An Integrated Circuit for ECG Processing

A Thesis

Submitted to the College of

Graduate Studies and Research

in Partial Fulfilment of the Requirements

for the Degree of

Master of Science

in the Department of Electrical Engineering

University of Saskatchewan

by

Kirby Goulet

Saskatoon, Saskatchewan

August, 1987

Copyright

Copyright

Abstract

UNIVERSITY of SASKATCHEWAN

Electrical Engineering Abstract 87A284

An Integrated Circuit for ECG Processing

Student: Kirby Goulet
Supervisor: R.J. Bolton
M.Sc. Thesis Presented to the College of Graduate Studies and Reasearch

August 1987

Abstract

This thesis presents the design of an IC for use in the analysis of electrocardiograms (ECG's). This IC was developed because the pattern recognition technique used in the analysis requires a relatively large amount of computation, and custom hardware designed specifically for these calculations is necessary to achieve the desired ECG analysis speed.

In this report, the morphology recognition algorithm, which is based upon the Bhattacharyya distance measure, is explained and then modified into a form suitable for implementation in an IC. The major components of the IC are identified as a square-root extractor, a multiplier, a register file, and a system controller. A circuit description and an IC layout is developed for each of these components. The circuits are developed using two circuit simulation computer programs, SPICE and RNL, whose accuracy is checked by comparing their simulation results to measurements taken from a fabricated circuit. The NETLIST description of the complete IC design is checked through RNL simulation.

The resulting design has a computational rate which is 10 to 25 times faster than a design based upon a general purpose microprocessor performing identical computations.

Acknowledgements

The author wishes to thank Professor R.J.Bolton for his assistance in completing this work.

Fabrication of integrated circuits was made possible by Northern Telecom Electronics and the Canadian Microelectronics Corporation.

Financial assistance was provided by the Natural Science and Engineering Research Council, and their support is gratefully acknowledged.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

This thesis investigates one application of integrated circuit (IC) technology to a problem in the computer-aided analysis of electrocardiograms (ECG's). The thesis objective is to design an IC which will improve the performance of a computer ECG analysis system.

1.1. Background to ECG Processing

The purpose of computer analysis of ECG's is to assist medical personnel in the detection of heart disease. The computer does this by analyzing the shape of the ECG waveform; a normal healthy heart produces a regular and easily identifiable ECG waveform, but heart disease can cause either long term changes in the shape of the ECG waveform or occasional abnormal heartbeats which appear in the ECG as a wave shape notably different from the wave shape produced by a normal heartbeat.

These abnormal heartbeats are commonly called arrhythmias or dysrhythmias, and in the 1960's, experience showed that in an unhealthy heart, a type of arrhythmia called ventricular depolarization could trigger the potentially fatal ventricular fibrillation (a state of the heart where the muscles of the ventricles twitch in a feeble and irregular manner which produces no useful pumping action) [1]. As a result, coronary care units were set up to monitor a patient's ECG for abnormal heartbeats, and if abnormal heartbeats occurred, drugs were administered to suppress them. These specialized coronary care units were successful in reducing the in-hospital mortality of patients suffering myocardial infarctions (a blockage of a coronary artery) [2].

The key to the success of these coronary care units was the monitoring of the ECG signals for arrhythmias. To monitor the ECG signal, nurses or other trained personnel watched the ECG signals for abnormal heartbeats. However, this task was tedious and prone to errors, and computers were seen as a way of reducing these problems. Since the health of the patient could be affected by the performance of the computer ECG analysis system, it was important that the computer program accurately and reliably report ECG abnormalities. Developing a computer program which performed ECG analysis as well as a human proved to be difficult. As a demonstration of this, a comparison of the ability to diagnose heart disease through ECG's was made between a well-known ECG analysis program, the U.S. Veteran's Administration ECG analysis system, and hospital clinicians [3]. The results showed that clinicians did superior analysis of ECG's with a correct diagnosis of 59% versus the computer program's correct diagnosis rate of 50%.

These rates are low because there were many different heart diseases to consider. However, when classifying heart beats into only two classes, normal and abnormal, computer programs have been found to be more successful. In the detection of a class of arrhythmias called premature ventricular contractions (PVC's), computer programs have an accuracy of 95% [4], and while the use of automated ECG monitoring in the coronary care unit has not been proven to reduce patient mortality, there is agreement that automated arrhythmia analysis systems are more accurate than ECG monitoring by human surveillance [4, 1].

One disadvantage of the automated arrhythmia analysis programs, though, is the generation of false alarms. Due to electrical noise or movement of the patient connected to the ECG analysis system, the automated arrhythmia analysis system may indicate an otherwise normal heartbeat as an abnormal heartbeat. Such indications are called false positives. The false positive rate is usually one per hour per patient [4], which is not a problem unless several patients are being monitored, in which case, the high number of false alarms can demoralize the nursing staff [4].

R.J. Bolton has proposed an ECG analysis algorithm [5] which is not presently used in current ECG analysis programs. The proposed analysis algorithm is based upon the Hilbert Transform. This algorithm has been demonstrated to be useful in the detection of many ECG abnormalities, but one drawback of this method is the large number of computations required to perform the analysis. Over 7600 integer multiplications and 8600 square-root extractions must be performed every second to analyze ECG data from one lead, and if a microprocessor based ECG analysis system is to analyze several leads in real-time, hardware designed to speed up the ECG analysis calculations is required.

The hardware to do these calculations could be developed in a variety of ways. One way would be to add a second (and possibly a third) microprocessor to the ECG analysis system. Another way would be to combine currently available IC's which provide functions such as multiplication, data storage, and addition into a circuit that carries out the intended calculations. Yet another option which has recently become available to the electronics circuit designer would be to design an IC specifically for morphology recognition. In order to learn about the capabilities of IC technology in biomedical signal processing, the last approach is taken in this thesis.

1.2. Integrated Circuit Design Approaches

The design of an IC involves the generation of a layout which is used to pattern the designed circuit onto the surface of a silicon chip. The generation of this layout may be performed by software or by human effort although ideally, the generation of the layout should be completely automatic in order to keep design costs to a minimum. However, automated IC layout usually produces IC layouts which are larger and slower than hand-crafted designs, and as a result of the trade-off between the lower design cost of automated design versus the efficient use of silicon by hand-crafted designs, three approaches to the generation of IC layout have evolved. They are gate-array designs, standard-cell designs, and full custom designs.

1.2.1. Gate-array Design

Gate-array IC design begins with a chip where rows of transistors are fabricated except for the interconnecting metal layers. The customer customizes the chip for his application by designing the layout for the interconnecting metal layers.

The transistors making up a gate-array IC are grouped into cells which in turn are placed in rows as depicted in Figure 1.1. A cell is made up of 4 to 16 transistors and the logic function which it performs is determined by the transistor inter-connections made when the metal layer is added. Figure 1.2 shows the layout for a simple 4 transistor cell, and Figure 1.3 shows how two functions can be made by changing the wiring of this basic four transistor cell.

To design a gate-array IