)][f(x_2)-f(x_1)]. \qquad (5.2)$$

In the above equation, x_1, x_2 are the interval numbers determined by the upper 8 bits of the argument and the term $(x-x_1)/(x_2-x_1)$ is simply the lower 3 bits of the argument after the radix point. Accordingly, both $f(x_1)$ (the left endpoint of the interval) and $[f(x_2)-f(x_1)]$ are stored in the table and denoted as T(x) and D(x). The table is generated for an angle argument range from 0 to 180 degrees and stored in a ROM. In the fractional-turn representation of an angle, the 1's in the two most significant bits correspond to 180 degrees and 90 degrees respectively. Since SIN(180+x) = -SIN(x), the sign bit of the angle is used to control the complement operation. Since COS(x) = SIN(90+x), the cosine function is obtained by adding one to the second significant bit of the angle argument. A special combinational logic circuit shown in Figure 5.5 is designed to implement the above logic operations under the control signal s/c.

Since fixed-point calculation is used and an 18-bit word can represent a two's complement integer number from -2^{-17} to $2^{-17}-1$, SIN(90) is set to $2^{-17}-1$, the largest integer value, thus preventing overflow. The multiplier assumes that one of the input operands has the implicit exponent value of 2^{-17} and the result will have the same exponent value as that of the other operand. Since the implicit exponent of the interpolation result is expected to be 2^{-17}, both D(x) and

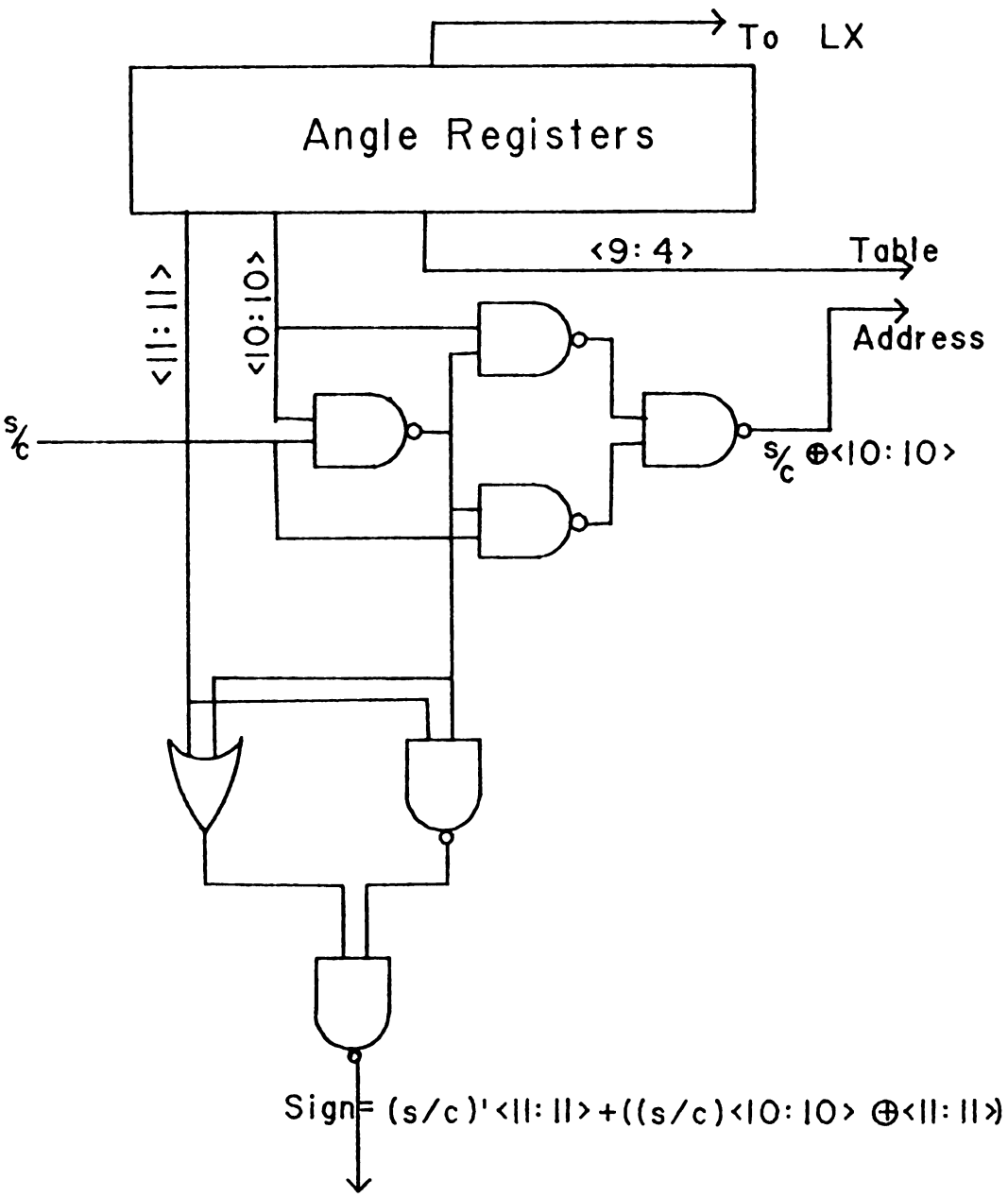


Figure 5.5. Logic circuit diagram of the SIN/COS control section.

$(x-x_1)/(x_2-x_1)$ of eq. 5.2 must have the combined exponent value of 2^{-34}. The lower 3 bits of the angle argument can be hardwired to the upper 3 bits (not including the sign bit) of the multiplier input latch. In practice, the exponent values of D(x) and $(x-x_1)/(x_2-x_1)$ can be adjusted to yield a more accurate result.

According to design rule 2, data flowing through stages of the C/L section must be separated by latches. It requires a total of three stages to generate the SIN/COS functions. During the first stage, the angle argument is fetched with its upper part flowing through the SIN/COS logic section to generate the table address and its lower part latched by a delay element. During the second stage, table values $T(x_U)$ and $D(x_U)$ are transferred via the dual bus to another delay element and the multiplier input respectively. During the last stage, the product of $D(x_U)$ and x_L is added to $T(x_U)$ to generate the desired function. The angle register output is connected to the SIN/COS logic section by a dedicated local interconnection path so that the angle fetch can be overlapped with other operations. The entire hardware design for SIN/COS generation is shown in Figure 5.6.

5.3 Timing

In the synchronous discipline of design, tasks are accomplished through a *logical sequence* of events with each event eventually realized under the constraint of the *physical timing* of transporting information (electrical signals) from one point to another. This dual meaning of

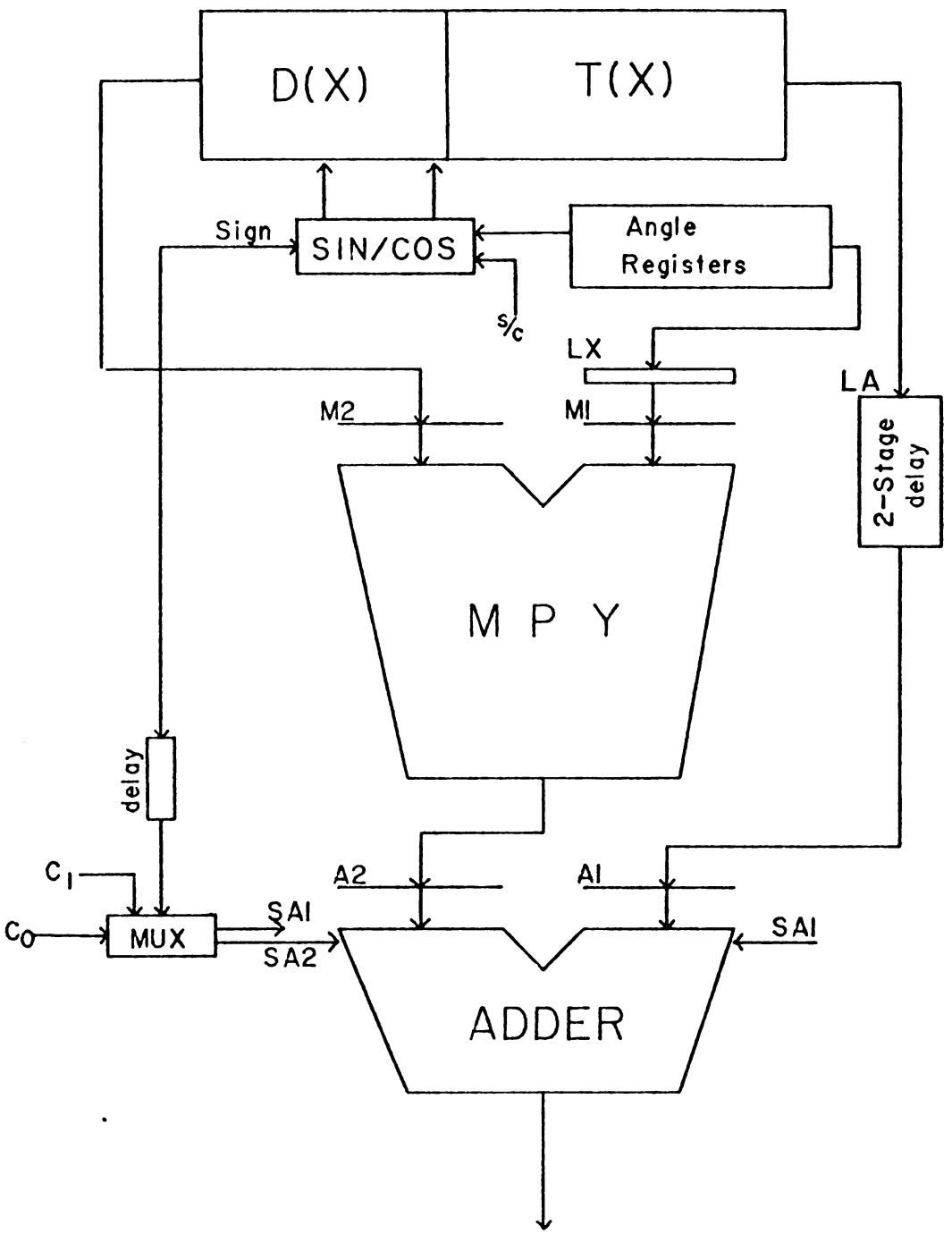


Figure 5.6. Block diagram of SIN/COS implementation.

timing is bound by a system-wide *clock* which serves both as the sequence reference and as the time reference. Hence the design of the clocking scheme has paramount importance to the system's correct functioning and performance.

5.3.1 Clock Period and Pipelining

The general form of a data path and the corresponding two-phase clocking scheme are illustrated in Figures 5.7 and 5.8.

As already seen in Chapter 4, the "pseudo-matrix-multiplication" involves repeated calculations of the form $a \cdot b + c \cdot d$; it is of great advantage to have a multiplier and an adder forming a pipe because the piped operation will save the bus bandwidth and storage of temporary results. Thus, for the DKS chip, the multiplier and adder become two major C/L sections in line.

From the timing diagram of the two-phase clocking scheme, it is clear that the minimum clock period is the sum of the maximum C/L delays of phase-one and phase-two plus some overhead of delay time and preset time. Intuitively, the C/L delay due to the multiplier may constitute the largest delay component in the minimum clock period calculation and thus one may further be motivated to introduce additional pipeline segments into the multiplier to shorten the clock period.

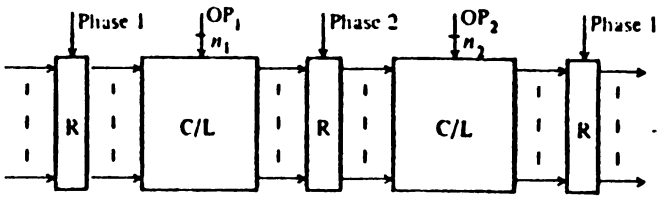


Figure 5.7. The general form of a data path [36].

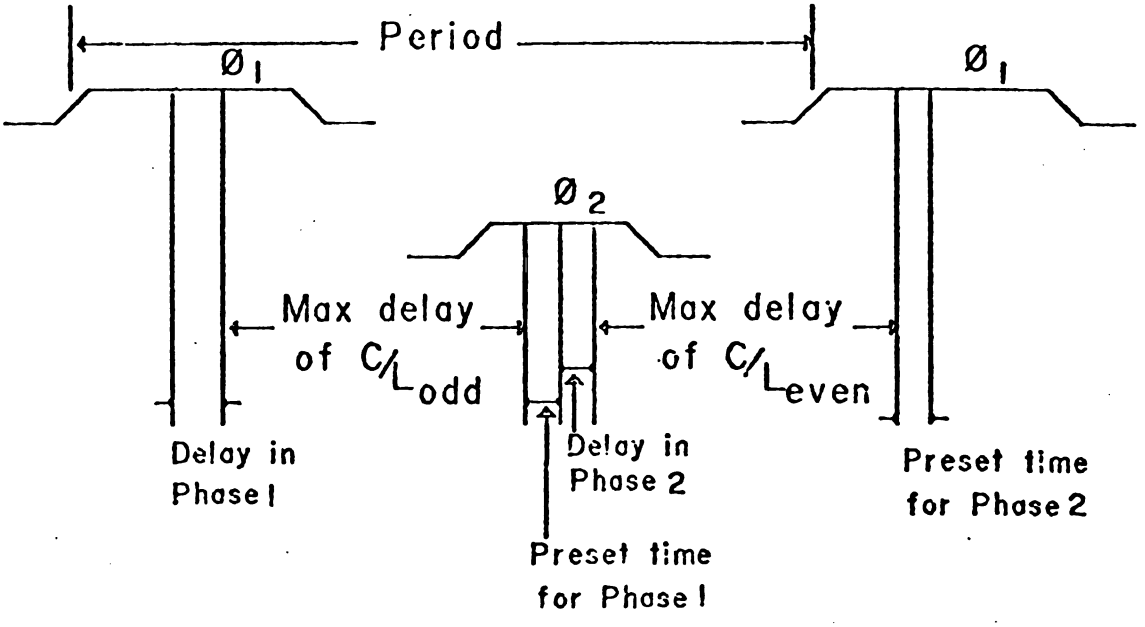


Figure 5.8. The two-phase clocking scheme.

While this idea is sound, there are at least two factors that should be taken into consideration. The first one is the scaling effect on the interconnect delay. Since it is uncertain at this point whether the DKS function can be realized on a single chip with 1-μm MOS technology, scaling may be necessary. However, as the feature size λ is scaled down by a factor of k, the gate delay τ is also scaled down by k, but the resistance of the interconnect path may scale up quadratically [37]. As a result, the interconnect delay may well become dominant and thus render the intra-functional unit pipelining effort meaningless. Recent studies show that refractory metal silicides can provide low-resistivity gates and interconnects. Taken together with appropriate interconnect scaling rules, the scale-down effects on interconnect delays may sustain down to a linewidth (2λ) of 0.5μm [38]. Therefore, in this work, if scale-down is necessary, it will be assumed that it is possible to reduce the interconnect delays linearly down to the feature size of 0.25μm.

Another factor is related to the particular choice of implementation technology. The estimation of various delays in this work is based on a full-custom design approach. Consequently, if the chip is to be implemented with a semi-custom approach, such as standard cell or macrocell implementation, the effect on the timing relationship of the C/L delay and the interconnect propagation delay is uncertain. Again, this suggests that the interconnect delay problem may possibly offset the intra-functional unit pipelining effort.

Nevertheless, the multiplier circuit can be divided into two stages quite naturally with the delay time of each stage about the same as that of a ripple-carry adder as indicated by the dotted line in Figure 5.1. An about-equal delay time of all C/L sections allows the system to achieve a higher degree of efficiency for a given clock period. Also, interconnect delay in this case seems very unlikely to exceed the delay of the adder.

5.3.2 Clocking Scheme Alternatives

Given the two-stage multiplier and an adder forming a pipeline, several clocking schemes are applicable. The first scheme is a brute-force application of the Mead and Conway approach as shown in Figure 5.9.

Figure 5.9 shows that the clock period is essentially the multiplication time plus some delay overhead. The advantage of this scheme is that when addition alone is needed, it only takes one cycle to empty the multiplier. Also, the second stage delay can be reduced by using a faster adder such as a CLA (Carry Look-Ahead) design. However, each C/L section is actually activated only half of a clock cycle with this clocking scheme.

To overcome this shortcoming, the C/L section can be selected to operate either at phase one or phase two. Since C/L sections are considered timeless and design rule one states that phase one inputs come from phase two outputs and vice-versa, additional latches are

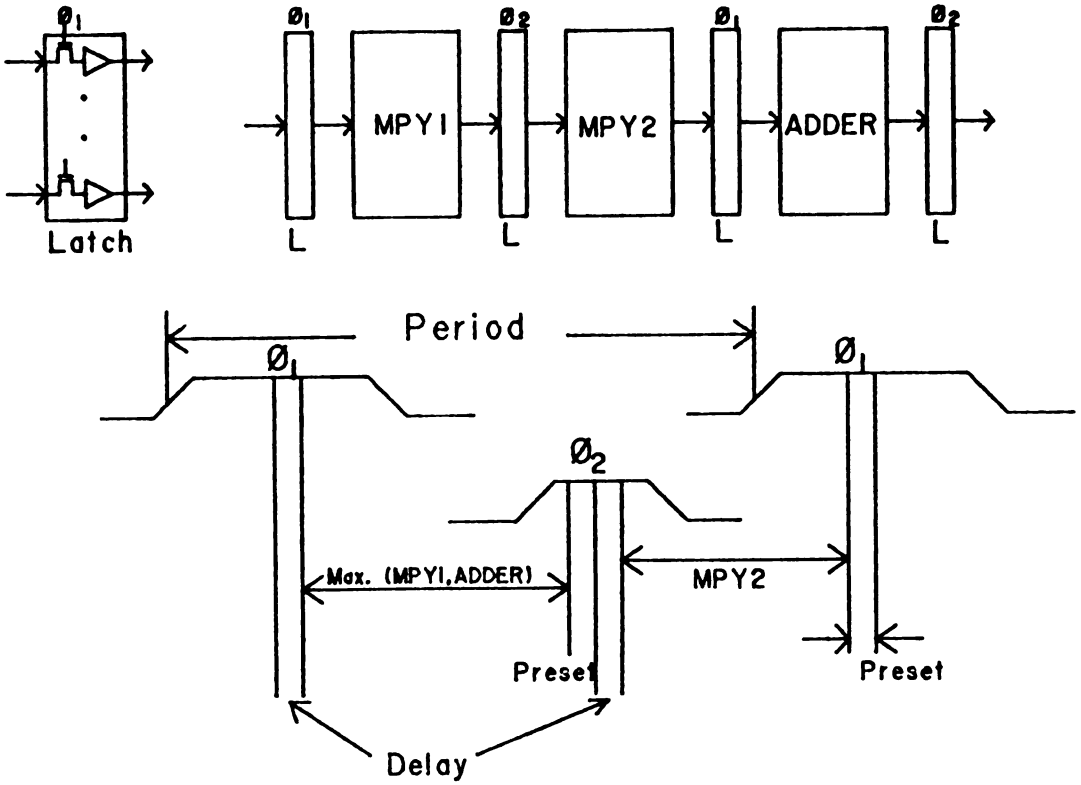


Figure 5.9. Clocking scheme one using single phase latches.

needed. Figures 5.10 and 5.11 illustrate two alternative clocking schemes that allow the C/L sections to operate in a single phase.

A requirement for the architecture of the second clocking scheme is that the clock period not exceed the refresh period of the dynamic storage elements used on the chip. In contrast, instead of making that assumption, the third clocking scheme accommodates the refresh rate consideration into the calculation of the clock period and is thus more flexible. Compared with the first clocking scheme, these two schemes have the apparent advantage of a shorter clock period with a slight increase in chip area. However, caution must be exercised for now two clock cycles are needed to empty the multiplier. Furthermore, data now

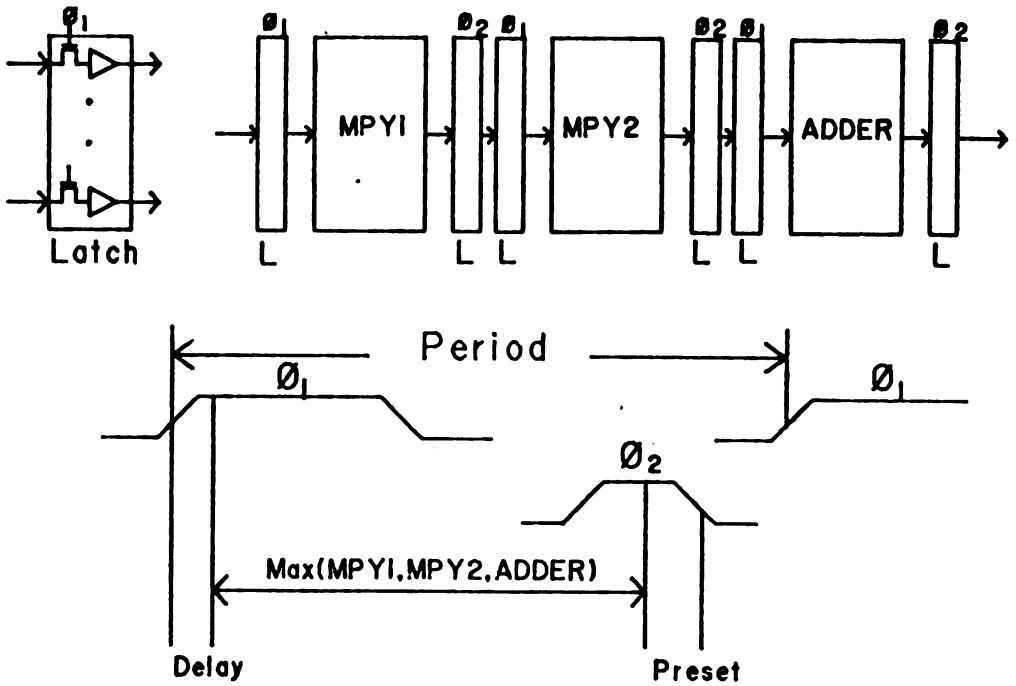


Figure 5.10. Clocking scheme two using single phase latches.

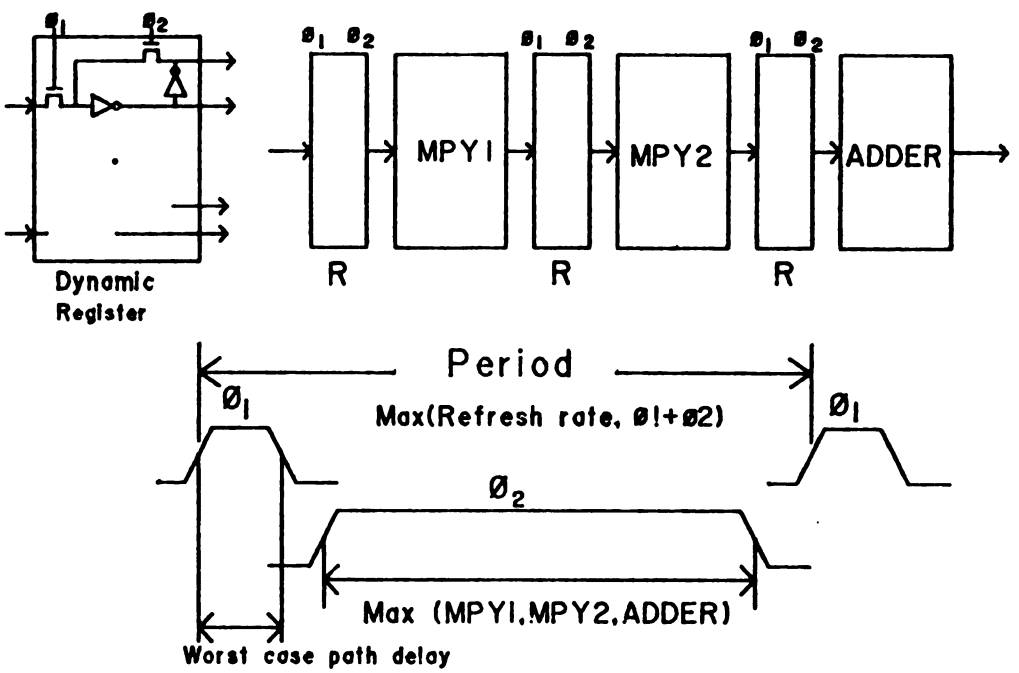


Figure 5.11. Clocking scheme three using dynamic registers.

take three cycles instead of two to pass through the entire pipe. This implies that if the degree of data dependency of the algorithm is high, the faster clock, stemming from the increase of pipeline segments, may not be advantageous. Since the second and third schemes are essentially the same with the third being more flexible, the study is focused on the trade-offs of the first and third clocking schemes.

5.3.3 Timing Diagrams

Given the basic computational structure and the two clocking scheme alternatives, the task now is to realize the desired algorithm within this framework. This is the logical aspect of timing and can be mastered efficiently by the aid of timing diagrams.

For full illustration of design options, the data flow timing diagram for this design should also provide some mechanisms for manipulating the data flow and be useful for later design work (for example, the interconnection solution and the RTL-programming). Accordingly, the timing diagram for this design work consists of three resource components: the dual bus, the multiplier, and the adder. Both adder input and multiplier input are identified by their logical meaning rather than their functional unit name. Thus, the logical sequence of the DKS can be easily followed. The add/subtract operation of the adder is also explicitly specified in the diagram to facilitate the manipulation of the neighboring matrix multiplication as discussed in Section 5.2.1. For each bus, both source and destination are identified

in the diagram. This proves very useful because the data flow pattern is soon discovered. By manipulating the data flow and modifying the interconnect path, the total DKS calculation cycles are reduced from 83 in the original design to 68 for clocking scheme one.

Figure 5.12 illustrates the timing diagram for one pseudo-matrix multiplication for clocking scheme one. The elements of the input matrix are logically identified as W1 to W9 but physically stored in R1 to R9 and may or may not be in sequential order. At the beginning of each iteration, sine and cosine values are calculated first and stored in Creg and Sreg respectively. A local interconnect path is assumed so that the result from the adder can be sent to the output register file without using an internal system bus.

Another timing diagram is constructed for clocking scheme three and the minimum total clock cycles for the entire DKS computation is found to be 73. Compared with the first clocking scheme, the data path of the third clocking scheme requires more area due to the additional dynamic registers including one needed for the delay of the T(x) term in the sine interpolation process. The resultant increase of the chip area, however, will be small compared with the total chip area. Even for a very unlikely 5% increase in total chip area, the time-area product of the third scheme will still be smaller than the first one if the clock period can be reduced at least 12%. By inspecting the timing diagrams of Figures 9 and 11, this reduction is apparently achievable. Thus, the third clocking scheme is adopted for the DKS chip. The entire DKS timing diagram with clocking scheme three is shown in Appendix 1.

$$\begin{bmatrix} w_1' & w_4' & w_7' \\ w_2' & w_5' & w_8' \\ w_3' & w_6' & w_9' \\ (0) & (0) & (1) \end{bmatrix} = \begin{bmatrix} C_3 & 0 & S_3 & a_2 \\ S_3 & 0 & -C_3 & 0 \\ 0 & 1 & 0 & 0 \\ (0) & (0) & (0) & (1) \end{bmatrix} \begin{bmatrix} w_1 & w_4 & w_7 \\ w_2 & w_5 & w_8 \\ w_3 & w_6 & w_9 \\ (0) & (0) & (1) \end{bmatrix}$$

After Multiplication:

R1 = w_1'

R2 = w_3'

R3 = w_2'

R4 = w_4'

R5 = w_6'

R6 = w_5'

R7 = w_7'

R8 = w_8'

R9 = w_9'

Before Multiplication:

R1 = w_1

R2 = w_2

R3 = w_3

R4 = w_4

R5 = w_5

R6 = w_6

R7 = w_7

R8 = w_8

R9 = w_9

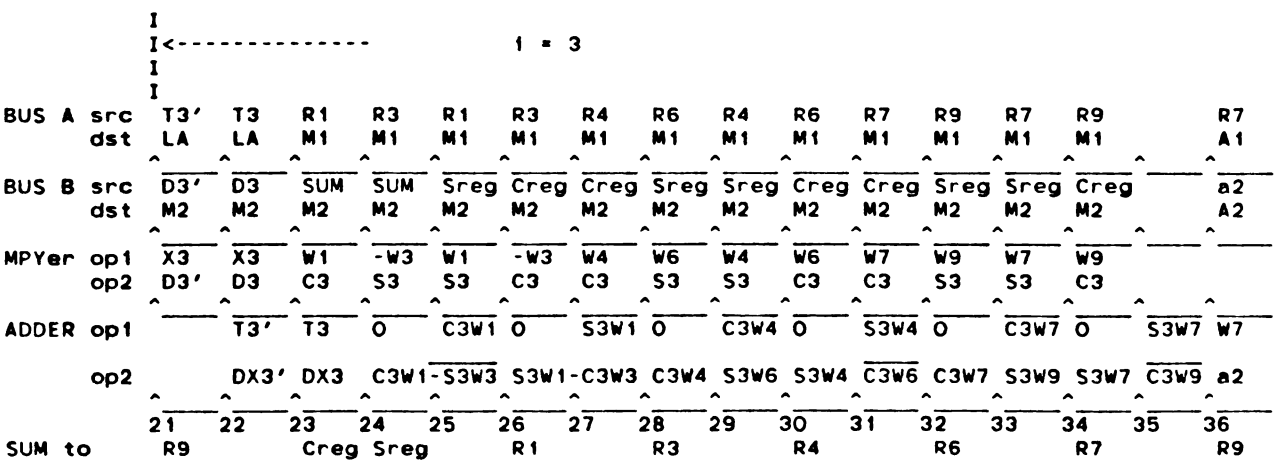


Figure 5.12. Timing diagram of one pseudo-matrix multiplication.

5.4 Architecture Development II

5.4.1 Synthesis of the Computational Structure

Now that the data path structure and clocking scheme are defined, the architecture design becomes a synthesis of previous sub-task development with some necessary refinements and modifications.

Due to the pipeline structure, delay elements are needed to synchronize the sine generation mechanism. The delay elements can be implemented by the same dynamic registers shown in Figure 5.11. Most of the input registers of functional units have more than one input source. The circuit diagram of a register cell capable of handling multiple sources is shown in Figure 5.13.

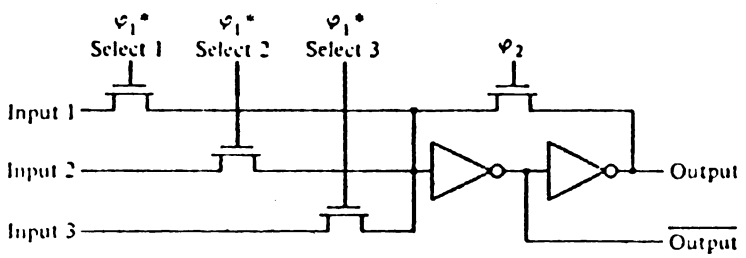
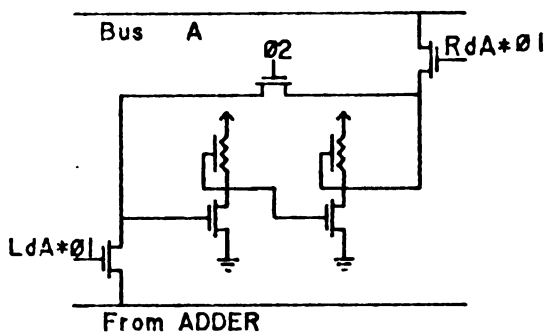


Figure 5.13. Circuit diagram of an input register cell with multiple sources [39].

For MOS technology, the propagation delay of a device increases as the number of loads increase. Therefore, if the number of device destinations in a connection path is more than one, all the destination latches or registers must be able to disconnect from the path. The register file is required to send information to bus A and receive a result from the adder output simultaneously (but not for the same register) as seen in the timing diagram. A two-port register cell with such capability is shown in Figure 5.14. And finally, the entire DKS chip architecture is presented in block diagram form in Figure 5.15.



Ø1		Ø2	Operation
RdA	LdA		
1	0	0	Bus A ← Reg
0	1	0	Reg ← ADDER
0	0	1	Reg ← Reg

Figure 5.14. A two-port register cell and its logic truth table.

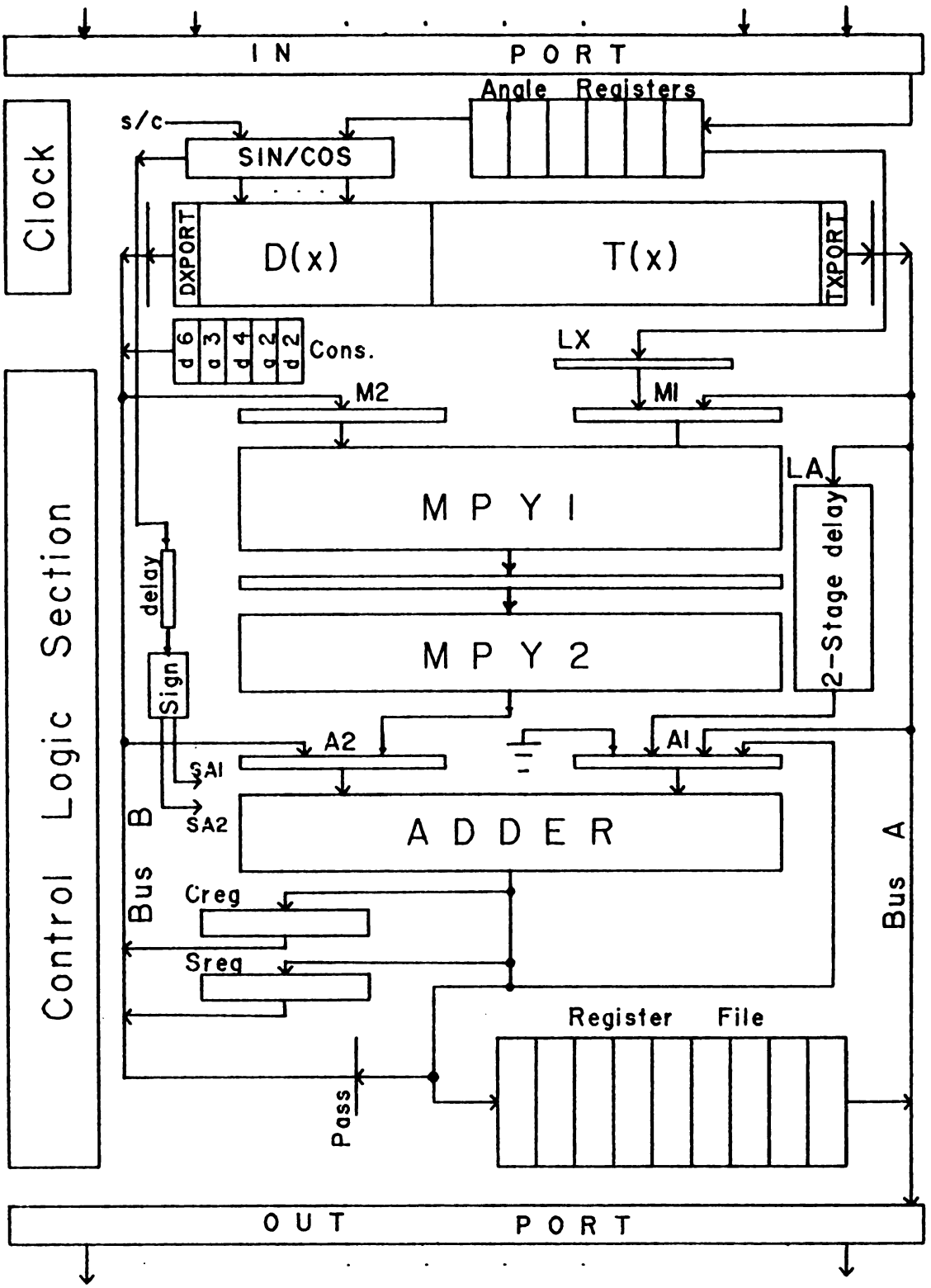


Figure 5.15. Block diagram of the DKS chip.

5.4.2 Description of Hardware in RTL

The block diagram of Figure 5.15 presents only the structural aspect of the chip architecture. The functional aspect, or the sequence of events ultimately realized by a control structure, is yet to be specified. Hardware Design and Description Language (HDL) is widely used to specify the functional, as well as structural, aspects of a digital hardware design and can be used as input to some hardware compilers. Because of their strong dependency on technology and developer, existing languages often evolve from a small design community and tend to be specialized [12]. Recognizing this, major research efforts are being made to standardize the HDL so as to strengthen the VLSI design process [40,41]. In view of this situation, this work uses RTL to describe the control flow structure of the DKS chip primarily for communication purposes. Furthermore, it is assumed that the RTL program presented may later be translated to other similar languages for implementation when necessary. The complete listing of the RTL program can be found in Appendix 2.

Chapter VI

CONTROL AND DESIGN VERIFICATION

Before the design process can proceed further, a legitimate question is whether the architecture of Figure 5.15, under the control structure described by the RTL program, will provide the desired DKS function. In answer to this question control signals are first identified and generated for the 73 clock cycles. These control signals are then used to drive a symbolic simulation program to confirm the correct behavior of the system.

6.1 Control Signal Specification

Even though the RTL program is created mainly for communication purposes, in practice, it is increasingly used for implementation purposes. In fact, control logic can be synthesized from an RTL program input with the capabilities of today's CAD systems. With this in mind, the question from a system designer's point of view is clearly not *how* the control logic is implemented but *what* the control signals should be.

It is obviously desirable to minimize the number of control signals as it implies less chip area. However, in deriving the needed signals from the RTL program, one must be aware that implicit states of certain functional entities may exist. For example, loading or transmitting a register is explicitly instructed, but what is not explicitly instructed is the implied state of neither loading nor transmitting and the

forbidden state of loading and transmitting simultaneously.

Such a problem can be solved by further clarifying the operation rules of various functional entities. For the design of the DKS chip, the following rules are observed:

1. no register is allowed to load and transmit at the same time;

2. no more than one source is allowed for any interconnection or register at a time;

3. the system bus can get data from only one source and give data to only one destination during any one cycle.

Another intriguing issue is the trade-offs of implementing the control signals with a horizontal, vertical, or a mixed policy [42]. In order to facilitate the generation of the control signals and the later verification process, the control signals are organized into fields and subfields. By assigning a meaningful name to each field, the control function of each signal can be easily captured. The way in which the control signals are partitioned, however, depends on which policy is chosen.

By employing decoding techniques, the vertical approach packs more functionality into a control word of a given size and thus has better chip area utilization. But it suffers a slower response time due to the decoding process. It also tends to make the encoding and simulation process more complex. On the other extreme, the horizontal approach

embodies maximum parallelism in generating the control signals and thus has a speed edge. But the waste of area is formidable for this approach even when a register file of moderate size is considered.

At first glance, this looks mostly like an implementation problem and therefore should not be determined during the design stage. But in practice, whenever a control field is defined, a certain amount of decision on the policy is already implied. In order to reduce the design effort, a mixed policy adapted from the horizontal approach is chosen in the partitioning of the DKS control signals. With this policy, a control field is created for each functional element. If a control field requires more than four bits, the vertical policy is employed within that particular control field.

Based on the above principle, the control signals required for the DKS chip are partitioned into 11 fields with each field further divided into two subfields for easy encoding. The 11 control fields with their signal definitions are shown in Table 6.1. Once the control signals are organized and defined, the specification of control signals for each clock cycle of the entire DKS calculation is reduced to a compilation of the RTL program into machine codes. This step is done manually in a straightforward manner. The control words, or machine codes, for the 73 cycles are listed in Appendix 3.

Table 6.1. Control fields and signal definitions.

No.	Field	Sub	Range	Definition
1	ADRTBL	Ev	0/1	1: TXPORT←T(x); LX←ANGLE(V)
		V	0-6	0: Angle Register output in high z state
2	SINCOS	s/c	0/1	0(1): Select SIN(COS) operation
		Et	0/1	1: BUSA←TXPORT; BUSB←DXPORT; LA←BUSA
3	ROM	EM	0/1	1: BUSB←CONS(Madr)
		Madr	1-5	Constants address
4	BUSAR	T	0/1	1: BUSA←REG(Radr)
		Radr	1-9	Register address
5	RSUM	L	0/1	1: REG(Radr)←ADDER
		Radr	1-9	Register address
6	MPY	M1	0-2	1: M1←LX
				2: M1←BUSA
		M2	0/1	1: M2←BUSB
7	A1	A1zd	0-2	1: A1←LA
				2: A1←0
		A1bs	0-2	1: A1←ADDER
				2: A1←BUSA
8	A2/G	A2	0-2	1: A2←MPY2
				2: A2←BUSB
		Gate	0/1	1: BUSB←ADDER
9	MINUS	CTRL	0/1	0: SA1,SA2←SIGN (from SIN/COS)
				1: SA1←0; SA2←SGN
		SGN	0/1	0: ADDER←A1+A2; 1: ADDER←A1-A2
10	Creg	Lc	0/1	1: CREG←ADDER
		Tc	0/1	1: BUSB←CREG
11	Sreg	Ls	0/1	1: SREG←ADDER
		Ts	0/1	1: BUSB←SREG

6.2 Verification by Symbolic Simulation

Many tasks of the DKS chip design, from the construction of the timing diagram to the manipulation of data flow, and from the development of the RTL program to its compilation into machine codes, are performed manually. The flexibility of human activity in many cases complements the limitation of available resources; but it is also prone to error. As the complexity grows, simulation increasingly becomes an indispensable tool for VLSI design.

Because the homogeneous transformation method is already well understood in the case of the DKS chip design, an exhaustive simulation is not necessary. Besides, even for an angle word size of six bits, the six joint angles form a total of $64^6 > 64X10^9$ joint angle vector inputs! An exhaustive simulation is plainly impossible. Generally speaking, symbolic simulation is more practical at the system level where the behavior of the system as a whole, rather than the correct functioning of an indvidual building block, is the major concern.

6.2.1 Simulation Principles

The validity of any simulation rests on the rationality of the mapping of the real system to the simulator. The behavior of the system can be thought of as the total sum of *events* occuring in various *entities*. From this perspective, entities may be modeled as program

variables while events are simply program statements. These two axiomatic correspondencies in turn constitute the foundation of the higher correspondence between the logical sequence of events of the DKS chip and the logic structure of the simulation program.

Three kinds of entities are observed in the architecture of the DKS chip. *Memory* elements including static devices (such as ROM) and dynamic devices (such as registers), are readily compatible with variables and constants of a program. *Interconnects* serve as the media for data transfer between entities; they may be multi-source/destination (such as a system bus) or dedicated (such as a wire). *Functional units*, such as multipliers and adders, are timeless combinational logic circuits which receive operands from dynamic storage elements, perform the assigned functions, and generate results.

Events of the system fall in two categories, data transfer and logic operations. The former can be simulated by program assignment statements with little effort. The establishment of interconnects as an entity enable the data transfer event to be decomposed into two logical phases: from source to medium, and from medium to destination. Various rule-checkings can subsequently be classified in either phase and enforced. In contrast, no rule needs be checked for the logic operations; but the function simulated by manipulating the symbolic values through program procedures is slightly more involved even for events seemingly as innocent as add or multiply. As mandated by the

clocking scheme of Figure 5.11, all functions are accomplished by C/L occurring exclusively in phase two.

6.2.2 Program Development

Based on the observed correspondence between entities and variables, a unique variable of type character string is assigned to each of the entities in the DKS system. The symbolic value of a variable represents a data item. The property of timelessness of the C/L allows the separation of the outcome from the symbolic manipulation process. Therefore, the variable of a functional unit is only associated with the outcome. For a dedicated path, the only event is the invariable source to interconnect to destination, which is simply simulated by a single assignment statement. On the other hand, the multiple source/destination situation of a system bus requires rule-checking. This is facilitated by designating the first character of the BUS variable as a flag. A load-bus operation is legal only when the flag indicates the bus is empty('E') and subsequently the flag will be set to 'F' for full. A receive-from-bus operation will set either the 'E' or 'F' flag to '0', and an error message will be issued if this operation is attempted when the flag is '0'.

The event of data transfer is simulated by scanning all the load bus cases first followed by processing all the load storage-element cases. During phase two, dynamic registers or delay elements have their

phase-one value transferred to the phase-two variable before the logic simulation is performed. The precharge of the bus is simulated by assigning a special value of 'E?' to signal the undefined state.

The events of multiply and add are simulated by appropriate manipulation of the symbolic values of the data and the outcomes are stored in the variables associated with the functional units.

The sine function generation mechanism is simulated as a sum of four independent events. In one event, during phase two, the value of 'T'i and 'DX'i will be assigned to TXPORT and DXPORT, respectively, when the address table signal is fired. The angle 'X'i will also be assigned to latch LX. In another event, the multiplier will take LX and BUSB as its operand sources and the data in BUSA will be latched by the first stage of a two-stage delay element. Then, in another event occuring in phase two, if the multiplier procedure detects the input operands having the symbolic values of 'X'i and 'DX'i, it will abort the normal multiply manipulation and assign 'DX'i to MPY1. The last event occurs when the adder procedure detects its input operand being 'T'i and 'DX'i; it will assign Si as the outcome stored in ADDER. It also checks the appropriate control field and the SIGN to guarantee that the adder control signal is correct. The cosine is simulated in a similar fashion with the help of a special character flag. Obviously, sine and cosine outputs will result only when a particular sequence of events occur.

The main program is a simple kernel driven by the machine code of Appendix 3. In order to help in the correction of design mistakes as

well as the debugging of the program, an error encoding technique is employed and a controlled dump of memory contents is provided. The flowcharts of the simulation program are presented in Figures 6.1-6.3. A complete listing of the simulation program can be found in Appendix 4. In Appendix 3, some of the signals are designated with the X (don't care) state. These conditions were converted into two machine code tables: one with X equal to 0 and another with X equal to 1. Two simulation runs, one with each table, are performed. Results are checked against the correct ones individually to prove the correctness. The simulation results are also shown in Appendix 4.

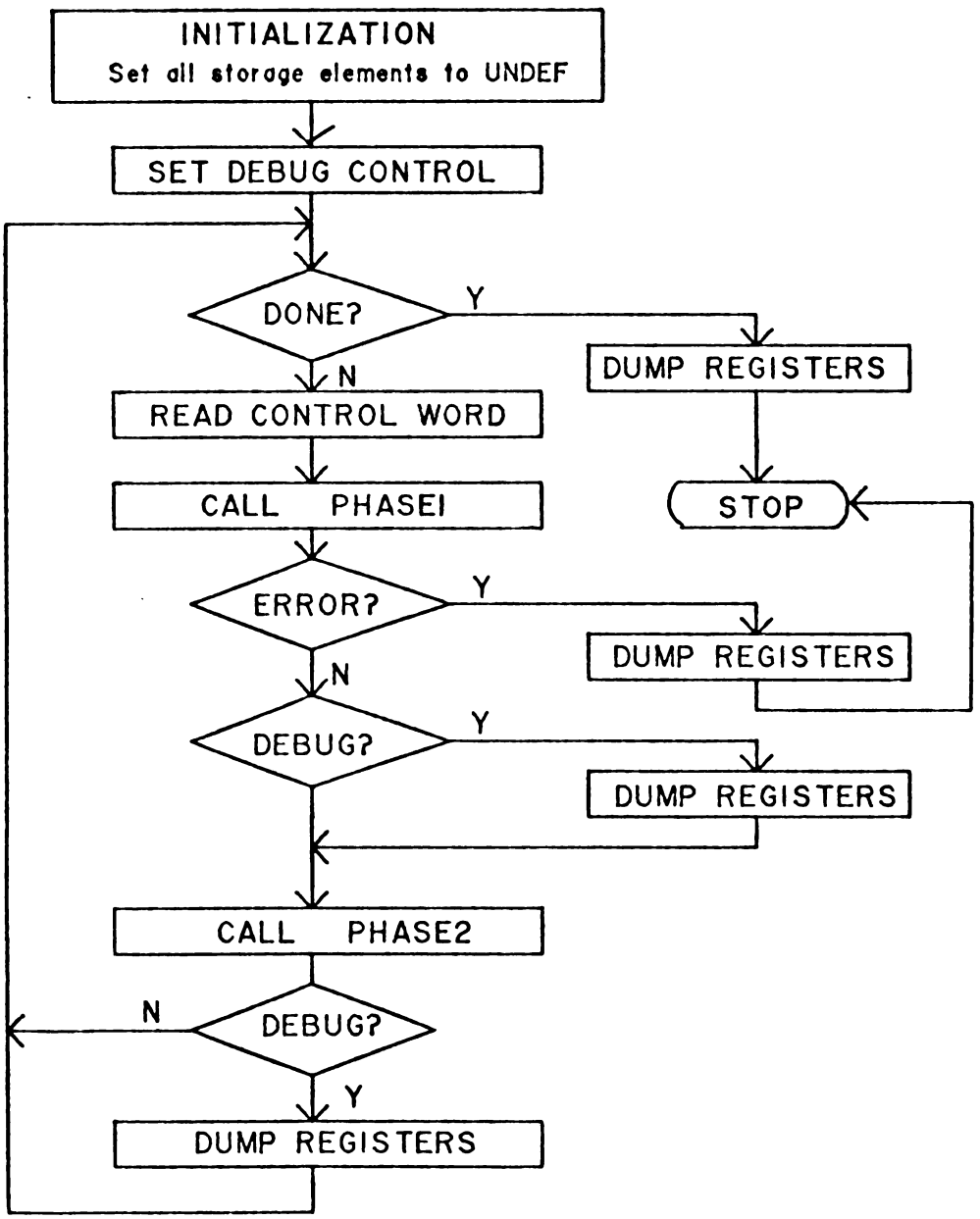


Figure 6.1. Flowchart of DKS symbolic simulation program.

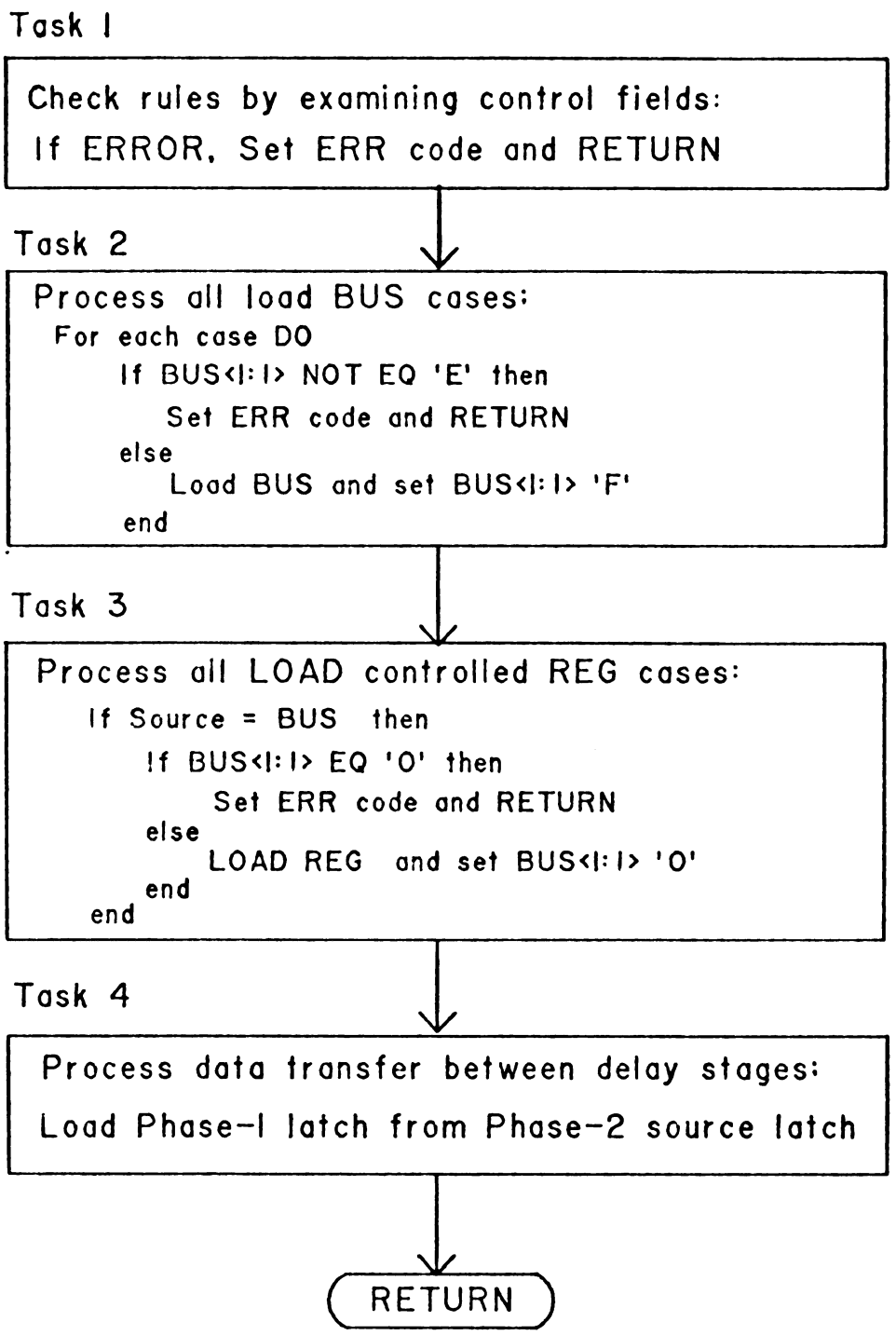


Figure 6.2. Flowchart of subroutine PHASE1.

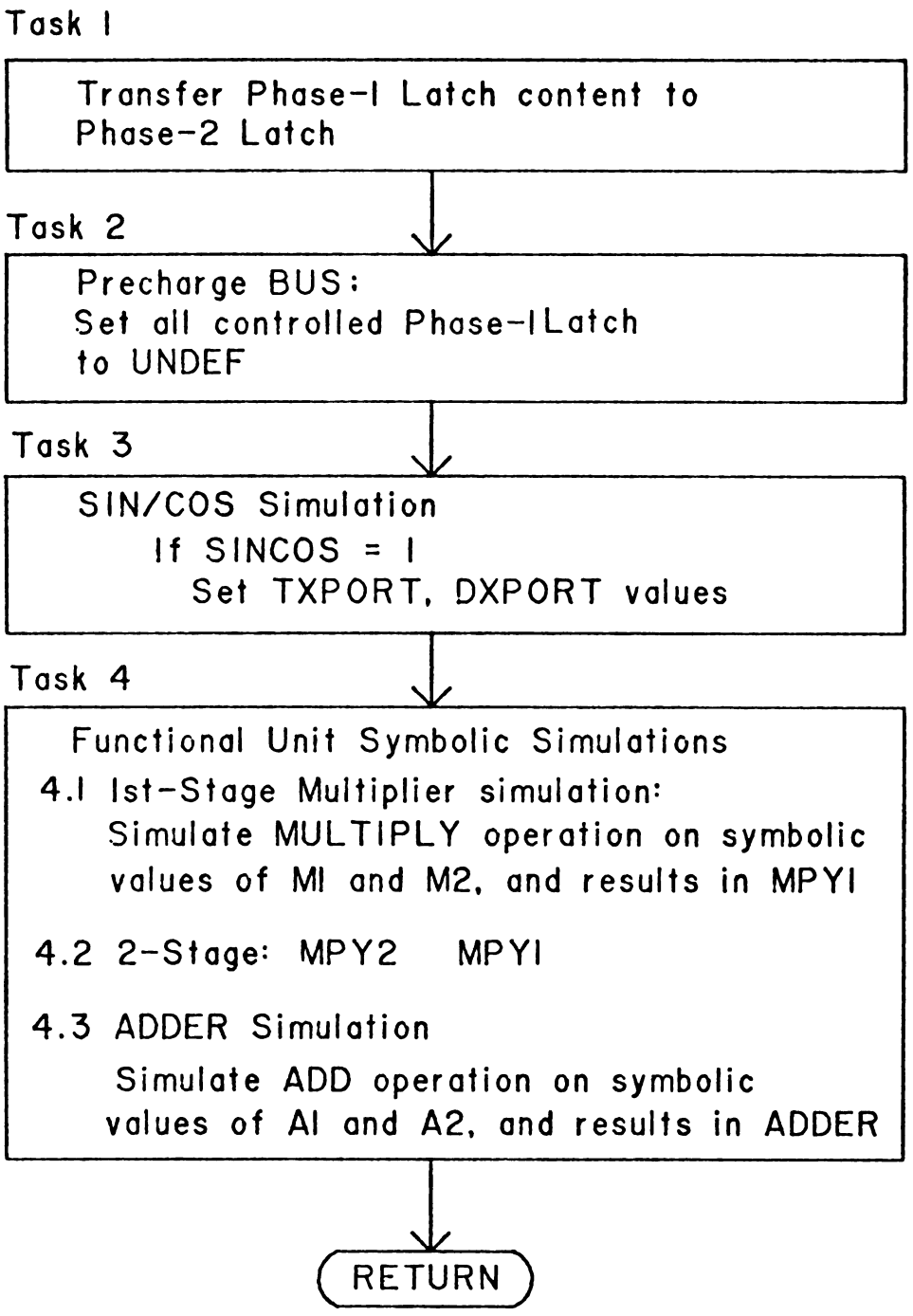


Figure 6.3. Flowchart of subroutine PHASE2.

Chapter VII

EVALUATION

7.1 Goal and Strategy

Based on the architecture presented in previous chapters, the precise performance of the DKS chip and area requirement must be determined by the physical design. Without the support of a physical design, any time/area evaluation is inevitably subject to suspicions. However, without even a qualitative estimation of the time/area complexity of the chip, a "go or no-go" decision on physical design is even harder, if not impossible, to make.

Since physical design is beyond the scope of this work, no attempt is made to acquire an accurate time/area evaluation. The goal here is to provide some basic statistics of the chip, and by comparing it with the capabilities of current technology, provide some sort of "confidence vote" on the feasibility of the DKS chip. In view of the industrial practice of prototyping with semi-custom design, special attention is paid to the possibility of the standard cell implementation approach. This approach has been predicted to become the dominant form of application-specific ICs [43,44].

Current technical information in the literature tends to be fragmented, and its technology-dependent nature makes it even harder to meaningfully use. At the time of this project, the only relatively complete information was the IBM Master Image Chip approach [8]. Both statistical data on functional units and cell utilization as well as

basic logic gate delays are available. The area of the DKS chip is evaluated by this approach. The area required for CMOS implementation is obtained from the NMOS figure according to the conversion rules suggested by Wollssen [45].

Semi-custom design usually requires more chip area. Therefore, the chip edge from the area estimation should be considered as an upper bound. The phase one clock width is evaluated by taking the chip edge as the worst case path for the propagation delay. Since the phase two clock width is determined by the maximum combinational logic delay, delay times from an actual multiplier IC and from gate delay summations are used to obtain a range of possible C/L delay values. The minimum clock period is then the sum of the two clock widths.

7.2 Area Estimation

The silicon real estate of a VLSI chip is used either for device fabrication or for interconnects. While the former is determined exclusively by circuit design, the later depends on many physical design factors such as layout, placement and wiring, which are beyond this design task. To overcome this difficulty, this work uses the physical device count as the major parameter, and the device count is then weighted with the transistor/cell utilization statistics obtained from published sources to get a realistic picture of the area required for the DKS chip including the interconnects.

7.2.1 Transistor Count

The transistor counts for various functional circuits in the DKS chip are shown in Table 7.1. All transistor counts are based on NMOS circuits taken from [3] except the adder circuit which is taken from [46].

The transistor count of the control logic is unknown until the actual logic design is carried out. A convenient way to get around this problem is to assume that the control logic is implemented by storing the machine code in a ROM with a counter generating the instruction sequentially. 37 individual signals are needed for the DKS calculation. This figure is rounded off to 40 to account for the I/O and other interface control functions. Since this approach generally requires larger chip area, the transistor count in the table may once again be interpreted as an upper bound of the control logic section.

Because transistors in driver circuits require larger area, they are counted separately. The table shows that the transistor counts of the regular logic circuits and the random logic circuits are about the same. Because of their regular nature, the former will occupy much less area. Also, the area for storing table D(x) can be further reduced as the first few bits of the D(x) values are always zeros.

The total number of transistors of the DKS chip is in the neighborhood of 25K, or about 10K logic gates. This figure is well under the projected limit of 50K logic gates for gate array implementation [47]. In fact, CMOS gate arrays of up to 24K logic gates

Table 7.1. Transistor count of the DKS chip.

Type	Function	No. of Transistors per Unit	No. of Units	Total No. of Transistors
Regular	TX(1 bit)	1	18X256	4,608
	DX(1 bit)	1	18X256	4,608
	Constant	1	18X5	90
	Control	1	40X73	2,920
			Subtotal	12,226
Random	Reg Cell	9	18X(9+6)	2,430
	Delay Cell	6	18X2+2	228
	controlled latches			
	1-source	7	18X2	252
	2-source	9	18X2	324
	4-source	13	18	234
	Pass Trans.	1	18X3	54
	Decoder			
	3-8	30	2	60
	4-16	72	2	144
	8-256	72	72X17+256	1,480
			Subtotal	5,206
	FA	18		
	MPY1	FA	18X17+2	5,544
	AND	5	2X17+17**2	1,615
	MPY2	FA	18	324
	ADD	FA	18	324
	2-1 MUX	2	18X2	72
			Subtotal	7,879
Driver	I/O Buf	12	16X2	385
	BUSA	5	18	90
	BUSB	11	18	198
	Control	8	55	440
			Subtotal	1,112
			Grand Total	25,232

have already appeared [48]. Therefore, the DKS chip can be implemented on a single chip with current semi-custom design technology provided gate utilization of higher than 50% can be achieved.

7.2.2 Macrocell Implementation

The gate array approach is ideal for implementing designs at the Boolean logic level but is not suitable for high performance system designs. To bridge the gaps of cost, design turn-around time and performance between semi-custom design and full custom design, the standard cell approach shows promising potential.

One such approach is the IBM Master Image Chip approach [8,49]. This approach uses a predesigned image with a fixed power bus distribution and preallocated areas for circuits and wiring. Well designed and tested functional circuits are stored in a macrocell

Table 7.2. IBM Master Image function-cell and chip statistics.

Function	No. of Cell
2-1 MUX	1
LSSD SRL	2
SRL CLK Driver	5
CLK Driver	3
CSA	6
4-bit Counter	10
16-by-16 Multiplier	630
256X16 ROS	200
128X16 ROS	120

Chip Item	Statistics λ=2μm	λ=1.25μm
Area(mm²)	7.4X7.4	8.0X8.0
Cell	2,432-2,952	8,468
Cell Util.	63-93%	79%
Gates	7.6K-11.8K	36K
Gate Delay	5-20ns	3.1-12.5ns *

* Estimated from figures of 2μm

library and can be called out during circuit design much the same way a digital designer selects an IC from a catalog of standard ICs. Design efforts are thus minimized and yet the system performance can be expected to approach that of fully custom designed ICs. Currently, the Master Image cell library contains only NMOS circuit designs; but CMOS implementation has been planned [8].

Table 7.2 shows the IBM Master Image Chip statistics used in the evaluation of the DKS chip. Table 7.3 shows the estimation of cell numbers. Linear dependence is assumed in the evaluation. For example, the 16-by-16 multiplier requires 630 cells, and the 18-by-18 multiplier of the DKS chip is assumed 18/16 times larger in each dimension, resulting in 798 cells. Note that the random logic represents 79.1% of the total cell count compared with only 51.7% of total transistor count.

Table 7.3. Cell count of the DKS chip.

Function	Cell Count
TX (256X18)	450
DX (256X18)	450
Constant	10
Control	100
Subtotal	560
Adder	108
18-by-18 MPY	798
Latches & Reg	828
Pass/ Mux	90
Counter	20
Clock Driver	165
SRL Driver	115
Subtotal	2,146
Total	2,706

In terms of a percentage of chip area, the 79.1% figure may be closer to reality. Assuming a 10% estimation error and a somewhat conservative 70% cell utilization, the number of cells required for the DKS chip is about 4.3K. The Master Image chip approach allows two feature size implementations: either 2-μm with 2,952 cells or 1.25-μm with 8,468 cells. Clearly, the DKS chip can be implemented with 1.25-μm technology. With half of the cell count, the chip edge of the DKS chip is estimated about 5.6mm.

For conversion of NMOS circuits to CMOS circuits, the rule of thumb is that the area of CMOS will be 30% larger than NMOS equivalent circuits for random logic but the same for memory cells [45]. Using these factors, the CMOS DKS chip will require a total of 5,317 cells, about 24% larger than the NMOS version. The resulant chip edge is estimated to be about 6.34mm.

7.3 Speed Estimation

The evaluation of the total DKS calculation time is based on average gate delays. The gate delay of a two-way NOR gate of the IBM Master Image approach using 2-μm NMOS technology ranges from 5-20ns [49]. Assuming linear scaling, the average gate delay for the 1.25-μm technology is 7.8ns. It should be pointed out that this figure is well below the state-of-the-art gate delay. For example, CMOS gate delays of 0.8ns using 1-μm technology has been reported [50]. What's more, the

limit of CMOS gate delay for 1-μm technology is projected to be 0.2ns [51]. Therefore, in this work, two values of gate delays are used. The IBM figure of 7.8ns is used mainly to provide a bottomline of performance while a 1ns/gate delay is used to represent what is actually achievable in the near term.

As shown in Figure 5.11, the minimum clock period is the sum of the clock widths of phase one and two. In calculating the minimum clock widths, empirical data is extracted from various sources and applied to the DKS chip design. Since the target feature size is 1.25μm as used in the estimation of the chip area, all timing data is adjusted according to the scaling rules.

For the interconnections, the delay is due to the RC time constant which is normally independent of scaling of the feature size λ. But, it may become inproportionally larger in the total delay calculation when λ is scaled down. However, improvements in conductivity of interconnect lines may be regarded as another scaling dimension. Combined with the separation of interconnect scaling rules from device scaling rules, the "scaling" of conductivity may justify the application of the same scaling factor to the interconnect delay [38].

According to the specified clocking scheme, the phase one clock width is the worst case path propagation delay. By inspecting Figure 5.15, the transfer of data from the register file to the input latch of the multiplier may be taken as the worst case path. The delay components include the register response time, bus propagation delay,

and the input latch response time. Table 7.4 shows the estimation of these three components. The gate counts are derived from the relevant circuits and the CMOS gate delay is assumed to be 1ns whereas figures in parenthesis are calulated from the IBM Master Image average gate delay. The total delay time for the phase one ranges from 20ns to 102ns.

The phase two clock width is the maximum combinational logic delay. In the DKS chip, the first stage of the multiplier pipe can be considered as the maximum C/L delay. Since the standard cell approach promises the ability of incorporating proven functional circuits as macros of the cell library, it is not unreasonable to assume that the multiply time achieved by today's multiplier IC is also achievable by the multiplier unit in the DKS chip. A survey of existing commercial multiplier ICs indicates that a 16-by-16 multiplier can be as fast as 45ns per operation [52]. An even faster multiply time of 20ns using

Table 7.4. Estimation of phase one delay.

Delay Component	Estimation Method	Delay Time (ns)
Register Response	3G + 4G + 2G * (Decode)+(Driver)+(Resp.)	9 (70.2)
Interconnect	6.34mmX1.25ns/mm @	8 (8)
Latch Response	3G	3 (23.4)
	Total	20 (102)

* 1 G = 1 (7.8) ns
@ Scaled down from 5ns/mm for 5-μm design rule [38].

1.5-μm NMOS design rules has been reported by Bell Labs [53]. Based on the somewhat moderate figure of 90ns per multiply-accumulate operation (from TRW's 1-μm CMOS 16-by-16 multiplier IC), a 100ns multiplication time for an 18-by-18 multiplier using 1.25-μm CMOS technology, appears appropriate. Since the delay times for the two stages of the multiplier are about the same, the phase two clock is set to half of the multiply time, i.e., 50ns.

On the other hand, if one full adder (FA) is assumed to require 6 gate delays, the 18-FA chain in the first stage of the multiplier pipe may require as many as 120 gate delays. Taking 1 (7.8)ns for one gate delay, the phase two clock width is estimated 120(936)ns.

Table 7.5 sums up the clock period and total calculation time estimations for various approaches and technologies. To achieve the goal of a 10μs calculation time, a gate delay of 1ns or less seems to be necessary.

Table 7.5 Total calculation times for various approaches.

Technology	Phase 1(ns)	Phase 2(ns)	Clock(ns)	Total(μs)
Fast Multiplier	20	50	70	5.1
Standard CMOS Cell (1ns/gate)	20	120	140	10.8
IBM Master Image (7.8ns/gate)	101.6	936	~1,000	73

Chapter VIII

CONCLUSION

8.1 Achievements

The advent of VLSI technology offers unprecedented freedom, as well as many challenges, for designers to integrate their knowledge and expertise into a tiny piece of material of utmost elegance. However, an ever-expanding gap exists today between system designers and the latest available technology [54]. One of the difficulties in bridging this gap arises from the fact that the design process at the different levels has different subjects of abstraction and requires different abstraction mechanisms. From this perspective, the greatest contribution by Mead and Conway to the VLSI system design is the idea of formulating a set of rules both as a way to express the subject of abstraction and as an abstraction mechanism. Following their lead, this work extends these ideas to higher level designs by proposing various rules as guidelines or templates for abstractions at different levels. The design of the DKS chip is subsequently treated as an instantiation of these rules. Although these rules will certainly be enriched as our understanding of VLSI design deepens, the accomplishment of this work is a verification that such an approach can potentially bridge the gap between the system designer's ability and the capability of technology.

In addition, this work develops a set of tools and techniques to facilitate the design at the system architecture level. Various mechanisms have been incorporated into the construct of the timing

diagram such that it not only demonstrates the data flow, but also allows for its manipulation. As to the development of simulation programs, the validity of the simulation approach has been rationalized. Based on the observation of the correspondencies between architecture elements and program elements, it proposes principles and rules to set down their relationship. Following these rules, a designer can map a design into a symbolic simulation program with relatively little effort.

This work also verifies the feasibility for fixed-point calculation of the DKS and further quantifies the number of bits required for the resolution in a standard industrial environment. The major structural and functional aspects of a VLSI chip architecture dedicated to the DKS computation have been specified. Results indicate that the design can be fabricated on a single chip using a semi-custom implementation approach with current technology.

8.2 Summary

From the timing diagram of Appendix 1, it is verified that the utilization of the bus, multiplier and adder reach 86.3%, 80.8% and 86.3%, respectively. They are somewhat lower than the first performance specification. This is mainly due to the introduction of pipeline segments into the multiplier which causes the data dependence problem to become more severe. In contrast, the first design, without pipelining inside the multiplier, achieves an average of 90% utilization.

The evaluation of the last chapter indicates that the total DKS

calculation time of 10μs is attainable provided the basic gate delay is 1.0ns or less.

In short, the architecture design presented in this thesis satisfies the major design objectives. The speed improvements, in retrospect, may be attributed to two fundamental improvements: (1) the increased speed of basic operations by using dedicated functional hardware and by inter- and intra- functional unit pipelining; and (2) the relief of the fundamental limitation of bus bandwidth by providing dedicated local interconnect paths.

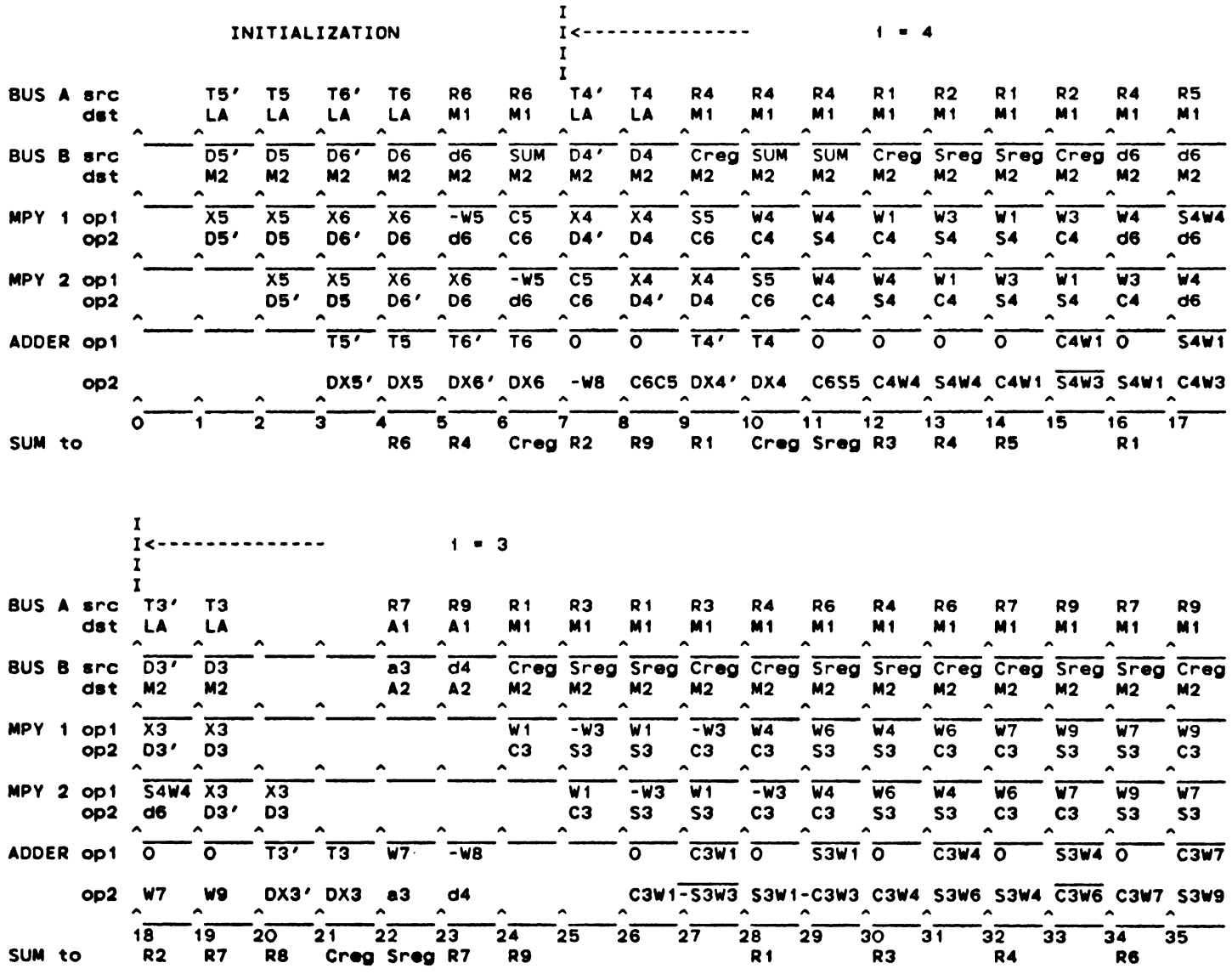
As stated at the beginning, this work is mainly concerned with the design specification at the system level. As a result, much work at the physical levels, such as logic design, circuit design and timing verification, remains to be done. At the system level, more pipeline segments may be introduced into the functional units. The I/O activities and mechanisms must be further specified. Depending on the data access time, I/O buffering may be desirable. All I/O and other interface control functions have to be incorporated into the control logic of the chip.

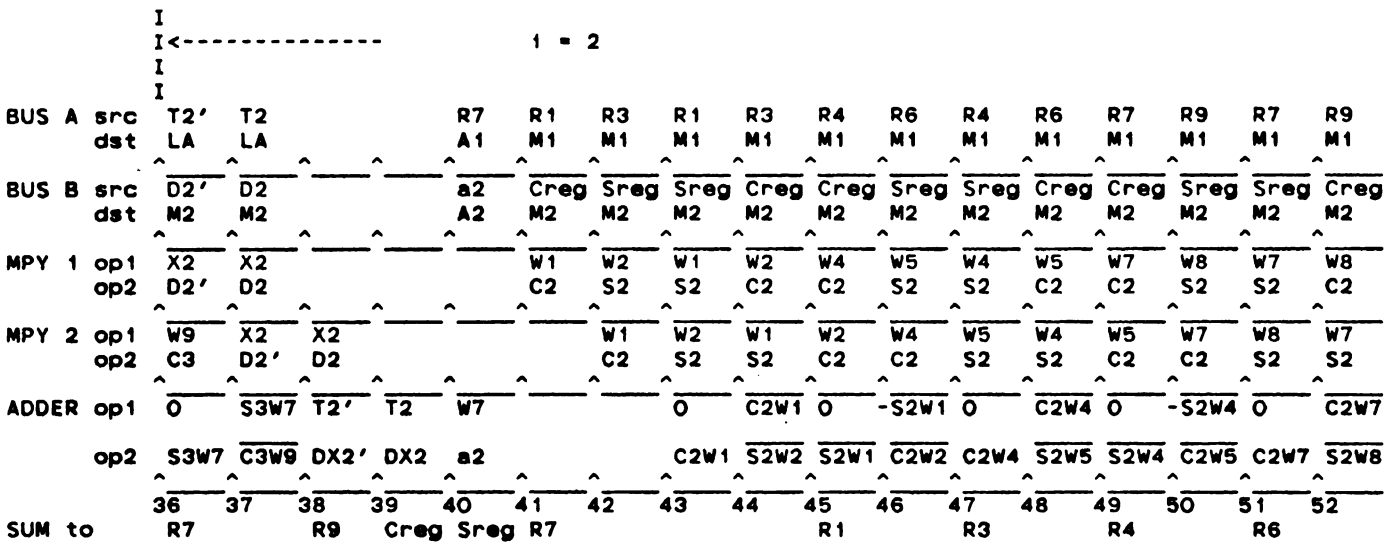
Beyond the implementation problem, the future utilization of the DKS chip may be more interesting. This work shows that with nascent VLSI technology, the DKS of a robot arm can be obtained in as little as 10μs. This result may have implications on the method of obtaining the Inverse Kinematic Solution (IKS). For example, with a modified algorithm using the DKS chip as its building block, the IKS may be approximated iteratively with superior speed.

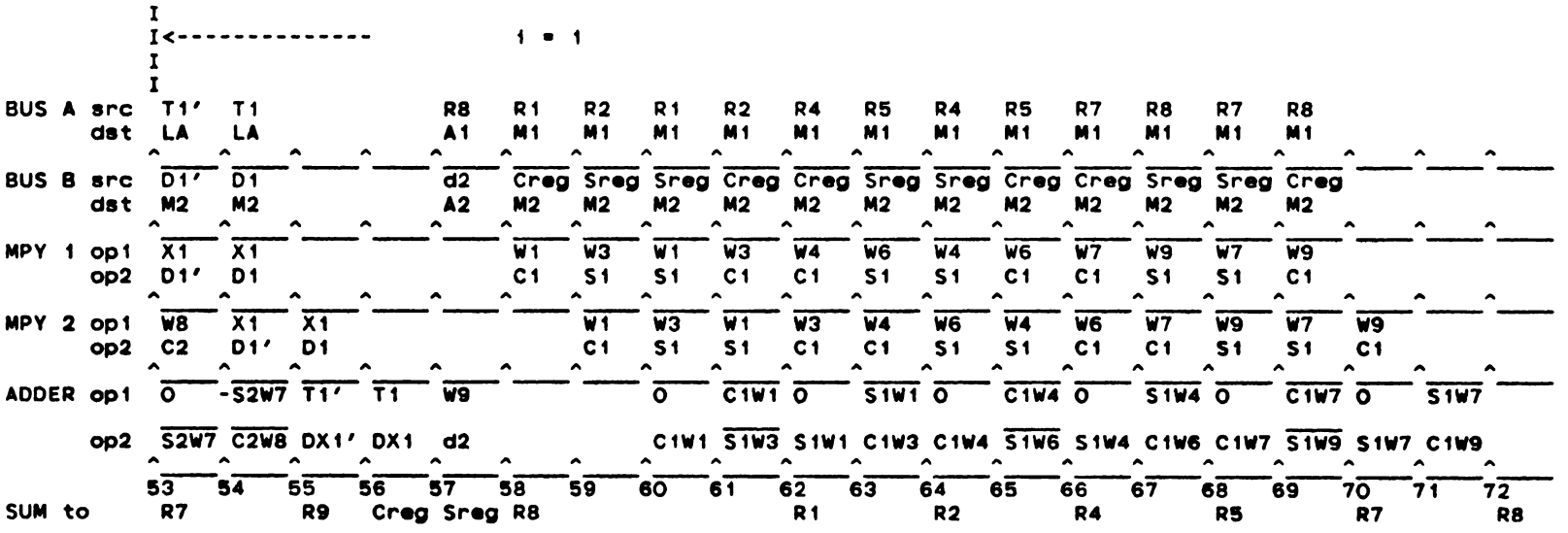
APPENDICES

APPENDIX 1

DKS TIMING DIAGRAM


INITIALIZATION
I
I<-------------- 1 = 4
I
I
BUS A src T5' T5 T6' T6 R6 R6 T4' T4 R4 R4 R4 R1 R2 R1 R2 R4 R5
dst LA LA LA LA M1 M1 LA LA M1 M1 M1 M1 M1 M1 M1 M1 M1
BUS B src D5' D5 D6' D6 d6 SUM D4' D4 Creg SUM SUM Creg Sreg Sreg Creg d6 d6
dst M2 M2 M2 M2 M2 M2 M2 M2 M2 M2 M2 M2 M2 M2 M2 M2 M2
MPY 1 op1 X5 X5 X6 X6 -W5 C5 X4 X4 S5 W4 W4 W1 W3 W1 W3 W4 S4W4
op2 D5' D5 D6' D6 d6 C6 D4' D4 C6 C4 S4 C4 S4 S4 C4 d6 d6
MPY 2 op1 X5 X5 X6 X6 -W5 C5 X4 X4 S5 W4 W4 W1 W3 W1 W3 W4
op2 D5' D5 D6' D6 d6 C6 D4' D4 C6 C4 S4 C4 S4 S4 C4 d6
ADDER op1 T5' T5 T6' T6 0 0 T4' T4 0 0 0 0 C4W1 0 S4W1
op2 DX5' DX5 DX6' DX6 -W8 C6C5 DX4' DX4 C6S5 C4W4 S4W4 C4W1 S4W3 S4W1 C4W3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17
SUM to R6 R4 Creg R2 R9 R1 Creg Sreg R3 R4 R5 R1
I
I<-------------- 1 = 3
I
I
BUS A src T3' T3 R7 R9 R1 R3 R1 R3 R4 R6 R4 R6 R7 R9 R7 R9
dst LA LA A1 A1 M1 M1 M1 M1 M1 M1 M1 M1 M1 M1 M1 M1
BUS B src D3' D3 a3 d4 Creg Sreg Sreg Creg Creg Sreg Sreg Creg Creg Sreg Sreg Creg
dst M2 M2 A2 A2 M2 M2 M2 M2 M2 M2 M2 M2 M2 M2 M2 M2
MPY 1 op1 X3 X3 W1 -W3 W1 -W3 W4 W6 W4 W6 W7 W9 W7 W9
op2 D3' D3 C3 S3 S3 C3 C3 S3 S3 C3 C3 S3 S3 C3
MPY 2 op1 S4W4 X3 X3 W1 -W3 W1 -W3 W4 W6 W4 W6 W7 W9 W7
op2 d6 D3' D3 C3 S3 S3 C3 C3 S3 S3 C3 C3 S3 S3
ADDER op1 0 0 T3' T3 W7 -W8 0 C3W1 0 S3W1 0 C3W4 0 S3W4 0 C3W7
op2 W7 W9 DX3' DX3 a3 d4 C3W1-S3W3 S3W1-C3W3 C3W4 S3W6 S3W4 C3W6 C3W7 S3W9
18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
SUM to R2 R7 R8 Creg Sreg R7 R9 R1 R3 R4 R6


I
I<---------- 1 = 2
I
I
BUS A src T2' T2 R7 R1 R3 R1 R3 R4 R6 R4 R6 R7 R9 R7 R9
dst LA LA A1 M1 M1 M1 M1 M1 M1 M1 M1 M1 M1 M1 M1
BUS B src D2' D2 a2 Creg Sreg Sreg Creg Creg Sreg Sreg Creg Creg Sreg Sreg Creg
dst M2 M2 A2 M2 M2 M2 M2 M2 M2 M2 M2 M2 M2 M2 M2
MPY 1 op1 X2 X2 W1 W2 W1 W2 W4 W5 W4 W5 W7 W8 W7 W8
op2 D2' D2 C2 S2 S2 C2 C2 S2 S2 C2 C2 S2 S2 C2
MPY 2 op1 W9 X2 X2 W1 W2 W1 W2 W4 W5 W4 W5 W7 W8 W7
op2 C3 D2' D2 C2 S2 S2 C2 C2 S2 S2 C2 C2 S2 S2
ADDER op1 0 S3W7 T2' T2 W7 0 C2W1 0 -S2W1 0 C2W4 0 -S2W4 0 C2W7
op2 S3W7 C3W9 DX2' DX2 a2 C2W1 S2W2 S2W1 C2W2 C2W4 S2W5 S2W4 C2W5 C2W7 S2W8
36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52
SUM to R7 R9 Creg Sreg R7 R1 R3 R4 R6


I
I<---------- 1 = 1
I
I
BUS A src T1' T1 R8 R1 R2 R1 R2 R4 R5 R4 R5 R7 R8 R7 R8
dst LA LA A1 M1 M1 M1 M1 M1 M1 M1 M1 M1 M1 M1 M1
BUS B src D1' D1 d2 Creg Sreg Sreg Creg Creg Sreg Sreg Creg Creg Sreg Sreg Creg
dst M2 M2 A2 M2 M2 M2 M2 M2 M2 M2 M2 M2 M2 M2 M2
MPY 1 op1 X1 X1 W1 W3 W1 W3 W4 W6 W4 W6 W7 W9 W7 W9
op2 D1' D1 C1 S1 S1 C1 C1 S1 S1 C1 C1 S1 S1 C1
MPY 2 op1 W8 X1 X1 W1 W3 W1 W3 W4 W6 W4 W6 W7 W9 W7 W9
op2 C2 D1' D1 C1 S1 S1 C1 C1 S1 S1 C1 C1 S1 S1 C1
ADDER op1 0 -S2W7 T1' T1 W9 0 C1W1 0 S1W1 0 C1W4 0 S1W4 0 C1W7 0 S1W7
op2 S2W7 C2W8 DX1' DX1 d2 C1W1 S1W3 S1W1 C1W3 C1W4 S1W6 S1W4 C1W6 C1W7 S1W9 S1W7 C1W9
53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
SUM to R7 R9 Creg Sreg R8 R1 R2 R4 R5 R7 R8

APPENDIX 2

DESCRIPTION OF THE

DKS CHIP IN RTL

```
type REGISTER : Bit<17:0>;
     DYNREG : Bit<17:0>;            /* Dynamic Register */
     JOINT : Bit<11:0>;
     ROM : Bit<17:0>;

var  R(9), CREG, SREG OF REGISTER;
     ANGLE(6) OF JOINT;
     LX, M1, M2, A1, A2, TXPORT, DXPORT, TXDLAY(2) OF DYNREG;
     TX(256), DX(256), CONS(5) OF ROM;
     MPY1, MPY2, ADDER OF Bit<17:0>;      /* C/L output */
     SINCOS OF Bit<7:0>;             /* Table address */
     SC, CTRL, SGN OF Bit<0:0>;      /* Control signal */

MACRO TRIG (FUNC, I)
Cycle N:    if FUNC = SIN then SC←0 else sc←1 endif,
            TXPORT←TX[SINCOS(ANGLE(I)<11:4>)],
            DXPORT←DX[SINCOS(ANGLE(I)<11:4>)];
Cycle N+1:  M1←LX, M2←DXPORT, TXDLAY(1)←TXPORT;
Cycle N+3:  A1←TXDLAY(2), A2←MPY2, CTRL←0;
Cycle N+4:  if FUNC = SIN then Ls←1 else Lc←1;
END TRIG

Begin
Cycle  (0): .TRIG(COS,5);
       (1): .TRIG(SIN,5);
       (2): .TRIG(COS,6);
       (3): .TRIG(SIN,6);
       (4): R(6)←ADDER;
       (5): M1←R(6), M2←CONS(0), R(4)←ADDER;        /* CONS(0)=d6 */
       (6): .TRIG(COS,4), M1←R(6), M2←ADDER;
       (7): .TRIG(SIN,4), A1←0, R(2)←ADDER;
       (8): A1←0, R(9)←ADDER;
       (9): M1←R(4), M2←CREG, R(1)←ADDER;
      (10): M1←R(4), M2←ADDER;
      (11): M1←R(4), M2←ADDER, A1←0;
      (12): M1←R(1), M2←CREG, A1←0, R(3)←ADDER;
      (13): M1←R(2), M2←SREG, A1←0, R(4)←ADDER;
      (14): M1←R(1), M2←SREG, A1←0, R(5)←ADDER;
      (15): M1←R(2), M2←CREG, A1←ADDER, SGN←1;
      (16): M1←R(4), M2←CONS(0), A1←0, R(1)←ADDER;
      (17): .TRIG(COS,3), M1←R(5), M2←CONS(0), A1←ADDER;
      (18): .TRIG(SIN,3), A1←0, R(2)←ADDER;
      (19): A1←0, R(7)←ADDER;
      (20): R(8)←ADDER;
      (21):
      (22): A1←R(7), A2←CONS(1);       /* CONS(1)=a3 */
      (23): A1←R(9), A2←CONS(2), R(7)←ADDER;      /* CONS(2)=d4 */
      (24): M1←R(1), M2←CREG, R(9)←ADDER;
      (25): M1←R(3), M2←SREG;
```

```
        (26): M1←R(1), M2←SREG, A1←0;
        (27): M1←R(3), M2←CREG, A1←ADDER, SGN←1;
        (28): M1←R(4), M2←CREG, A1←0, R1←ADDER;
        (29): M1←R(6), M2←SREG, A1←ADDER;
        (30): M1←R(4), M2←SREG, A1←0, R3←ADDER;
        (31): M1←R(6), M2←CREG, A1←ADDER;
        (32): M1←R(7), M2←CREG, A1←0, R4←ADDER;
        (33): M1←R(9), M2←SREG, A1←ADDER, SGN←1;
        (34): M1←R(7), M2←SREG, A1←0, R6←ADDER;
        (35): .TRIG(COS,2), M1←R(9), M2←CREG, A1←ADDER;
        (36): .TRIG(SIN,2), A1←0, R(7)←ADDER;
        (37): A1←ADDER, SGN←1;
        (38): R(9)←ADDER;
        (39):
        (40): A1←R(7), A2←CONS(3);        /* CONS(3)=a2 */
        (41): M1←R(1), M2←CREG, R(7)←ADDER;
        (42): M1←R(3), M2←SREG;
        (43): M1←R(1), M2←SREG, A1←0;
        (44): M1←R(3), M2←CREG, A1←ADDER, SGN←1;
        (45): M1←R(4), M2←CREG, A1←0, SGN←1, R1←ADDER;
        (46): M1←R(6), M2←SREG, A1←ADDER, SGN←1;
        (47): M1←R(4), M2←SREG, A1←0, R3←ADDER;
        (48): M1←R(6), M2←CREG, A1←ADDER, SGN←1;
        (49): M1←R(7), M2←CREG, A1←0, SGN←1, R4←ADDER;
        (50): M1←R(9), M2←SREG, A1←ADDER, SGN←1;
        (51): M1←R(7), M2←SREG, A1←0, R6←ADDER;
        (52): .TRIG(COS,1), M1←R(9), M2←CREG, A1←ADDER, SGN←1;
        (53): .TRIG(SIN,1), A1←0, SGN←1, R(7)←ADDER;
        (54): A1←ADDER, SGN←1;
        (55): R(9)←ADDER;
        (56):
        (57): A1←R(8), A2←CONS(4);        /* CONS(4)=d2 */
        (58): M1←R(1), M2←CREG, R(8)←ADDER;
        (59): M1←R(2), M2←SREG;
        (60): M1←R(1), M2←SREG, A1←0;
        (61): M1←R(2), M2←CREG, A1←ADDER, SGN←1;
        (62): M1←R(4), M2←CREG, A1←0, R1←ADDER;
        (63): M1←R(5), M2←SREG, A1←ADDER;
        (64): M1←R(4), M2←SREG, A1←0, R3←ADDER;
        (65): M1←R(5), M2←CREG, A1←ADDER, SGN←1;
        (66): M1←R(7), M2←CREG, A1←0, R4←ADDER;
        (67): M1←R(8), M2←SREG, A1←ADDER;
        (68): M1←R(7), M2←SREG, A1←0, R5←ADDER;
        (69): M1←R(9), M2←CREG, A1←ADDER, SGN←1;
        (70): A1←0, R(7)←ADDER;
        (71): A1←ADDER;
        (72): R(8)←ADDER
End.
```

APPENDIX 3

DKS MACHINE CODE

CLOCK CYCLE	ADRTAB Ev	v	SINCOS s/c	Et	ROM EM	adr	BUSAR T	adr	RSUM L	adr	MPY M1	M2	A zd	1 bs	A2/G A2	G	MINUS cO	SGN	CREG Lc	Tc	SREG Ls	Ts
0	1	5	1	0	0	X	0	X	0	X	X	0	X	0	X	0	X	X	0	0	0	0
1	1	5	0	1	0	X	0	X	0	X	1	1	X	0	X	0	X	X	0	0	0	0
2	1	6	1	1	0	X	0	X	0	X	1	1	X	0	X	0	X	X	0	0	0	0
3	1	6	0	1	0	X	0	X	0	X	1	1	1	0	1	0	0	X	0	0	0	0
4	X	0	X	1	0	X	0	X	1	6	1	1	1	0	1	0	0	X	0	0	0	0
5	X	0	X	0	1	0	1	6	1	4	2	1	1	0	1	0	0	X	0	0	0	0
6	1	4	1	0	0	X	1	6	0	X	2	1	1	0	1	1	0	X	1	0	0	0
7	1	4	0	1	0	X	0	X	1	2	1	1	2	0	1	0	1	0	0	0	0	0
8	X	0	X	1	0	X	0	X	1	9	1	1	2	0	1	0	1	0	0	0	0	0
9	X	0	X	0	0	X	1	4	1	1	2	1	1	0	1	0	0	X	0	1	0	0
10	X	0	X	0	0	X	1	4	0	X	2	1	1	0	1	1	0	X	1	0	0	0
11	X	0	X	0	0	X	1	4	0	X	2	1	2	0	1	1	1	0	0	0	1	0
12	X	0	X	0	0	X	1	1	1	3	2	1	2	0	1	0	1	0	0	1	0	0
13	X	0	X	0	0	X	1	2	1	4	2	1	2	0	1	0	1	0	0	0	0	1
14	X	0	X	0	0	X	1	1	1	5	2	1	2	0	1	0	1	0	0	0	0	1
15	X	0	X	0	0	X	1	2	0	X	2	1	0	1	1	0	1	1	0	1	0	0
16	X	0	X	0	1	0	1	4	1	1	2	1	2	0	1	0	1	0	0	0	0	0
17	1	3	1	0	1	0	1	5	0	X	2	1	0	1	1	0	1	0	0	0	0	0
18	1	3	0	1	0	X	0	X	1	2	1	1	2	0	1	0	1	0	0	0	0	0
19	X	0	X	1	0	X	0	X	1	7	1	1	2	0	1	0	1	0	0	0	0	0
20	X	0	X	0	0	X	0	X	1	8	X	0	1	0	1	0	0	X	0	0	0	0
21	X	0	X	0	0	X	0	X	0	X	X	0	1	0	1	0	0	X	1	0	0	0
22	X	0	X	0	1	1	1	7	0	X	X	0	0	2	2	0	1	0	0	0	1	0
23	X	0	X	0	1	2	1	9	1	7	X	0	0	2	2	0	1	0	0	0	0	0
24	X	0	X	0	0	X	1	1	1	9	2	1	X	0	X	0	X	X	0	1	0	0

CLOCK CYCLE	ADRTAB Ev	ADRTAB v	SINCOS s/c	SINCOS Et	ROM EM	ROM adr	BUSAR T	BUSAR adr	RSUM L	RSUM adr	MPY M1	MPY M2	A zd	A 1 bs	A2/G A2	A2/G G	MINUS cO	MINUS SGN	CREG Lc	CREG Tc	SREG Ls	SREG Ts
25	X	0	X	0	0	X	1	3	0	X	2	1	X	0	X	0	X	X	0	0	0	1
26	X	0	X	0	0	X	1	1	0	X	2	1	2	0	1	0	1	0	0	0	0	1
27	X	0	X	0	0	X	1	3	0	X	2	1	0	1	1	0	1	1	0	1	0	0
28	X	0	X	0	0	X	1	4	1	1	2	1	2	0	1	0	1	0	0	1	0	0
29	X	0	X	0	0	X	1	6	0	X	2	1	0	1	1	0	1	0	0	0	0	1
30	X	0	X	0	0	X	1	4	1	3	2	1	2	0	1	0	1	0	0	0	0	1
31	X	0	X	0	0	X	1	6	0	X	2	1	0	1	1	0	1	0	0	1	0	0
32	X	0	X	0	0	X	1	7	1	4	2	1	2	0	1	0	1	0	0	1	0	0
33	X	0	X	0	0	X	1	9	0	X	2	1	0	1	1	0	1	1	0	0	0	1
34	X	0	X	0	0	X	1	7	1	6	2	1	2	0	1	0	1	0	0	0	0	1
35	1	2	1	0	0	X	1	9	0	X	2	1	0	1	1	0	1	0	0	1	0	0
36	1	2	0	1	0	X	0	X	1	7	1	1	2	0	1	0	1	0	0	0	0	0
37	X	0	X	1	0	X	0	X	0	X	1	1	0	1	1	0	1	1	0	0	0	0
38	X	0	X	0	0	X	0	X	1	9	X	0	1	0	1	0	0	X	0	0	0	0
39	X	0	X	0	0	X	0	X	0	X	X	0	1	0	1	0	0	X	1	0	0	0
40	X	0	X	0	1	3	1	7	0	X	X	0	0	2	2	0	1	0	0	0	1	0
41	X	0	X	0	0	X	1	1	1	7	2	1	X	0	X	0	X	X	0	1	0	0
42	X	0	X	0	0	X	1	3	0	X	2	1	X	0	X	0	X	X	0	0	0	1
43	X	0	X	0	0	X	1	1	0	X	2	1	2	0	1	0	1	0	0	0	0	1
44	X	0	X	0	0	X	1	3	0	X	2	1	0	1	1	0	1	1	0	1	0	0
45	X	0	X	0	0	X	1	4	1	1	2	1	2	0	1	0	1	1	0	1	0	0
46	X	0	X	0	0	X	1	6	0	X	2	1	0	1	1	0	1	1	0	0	0	1
47	X	0	X	0	0	X	1	4	1	3	2	1	2	0	1	0	1	0	0	0	0	1
48	X	0	X	0	0	X	1	6	0	X	2	1	0	1	1	0	1	1	0	1	0	0
49	X	0	X	0	0	X	1	7	1	4	2	1	2	0	1	0	1	1	0	1	0	0
50	X	0	X	0	0	X	1	9	0	X	2	1	0	1	1	0	1	1	0	0	0	1

CLOCK CYCLE	ADR Ev	TAB v	SINCOS s/c	Et	ROM EM	adr	BUSAR T	adr	RSUM L	adr	MPY M1	M2	A zd	1 bs	A2/G A2	G	MINUS cO	SGN	CREG Lc	Tc	SREG Ls	Ts
51	X	0	X	0	0	X	1	7	1	6	2	1	2	0	1	0	1	0	0	0	0	1
52	1	1	1	0	0	X	1	9	0	X	2	1	0	1	1	0	1	1	0	1	0	0
53	1	1	0	1	0	X	0	X	1	7	1	1	2	0	1	0	1	1	0	0	0	0
54	X	0	X	1	0	X	0	X	0	X	1	1	0	1	1	0	1	1	0	0	0	0
55	X	0	X	0	0	X	0	X	1	9	X	0	1	0	1	0	0	X	0	0	0	0
56	X	0	X	0	0	X	0	X	0	X	X	0	1	0	1	0	0	X	1	0	0	0
57	X	0	X	0	1	4	1	8	0	X	X	0	0	2	2	0	1	0	0	0	1	0
58	X	0	X	0	0	X	1	1	1	8	2	1	X	0	X	0	X	X	0	1	0	0
59	X	0	X	0	0	X	1	2	0	X	2	1	X	0	X	0	X	X	0	0	0	1
60	X	0	X	0	0	X	1	1	0	X	2	1	2	0	1	0	1	0	0	0	0	1
61	X	0	X	0	0	X	1	2	0	X	2	1	0	1	1	0	1	1	0	1	0	0
62	X	0	X	0	0	X	1	4	1	1	2	1	2	0	1	0	1	0	0	1	0	0
63	X	0	X	0	0	X	1	5	0	X	2	1	0	1	1	0	1	0	0	0	0	1
64	X	0	X	0	0	X	1	4	1	2	2	1	2	0	1	0	1	0	0	0	0	1
65	X	0	X	0	0	X	1	5	0	X	2	1	0	1	1	0	1	1	0	1	0	0
66	X	0	X	0	0	X	1	7	1	4	2	1	2	0	1	0	1	0	0	1	0	0
67	X	0	X	0	0	X	1	8	0	X	2	1	0	1	1	0	1	0	0	0	0	1
68	X	0	X	0	0	X	1	7	1	5	2	1	2	0	1	0	1	0	0	0	0	1
69	X	0	X	0	0	X	1	8	0	X	2	1	0	1	1	0	1	1	0	1	0	0
70	X	0	X	0	0	X	0	X	1	7	X	0	2	0	1	0	1	1	0	0	0	0
71	X	0	X	0	0	X	0	X	0	X	X	0	0	1	1	0	1	0	0	0	0	0
72	X	0	X	0	0	X	0	X	1	8	X	0	X	0	X	0	X	X	0	0	0	0

APPENDIX 4

PROGRAM LISTING AND SIMULATION RESULT

```
   PROGRAM DKSVRF      74/175  OPT=1,ROUND= A/ S/ M/-D,-DS      FTN 5.1+587          02/27/85
DO=-LONG/-OT,ARG=-COMMON/-FIXED,CS= USER/-FIXED,DB=-TB/-SB/-SL/ ER/-ID/-PMD/-ST,PL=50000
FTN5.

      PROGRAM DKSVRF
C
C     THIS PROGRAM SIMULATES THE DATA FLOW (USING SYMBOLIC VALUES)
C     IN THE DKS CHIP TO VERIFY THE CONTROL SIGNAL SPECIFICATION
C     AND THE DESIGN AS A WHOLE AT THE BEHAVIORAL LEVEL.
C
      IMPLICIT INTEGER (A-Z)
      COMMON /CONTRL/CONTRL(11,2)
      COMMON /CFIELD/ADRTBL,SINCOS,CONS,BUSAR,RSUM,MPY,A1,A2,MINUS,C,S
      COMMON /MEM/ROM(0:4),THETA(6),UNDEF,CSFLAG,SIGN
      COMMON /REG/REG(9),CREG,SREG
      COMMON /LATCH/DYNREG(4,2),DELAY(4,2),TXPORT,DXPORT,LX
      COMMON /MISC/BUSA,BUSB,MPY1,MPY2,ADDER,MYPIPE(1,2)
C
      CHARACTER*2 ROM,THETA,UNDEF,CREG,SREG,CSFLAG,SIGN
      CHARACTER*4 TXPORT,DXPORT,LX,DELAY
      CHARACTER*100 DYNREG,REG,MPY1,MPY2,ADDER,BUSA,BUSB,MYPIPE
      EXTERNAL PHASE1,PHASE2
C
C     FOR INFORMATION ABOUT THE USAGE OF THE VARIABLES,
C     SEE COMMENTS IN SUBROUTINE PHASE1.
C
      DATA ROM/'D6','A3','D4','A2','D2'/
      DATA THETA/'X1','X2','X3','X4','X5','X6'/
      DATA ADRTBL,SINCOS,CONS,BUSAR,RSUM,MPY,A1/1,2,3,4,5,6,7/
      DATA A2,MINUS,C,S/8,9,10,11/
      DATA UNDEF/'E?'/
C
C     INITIALIZATION: SET THE CONTENTS OF ALL STORAGE ELEMENTS TO UNDEFINED
C
      DO 1 I=1,4
        DYNREG(I,1)=UNDEF
        DYNREG(I,2)=DELAY(I,1)=DELAY(I,2)=UNDEF(2:)
1     CONTINUE
      DO 2 I=1,9
        REG(I)=UNDEF(2:)
2     CONTINUE
      BUSA=BUSB=UNDEF
      CREG=SREG=UNDEF(2:)
      TXPORT=DXPORT=LX=UNDEF(2:)
      MPY1=MPY2=ADDER=UNDEF(2:)
      MYPIPE(1,1)=MYPIPE(1,2)=UNDEF(2:)
      CSFLAG=SIGN=UNDEF(2:)
      PRINT 3
3     FORMAT ('1')
C
C     THE BEHAVIOR OF THE CHIP IS SIMULATED THROUGH THE COMPLETE D.K.S.
C     CALCULATION PERIOD DRIVEN BY THE CONTROL SIGNAL TABLE. THE ENTIRE
C     PERIOD CONSISTS OF 77 2-PHASE, NON-OVERLAPPED SYSTEM CLOCK CYCLES.
C     DURING PHASE ONE, DATA FLOWS BETWEEN VARIOUS STORAGE ELEMENTS VIA
C     SYSTEM OR LOCAL BUSES. DURING PHASE TWO,  DATA  IS TRANSFERRED TO
C     PHASE-TWO LATCHES AND PROCESSED  BY COMBINATIONAL LOGIC SECTIONS.
C     THE ASSIGNED FUNCTION IS SIMULATED  BY  MANIPULATING THE SYMBOLIC
C     VALUES OF DATA. THE BUS IS ALSO PRECHARGED DURING PHASE TWO.
C
      ERR=0
      CYCLE=0
      OPEN (1,FILE='TABLE')
      READ (1,*) NUM,DBUG
      IF (NUM .GT. 76) NUM=76
100   IF (CYCLE .GT. NUM) THEN
        CLOSE (1)
        CALL DMPREG
        STOP
        ENDIF
        READ (1,*) ((CONTRL(FIELD,SUB), SUB=1,2), FIELD=ADRTBL,S)
        IF (DBUG .EQ. 1) THEN
          PRINT *, ' '
          PRINT *, '   CYCLE = ', CYCLE, '    CONTROL WORD = ', ((CONTRL
     +(FIELD,SUB), SUB=1,2), FIELD=ADRTBL,S)
        ENDIF
C
        CALL PHASE1 (ERR)
C
        IF (ERR .NE. 0) GOTO 200
```

```
PROGRAM DKSVRF      74/175  OPT=1,ROUND= A/ S/ M/-D,-DS      FTN 5.1+587          02/27/85

            IF (DBUG .EQ. 1) CALL DMPDYN
      C
            CALL PHASE2
      C
            IF (DBUG .EQ. 1) THEN
              CALL DMPDYN
              PRINT *, '  MPY1 = ',MPY1
              PRINT *, ' '
              PRINT *, '  ADDER = ',ADDER
              PRINT *, ' '
              CALL DMPREG
            ENDIF
      C
            IF ((CYCLE.EQ.9).OR.(CYCLE.EQ.24).OR.(CYCLE.EQ.40).OR.(CYCLE.EQ
           +.58)) CALL DMPREG
              CYCLE=CYCLE+1
            GOTO 100
      C
      C     ERROR HANDLING SECTION
      C
      200   PRINT *, '    *****  ERROR  *****'
            PRINT *, '    CYCLE = ', CYCLE, '          ERROR CODE = ', ERR
            PRINT *, '    CONTROL WORD = ', ((CONTRL(FIELD,SUB), SUB=1,2), FIEL
           +D=ADRTBL,5)
            PRINT *, ' '
            PRINT *, '    BUSA = ', BUSA
            PRINT *, '    BUSB = ', BUSB
            PRINT *, ' '
            CALL  DMPDYN
            CALL  DMPREG
            STOP
            END
```

```
SUBROUTINE DMPREG     74/175  OPT=1,ROUND= A/ S/ M/-D,-DS      FTN 5.1+587          02/27/85
DO=-LONG/-OT,ARG=-COMMON/-FIXED,CS= USER/-FIXED,DB=-TB/-SB/-SL/ ER/-ID/-PMD/-ST,PL=50000
FTN5.

            SUBROUTINE DMPREG
            COMMON /REG/REG(9)
            CHARACTER*100 REG
            PRINT *, '    *****  OUTPUT REGISTER CONTENT DUMP  *****'
            DO 1 I=1,9
              PRINT *, '    OUT REG ', I, ' = ', REG(I)
      1     CONTINUE
            PRINT *, ' '
            RETURN
            END
```

```
SUBROUTINE DMPDYN     74/175  OPT=1,ROUND= A/ S/ M/-D,-DS      FTN 5.1+587          02/27/85
DO=-LONG/-OT,ARG=-COMMON/-FIXED,CS= USER/-FIXED,DB=-TB/-SB/-SL/ ER/-ID/-PMD/-ST,PL=50000
FTN5.

            SUBROUTINE DMPDYN
            COMMON /LATCH/DYNREG(4,2)
            CHARACTER*100 DYNREG
            PRINT *, '    *****  DYNAMIC REGISTER CONTENT DUMP  *****'
            DO 1 I=1,4
              DO 1 J=1,2
                PRINT *, '    DYN REG ', I, ' PHASE ', J, ' = ', DYNREG(I,J)
      1     CONTINUE
            PRINT *, ' '
            RETURN
            END
```

```
SUBROUTINE PHASE1      74/175  OPT=1,ROUND= A/ S/ M/-D,-DS      FTN 5.1+587          02/27/85
DO=-LONG/-OT,ARG=-COMMON/-FIXED,CS= USER/-FIXED,DB=-TB/-SB/-SL/ ER/-ID/-PMD/-ST,PL=50000
FTN5,.

      SUBROUTINE PHASE1 (ERR)
      IMPLICIT INTEGER (A-Z)
      COMMON /CONTRL/CONTRL(11,2)
      COMMON /CFIELD/ADRTBL,SINCOS,CONS,BUSAR,RSUM,MPY,A1,A2,MINUS,C,S
      COMMON /MEM/ROM(0:4),THETA(6),UNDEF,CSFLAG,SIGN
      COMMON /REG/REG(9),CREG,SREG
      COMMON /LATCH/DYNREG(4,2),DELAY(4,2),TXPORT,DXPORT,LX
      COMMON /MISC/BUSA,BUSB,MPY1,MPY2,ADDER,MYPIPE(1,2)
C
      CHARACTER*2 ROM,THETA,UNDEF,CREG,SREG,CSFLAG,SIGN
      CHARACTER*4 TXPORT,DXPORT,LX,DELAY
      CHARACTER*100 DYNREG,REG,MPY1,MPY2,ADDER,BUSA,BUSB,MYPIPE
C
C     VARIABLE USAGE EXPLANATION:
C
C       CONTRL--CONTROL WORD OF THE CURRENT CYCLE
C               1ST INDEX IDENTIFIES 'FIELD' AS IN 'CFIELD' COMMON BLOCK
C              EACH FIELD HAS 2 SUBFIELDS IDENTIFIED BY THE 2ND INDEX
C       DYNREG(I,PHASE)--DYNAMIC REGISTERS CONTROLLED BY PHASE = 1 OR 2
C                        I=1: M1, MULTIPLIER OPERAND LATCH CONNECTED TO BUSA
C                        I=2: M2, MULTIPLIER OPERAND LATCH CONNECTED TO BUSB
C                        I=3: A1, ADDER OPERAND LATCH CONNECTED TO BUSA
C                        I=4: A2, ADDER OPERAND LATCH CONNECTED TO BUSB
C       DELAY(I,PHASE)--DELAY ELEMENTS CONTROLLED BY PHASE = 1 OR 2
C                      I=1: 1ST DELAY STAGE FOR TX
C                      I=2: 2ND DELAY STAGE FOR TX
C                      I=3: 1ST DELAY STAGE FOR SIGN CONTROL IN SIN CALCULATION
C                      I=4: 2ND DELAY STAGE FOR SIGN CONTROL IN SIN CALCULATION
C       BUSA(B)--THE 1ST CHARACTER IS A E/F FLAG FUNCTIONS SIMILAR TO CTRLAH
C
C
C     TASK 1: RULE CHECKING
C
      IF (CONTRL(BUSAR,1)+CONTRL(RSUM,1) .GT. 1) THEN
        IF (CONTRL(BUSAR,2) .EQ. CONTRL(RSUM,2)) ERR=1
C     *****  ERROR: TRY TO LOAD AND TRANSMIT SAME REG AT THE SAME TIME  *****
      ELSEIF (CONTRL(C,1)+CONTRL(C,2) .GT. 1) THEN
        ERR=2
C     *****  ERROR: TRY TO LOAD AND TRANSMIT CREG AT THE SAME TIME  *****
      ELSEIF (CONTRL(S,1)+CONTRL(S,2) .GT. 1) THEN
        ERR=3
C     *****  ERROR: TRY TO LOAD AND TRANSMIT SREG AT THE SAME TIME  *****
      ENDIF
      IF (ERR .NE. 0) RETURN
C
C     TASK 2: PROCESS LOAD BUS CASES
C
      IF (CONTRL(SINCOS,2) .EQ. 1) THEN
        BUSA='F'//TXPORT
        BUSB='F'//DXPORT
      ENDIF
C
      IF ((CONTRL(BUSAR,1) .EQ. 1) .AND. (CONTRL(BUSAR,2) .NE. 0)) THEN
        IF (BUSA(:1) .NE. 'E') THEN
          ERR=11
C     *****  REGISTER ATTEMP TO LOAD THE ALREADY LOADED BUS A  *****
          RETURN
        ELSE
          BUSA='F'//REG(CONTRL(BUSAR,2))
        ENDIF
      ENDIF
C
      IF (CONTRL(CONS,1) .EQ. 1) THEN
        IF (BUSB(:1) .NE. 'E') THEN
          ERR=12
C     *****  ROM ATTEMP TO LOAD THE ALREADY LOADED BUS B  *****
          RETURN
        ELSE
          BUSB='F'//ROM(CONTRL(CONS,2))
        ENDIF
      ENDIF
C
      IF (CONTRL(C,2) .EQ. 1) THEN
        IF (BUSB(:1) .NE. 'E') THEN
          ERR=13
```

```
SUBROUTINE PHASE1     74/175  OPT=1,ROUND= A/ S/ M/-D,-DS     FTN 5.1+587        02/27/85

C     *****  CREG ATTEMP TO LOAD THE ALREADY LOADED BUS B  *****
          RETURN
        ELSE
          BUSB='F'//CREG
        ENDIF
      ENDIF
C
      IF (CONTRL(S,2) .EQ. 1) THEN
        IF (BUSB(:1) .NE. 'E') THEN
          ERR=14
C     *****  SREG ATTEMP TO LOAD THE ALREADY LOADED BUS B  *****
          RETURN
        ELSE
          BUSB='F'//SREG
        ENDIF
      ENDIF
C
      IF (CONTRL(A2,2) .EQ. 1) THEN
        IF (BUSB(:1) .NE. 'E') THEN
          ERR=15
C     *****  ADDER OUTPUT ATTEMP TO LOAD THE ALREADY LOADED BUS B  *****
          RETURN
        ELSE
          BUSB='F'//ADDER
        ENDIF
      ENDIF
C
C     TASK 3: PROCESS THE LOADINGS OF REGISTERS OR CONTROLLED-LATCHES
C
      IF ((CONTRL(ADRTBL,1) .EQ. 0) .AND. (CONTRL(ADRTBL,2) .NE.0)) THEN
        PRINT *, '    *****  WARNING  *****'
        PRINT *, '    AT CYCLE ', CYCLE-1, ' ATTEMP TO LOAD THETA, INSTRU
     +CTION IGNORED.'
      ENDIF
C
      IF ((CONTRL(RSUM,1) .EQ. 1) .AND. (CONTRL(RSUM,2) .NE. 0)) THEN
        REG(CONTRL(RSUM,2))=ADDER
      ENDIF
C
      IF (CONTRL(MPY,1) .GT. 2) THEN
        ERR=21
C     *****  ATTEMP TO LOAD MULTIPLIER INPUT LATCH MORE THAN ONCE  *****
        RETURN
      ELSE
        IF (CONTRL(MPY,1) .EQ. 1) THEN
          DYNREG(1,1)=LX
        ELSEIF (CONTRL(MPY,1) .EQ. 2) THEN
          IF (BUSA(:1) .EQ. 'O') THEN
            ERR=22
            RETURN
          ELSE
            DYNREG(1,1)=BUSA(2:)
            BUSA(:1)='O'
          ENDIF
        ENDIF
      ENDIF
C
      IF (CONTRL(MPY,2) .EQ. 1) THEN
        IF (BUSB(:1) .EQ. 'O') THEN
          ERR=23
C     *****  ATTEMP TO LOAD MULTIPLIER INPUT LATCH MORE THAN ONCE  *****
          RETURN
        ELSE
          DYNREG(2,1)=BUSB(2:)
          BUSB(:1)='O'
        ENDIF
      ENDIF
C
      IF ((CONTRL(A1,1) .NE. 0) .AND. (CONTRL(A1,2) .NE. 0)) THEN
        ERR=24
C     *****  ATTEMP TO LOAD A1 MORE THAN ONCE  *****
        RETURN
      ELSEIF ((CONTRL(A1,1) .GT. 2) .OR. (CONTRL(A1,2) .GT. 2)) THEN
        ERR=25
C     *****  ATTEMP TO LOAD A1 MORE THAN ONCE  *****
        RETURN
      ELSEIF (CONTRL(A1,1) .EQ. 2) THEN
```

```
SUBROUTINE PHASE1      74/175  OPT=1,ROUND= A/ S/ M/-D,-DS      FTN 5.1+587          02/27/85

            DYNREG(3,1)='O'
          ELSEIF (CONTRL(A1,1) .EQ. 1) THEN
            DYNREG(3,1)=DELAY(2,2)
          ELSEIF (CONTRL(A1,2) .EQ. 1) THEN
            DYNREG(3,1)=ADDER
          ELSEIF (CONTRL(A1,2) .EQ. 2) THEN
            IF (BUSA(:1) .EQ. 'O') THEN
              ERR=26
              RETURN
            ELSE
              DYNREG(3,1)=BUSA(2:)
              BUSA(:1)='O'
            ENDIF
          ENDIF
C
          IF (CONTRL(A2,1) .GT. 2) THEN
            ERR=27
C     *****  ATTEMP TO LOAD A2 LATCH TWICE  *****
            RETURN
          ELSEIF (CONTRL(A2,1) .EQ. 1) THEN
            DYNREG(4,1)=MPY2
          ELSEIF (CONTRL(A2,1) .EQ. 2) THEN
            IF (BUSB(:1) .EQ. 'O') THEN
              ERR=28
              RETURN
            ELSE
              DYNREG(4,1)=BUSB(2:)
              BUSB(:1)='O'
            ENDIF
          ENDIF
C
          IF (CONTRL(SINCOS,2) .EQ. 1) THEN
            IF (BUSA(:1) .EQ. 'O') THEN
              ERR=29
              RETURN
            ELSE
              DELAY(1,1)=BUSA(2:)
              BUSA(:1)='O'
            ENDIF
          ENDIF
C
          IF (CONTRL(C,1) .EQ. 1) CREG=ADDER
          IF (CONTRL(S,1) .EQ. 1) SREG=ADDER
C
C         TASK 4: TRANSFER DATA TO OTHER PHASE-1 LATCHES FROM PHASE-2 LATCHES
C
          DELAY(2,1)=DELAY(1,2)
          DELAY(3,1)=CSFLAG
          DELAY(4,1)=DELAY(3,2)
          SIGN=DELAY(4,2)
          MYPIPE(1,1)=MPY1
C
          RETURN
          END
```

```
SUBROUTINE PHASE2      74/175  OPT=1,ROUND= A/ S/ M/-D,-DS      FTN 5.1+587          02/27/85
DO=-LONG/-OT,ARG=-COMMON/-FIXED,CS= USER/-FIXED,DB=-TB/-SB/-SL/ ER/-ID/-PMD/-ST,PL=50000
FTN5.

      SUBROUTINE PHASE2
      IMPLICIT INTEGER (A-Z)
      COMMON /CONTRL/CONTRL(11,2)
      COMMON /CFIELD/ADRTBL,SINCOS,CONS,BUSAR,RSUM,MPY,A1,A2,MINUS,C,S
      COMMON /MEM/ROM(0:4),THETA(6),UNDEF,CSFLAG,SIGN
      COMMON /REG/REG(9),CREG,SREG
      COMMON /LATCH/DYNREG(4,2),DELAY(4,2),TXPORT,DXPORT,LX
      COMMON /MISC/BUSA,BUSB,MPY1,MPY2,ADDER,MYPIPE(1,2)
C
      CHARACTER*2 ROM,THETA,UNDEF,CREG,SREG,CSFLAG,SIGN
      CHARACTER*4 TXPORT,DXPORT,LX,DELAY
      CHARACTER*100 DYNREG,REG,MPY1,MPY2,ADDER,BUSA,BUSB,MYPIPE
      CHARACTER*100 OP2
C
C     TASK 1: TRANSFER DATA FROM PHASE-1 LATCHES TO PHASE-2 LATCHES
C
      DO 1 I=1,4
        DYNREG(I,2)=DYNREG(I,1)
        DELAY(I,2)=DELAY(I,1)
1     CONTINUE
      MYPIPE(1,2)=MYPIPE(1,1)
C
C     TASK 2: PRECHARGE BUS AND UNDEFINE PHASE-1 CONTROLLED LATCHES
C
      BUSA=BUSB=UNDEF
      DO 2 I=1,4
        DYNREG(I,1)=UNDEF(2:)
2     CONTINUE
      DELAY(1,1)=UNDEF(2:)
C
C     TASK 3: SIN/COS SIMULATION
C
      IF (CONTRL(ADRTBL,1) .EQ. 1) THEN
        CSFLAG='1'
        LX=THETA(CONTRL(ADRTBL,2))
        TXPORT='T'//THETA(CONTRL(ADRTBL,2))(2:2)
        DXPORT='D'//THETA(CONTRL(ADRTBL,2))
        IF (CONTRL(SINCOS,1) .EQ. 1) THEN
          TXPORT(3:3)=''''
          DXPORT(4:4)=''''
        ENDIF
      ELSE
        LX=UNDEF(2:)
        CSFLAG=UNDEF(2:)
        TXPORT=DXPORT=UNDEF(2:)
      ENDIF
C
C     TASK 4: SIMULATE DATA FLOWING THROUGH COMBINATIONAL LOGIC SECTIONS
C
C       TASK 4.1: SIMULATE THE 1ST STAGE OF MULTIPLIER PIPE
C
      IF ((DYNREG(1,2)(:1).EQ.'X') .AND. (DYNREG(2,2)(:2).EQ.'DX')) THEN
        MPY1=DYNREG(2,2)
      ELSE
        CALL ACTLEN(M1,FLAG,DYNREG(1,2))
        SGN=1
        IF (DYNREG(1,2)(:1) .EQ. '-') SGN=-SGN
        IF (FLAG .EQ. 0) THEN
          IF (SGN .GT. 0) THEN
            MPY1=DYNREG(2,2)(:2)//DYNREG(1,2)
          ELSE
            MPY1='-'//DYNREG(2,2)(:2)//DYNREG(1,2)(2:)
          ENDIF
        ELSE
          IF (SGN .GT. 0) THEN
            MPY1=DYNREG(2,2)(:2)//'('//DYNREG(1,2)(:M1)//')'
          ELSE
            MPY1='-'//DYNREG(2,2)(:2)//DYNREG(1,2)(2:)
          ENDIF
        ENDIF
      ENDIF
C
C       TASK 4.2: SIMULATE THE 2ND STAGE OF MULTIPLIER PIPE
C
      MPY2=MYPIPE(1,2)
```

```
SUBROUTINE PHASE2     74/175  OPT=1,ROUND= A/ S/ M/-D,-DS     FTN 5.1+587        02/27/85

C
C       TASK 4.3: SIMULATE ADDER FUNCTION
C
      IF ((DYNREG(3,2)(:1).EQ.'T') .AND. (DYNREG(4,2)(:2).EQ.'DX')) THEN
        IF (DYNREG(3,2)(3:3) .EQ. '''') THEN
          ADDER='C'//DYNREG(3,2)(2:2)
        ELSE
          ADDER='S'//DYNREG(3,2)(2:2)
        ENDIF
        IF ((SIGN(:1) .NE. '1') .OR. (CONTRL(MINUS,1) .NE. 0)) THEN
          PRINT *,' '
          PRINT *,' ,***   ADDER CONTROL ERROR IN SINE SIMULATION  ***'
          PRINT *,' '
        ENDIF
      ELSE
        CALL ACTLEN(L2,FLAG,DYNREG(4,2))
        SGN=1
        IF (DYNREG(4,2)(:1) .EQ. '-') THEN
          SGN=-SGN
          IF (FLAG .EQ. 2) THEN
            OP2=DYNREG(4,2)(3:L2-1)
            L2=L2-3
            FLAG=1
          ELSE
            OP2=DYNREG(4,2)(2:)
            L2=L2-1
          ENDIF
        ELSE
          OP2=DYNREG(4,2)
        ENDIF
        IF (CONTRL(MINUS,1)+CONTRL(MINUS,2) .EQ. 2) SGN=-SGN
        IF (DYNREG(3,2)(:1) .EQ. 'O') THEN
          IF (SGN .GT. 0) THEN
            ADDER=OP2
          ELSEIF (FLAG .EQ. 0) THEN
            ADDER='-'//OP2
          ELSE
            ADDER='-('//OP2(:L2)//')'
          ENDIF
        ELSE
          CALL ACTLEN(L1,FLAG1,DYNREG(3,2))
          IF (DYNREG(3,2)(:1) .NE. '-') THEN
            IF (SGN .GT. 0) THEN
              ADDER=DYNREG(3,2)(:L1)//'+'//OP2(:L2)
            ELSEIF (FLAG .EQ. 0) THEN
              ADDER=DYNREG(3,2)(:L1)//'-'//OP2(:L2)
            ELSE
              ADDER=DYNREG(3,2)(:L1)//'-('//OP2(:L2)//')'
            ENDIF
          ELSE
            SGN=-SGN
            IF (FLAG1 .EQ. 0) THEN
              IF (SGN .GT. 0) THEN
                ADDER='-('//DYNREG(3,2)(2:L1)//'+'//OP2(:L2)//')'
              ELSEIF (FLAG .EQ. 0) THEN
                ADDER='-('//DYNREG(3,2)(2:L1)//'-'//OP2(:L2)//')'
              ELSE
                ADDER='-('//DYNREG(3,2)(2:L1)//'-('//OP2(:L2)//'))'
              ENDIF
            ELSE
              IF (SGN .GT. 0) THEN
                ADDER=DYNREG(3,2)(:L1-1)//'+'//OP2(:L2)//')'
              ELSEIF (FLAG .EQ. 0) THEN
                ADDER=DYNREG(3,2)(:L1-1)//'-'//OP2(:L2)//')'
              ELSE
                ADDER=DYNREG(3,2)(:L1-1)//'-('//OP2(:L2)//'))'
              ENDIF
            ENDIF
          ENDIF
        ENDIF
      ENDIF
      RETURN
      END
```

```
SUBROUTINE ACTLEN      74/175  OPT=1,ROUND=A/S/M/-D,-DS      FTN 5.1+587          02/27/85
DO=-LONG/-OT,ARG=-COMMON/-FIXED,CS=USER/-FIXED,DB=-TB/-SB/-SL/ER/-ID/-PMD/-ST,PL=50000
FTN5.

                SUBROUTINE ACTLEN (I,FLAG,STRING)
                INTEGER FLAG
                CHARACTER*100 STRING
                I=2
                J=0
                IF (STRING(I:I) .EQ. '(') THEN
                  FLAG=2
                ELSE
                  FLAG=0
                ENDIF
          1     IF (STRING(I:I) .EQ. ' ') GOTO 2
                  IF (FLAG .EQ. 2) GOTO 3
                  IF (STRING(I:I) .EQ. '(') THEN
                    J=J+1
                  ELSEIF (STRING(I:I) .EQ. ')') THEN
                    J=J-1
                  ELSEIF ((STRING(I:I) .EQ. '+') .OR. (STRING(I:I) .EQ. '-')) THEN
                    IF ((FLAG .EQ. 0) .AND. (J .EQ. 0)) FLAG=1
                  ENDIF
          3       I=I+1
                IF (I .LT. 100) GOTO 1
          2     I=I-1
                RETURN
                END
```

```
*****  OUTPUT REGISTER CONTENT DUMP  *****
OUT REG 1 = C6C5
OUT REG 2 = S6
OUT REG 3 = ?
OUT REG 4 = S5
OUT REG 5 = ?
OUT REG 6 = C5
OUT REG 7 = ?
OUT REG 8 = ?
OUT REG 9 = D6C5

*****  OUTPUT REGISTER CONTENT DUMP  *****
OUT REG 1 = C4C6C5-S4S6
OUT REG 2 = S4C6C5+C4S6
OUT REG 3 = C6S5
OUT REG 4 = C4S5
OUT REG 5 = S4S5
OUT REG 6 = C5
OUT REG 7 = D6C4S5+A3
OUT REG 8 = D6S4S5
OUT REG 9 = D6C5+D4

*****  OUTPUT REGISTER CONTENT DUMP  *****
OUT REG 1 = C3(C4C6C5-S4S6)-S3C6S5
OUT REG 2 = S4C6C5+C4S6
OUT REG 3 = S3(C4C6C5-S4S6)+C3C6S5
OUT REG 4 = C3C4S5+S3C5
OUT REG 5 = S4S5
OUT REG 6 = S3C4S5-C3C5
OUT REG 7 = C3(D6C4S5+A3)+S3(D6C5+D4)
OUT REG 8 = D6S4S5
OUT REG 9 = S3(D6C4S5+A3)-C3(D6C5+D4)

*****  OUTPUT REGISTER CONTENT DUMP  *****
OUT REG 1 = C2(C3(C4C6C5-S4S6)-S3C6S5)-S2(S3(C4C6C5-S4S6)+C3C6S5)
OUT REG 2 = S4C6C5+C4S6
OUT REG 3 = -(S2(C3(C4C6C5-S4S6)-S3C6S5)+C2(S3(C4C6C5-S4S6)+C3C6S5))
OUT REG 4 = C2(C3C4S5+S3C5)-S2(S3C4S5-C3C5)
OUT REG 5 = S4S5
OUT REG 6 = -(S2(C3C4S5+S3C5)+C2(S3C4S5-C3C5))
OUT REG 7 = C2(C3(D6C4S5+A3)+S3(D6C5+D4)+A2)-S2(S3(D6C4S5+A3)-C3(D6C5+D4))
OUT REG 8 = D6S4S5+D2
OUT REG 9 = -(S2(C3(D6C4S5+A3)+S3(D6C5+D4)+A2)+C2(S3(D6C4S5+A3)-C3(D6C5+D4)))

*****  OUTPUT REGISTER CONTENT DUMP  *****
OUT REG 1 = C1(C2(C3(C4C6C5-S4S6)-S3C6S5)-S2(S3(C4C6C5-S4S6)+C3C6S5))-S1(S4C6C5+C4S6)
OUT REG 2 = S1(C2(C3(C4C6C5-S4S6)-S3C6S5)-S2(S3(C4C6C5-S4S6)+C3C6S5))+C1(S4C6C5+C4S6)
OUT REG 3 = -(S2(C3(C4C6C5-S4S6)-S3C6S5)+C2(S3(C4C6C5-S4S6)+C3C6S5))
OUT REG 4 = C1(C2(C3C4S5+S3C5)-S2(S3C4S5-C3C5))-S1S4S5
OUT REG 5 = S1(C2(C3C4S5+S3C5)-S2(S3C4S5-C3C5))+C1S4S5
OUT REG 6 = -(S2(C3C4S5+S3C5)+C2(S3C4S5-C3C5))
OUT REG 7 = C1(C2(C3(D6C4S5+A3)+S3(D6C5+D4)+A2)-S2(S3(D6C4S5+A3)-C3(D6C5+D4)))-S1(D6S4S5+D2)
OUT REG 8 = -(S1(C2(C3(D6C4S5+A3)+S3(D6C5+D4)+A2)-S2(S3(D6C4S5+A3)-C3(D6C5+D4)))-C1(D6S4S5+D2))
OUT REG 9 = -(S2(C3(D6C4S5+A3)+S3(D6C5+D4)+A2)+C2(S3(D6C4S5+A3)-C3(D6C5+D4)))
```

BIBLIOGRAPHY

BIBLIOGRAPHY

1. J. Denavit, and R. S. Hartenberg, "A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices," *Journal of Applied Mechanics*, 1955, pp. 215-221.

2. E. Ribble and K. Olson, "Skeletal Motion Processor for High Speed Robotics and Graphic Computations," *Proceedings of 1984 International Conference on Robotics*, IEEE Computer Society Press, 1984, pp. 230-238.

3. *Introduction to VLSI Systems*, C. Mead and L. Conway, Readings, MA: Addison-Wesley, 1978, Chapter 3-5.

4. *Robot Manipulators: Mathematics, Programming and Control*, R. P. Paul, Cambridge, MA: MIT Press, 1981, Chapter 1,2.

5. G. C. S. Lee, "Robot Arm Kinematics" in *Tutorial on Robotics*, G. C. S. Lee, R. C. Gonzalez, and K. S. Fu, IEEE Computer Society Press, 1983, pp. 47-65.

6. *VLSI System Design, When and How to Design Very-Large-Scale Integrated Circuits*, S. Muroga, Wiley, 1982, p. 42, Chapter 7-9.

7. J. A. Darringer, et al., "LSS: A System for Production Logic Synthesis," *IBM Journal of Research and Development*, Vol. 28, No. 5, September 1984, pp. 537-545.

8. W. H. Elder, P. P. Zenewicz and R. R. Alvarodiaz, "An Interactive System for VLSI Chip Physical Design," *IBM Journal of Research and Development*, Vol. 28, No. 5, September 1984, pp. 524-536.

9. F. J. Hill, Z. Navabi, et al., "Hardware Compilation from an RTL to a Storage Logic Array Target," *IEEE Trans. on Computer-Aided Design*, Vol CAD-3, No. 3, July 1984, pp. 208-217.

10. S. C. Johnson, "VLSI Circuit Design Reaches the Level of Architecture Description," *Electronics*, May 3, 1984, pp. 121-128.

11. P. Wallick, "On the Horizon: Fast Chip Quickly," IEEE *Spectrum*, Vol. 21, No. 3, March 1984, pp. 28-34.

12. W. M. vanCleemput and H. Ofek, "Design Automation for Digital Systems," IEEE *Computer*, Vol. 17, No. 10, October 1984, pp. 114-122.

13. C. U. Smith and J. A. Dallen, "A Comparison of Design Strategies for Software and for VLSI," *IEEE 1984 Workshop on the Engineering of VLSI and Software*, IEEE Computer Society, 1984, pp. 160-165.

14. D. H. Barbe, "VHSIC Systems and Technology," IEEE *Computer*, Vol. 14, No. 2, February 1981, pp. 13-22.

15. F. Guterl, P. Wallich, and M. A. Fischetti, "In Pursuit of the One-Month Chip," IEEE *Spectrum*, Vol. 21, No. 9, September 1984, pp. 28-49.

16. H. H. Baller, "VLSI/Software Engineering Design Methodology," *IEEE 1984 Workshop on the Engineering of VLSI and Software*, IEEE Computer Society Press, 1984, pp. 3-5.

17. D. D. Gajski and R. H. Kuhn, "New VLSI Tools," IEEE *Computer*, Vol. 16, No. 12, December 1983, pp. 11-14.

18. C. V. Ramamoorthy, et al., "Software Engineering: Problems and Perspectives," IEEE *Computer*, Vol. 17, No. 10, October 1984, pp. 191-209.

19. *Industrial Robots Computer Interfacing and Control*, W. E. Snyder, Englewood Cliffs, NJ: Prentice-Hall, 1985, pp. 24-28, 119.

20. L. D. Harmon, "Automated Tactile Sensing," *The International Journal of Robotics Research*, vol. 1, no. 2, Summer, 1982.

21. J. F. Jarvis, "Robotics," IEEE *Computer*, Vol. 17, No. 10, October 1984, pp.283-292.

22. J. J. Uicker, J. Denavit, and R. S. Hartenberg, "An iterative Method for the Displacement Analysis of Spatial Mechanism," *Transactions of ASME, Journal of Applied Mechanics*, vol. 31, Series E, 1964, pp. 309-314.

23. *ElectronicsWeek*, November 5, 1984, p. 88.

24. *ElectronicsWeek*, July 12, 1984, p. 46.

25. M. A. Fischetti, "Solid State" in "Technology '85," IEEE *Spectrum*, Vol. 22, No. 1, January 1985, pp. 60-64.

26. C. F. Ruoff, "Fast Trig Functions for Robot Control," in *Robotics Age in the Beginning*, ed. by C. T. Helmers, Hasbrouck Heights, NY: Hayden Book, 1983, pp. 73-79.

27. J. E. Valder, "The CORDIC Trigonometric Computing Technique," *IEEE Trans. on Electronics and Computers*, Vol. EC-9, September 1960, pp. 227-231.

28. *Computer Arithmetic*, K. Hwang, John Wiley and Son, 1979, pp. 368-373.

29. S. Muroga, op. cit., pp. 313-316.

30. M. J. Foster and H. T. Kung, "The Design of Special-Purpose VLSI Chip," IEEE *Computer*, Vol. 13, No. 1, January 1980, pp. 26-40.

31. P. C. Treleaven, "VLSI Processor Architecture," IEEE *Computer*, Vol. 15, No. 6, June 1982, pp.33-45.

32. *Computer Arithmetic and Parallel Processing*, K. Hwang and F. A. Briggs, New York, McGraw Hill, 1984, pp. 788-803.

33. W. C. Hsu and M. A. Shanblatt, " Evaluation of a Single VLSI Chip Algorithm for Triangulating Large Band Form Matrices," Tech. Report No. MSU-ENGR 82-015, Michigan State University, E. Lansing, Michigan (August 1982).

34. *The Design and Description of Computer Architecture*, S. Dasgupta, Wiley, 1984, Chapter 1.

35. K. Hwang, op. cit., Chapter 3-6.

36. C. Mead and L. Conway, op. cit., p. 26.

37. C. Mead and L. Conway, op. cit., p. 231.

38. D. J. McGreivy, "Interconnects/Gates in VLSI Technologies," in *VLSI Through the 80's and Beyond*, ed. by D. J. McGreivy and K. A. Picker, IEEE Computer Society Press, 1982.

39. C. Mead and L. Conway, op. cit., p. 155.

40. L. I. Maissel and H. Ofek, "Hardware Design and Description Language in IBM," *IBM Journal of Research and Development*, Vol. 28, No. 5, September 1984, pp. 557-563.

41. M. Shahdad, et al., "VHSIC Hardware Description Language," IEEE *Computer*, Vol. 18, No. 2, February 1985, pp. 94-103.

42. *Structured Computer Organization*, A. S. Tanenbaum, Englewood Cliffs, NJ: Prentice-Hall, 1984, pp. 149-156.

43. R. Beresford, "Advances in Customization Free VLSI System Designers," *Electronics*, February 10, 1983, pp. 134-145.

44. B. R. Bourbon, "ICs Tailored to Applications Gain Ground," *ElectronicsWeek*, September 3, 1984, pp. 117-121.

45. D. L. Wollesen, "CMOS LSI - The Computer Component Process of the 80's," IEEE *Computer*, Vol. 13, No. 2, February 1980, pp. 59-67.

46. S. Muroga, op. cit. pp. 163-164.

47. D. J. McGreivy, "VLSI Chip Trend - Size, Complexity, Cost," in *VLSI Through the 80's and Beyond*, ed. by D. J. McGreivy and K. A. Picker, IEEE Computer Society Press, 1982.

48. B. C. Cole, "Memories Dominate ISSCC," *ElectronicsWeek*, February 11, 1985, pp. 51-60.

49. R. L. Donze and G. Sporzynski, "Masterimage Approach to VLSI Design," IEEE *Computer*, Vol. 16, No. 12, December 1983, pp. 18-25.

50. S. Zollo, "CMOS Array Break 1NS," *ElectronicsWeek*, January 21, 1985, pp. 55-56.

51. K. Kokkonem and R. Pashley, "Modular Approach to CMOS Technology Tailors Process to Application," *Electronics*, May 3, 1984, pp. 129-133.

52. L. Waller, "IC Houses are Fruitful in Multipliers," *Electronics*, July 14, 1983, pp. 155-157.

53. P. T. Greiling, "High Speed Digital IC Performance Outlook," Seminar, Michigan State University, January 31, 1985.

54. J. E. Dykes, "Bridging the Gap," *IEEE Proceedings of the 1984 Custom Integrated Circuits Conference*, Rochester, NY, 1984, pp. 5-8.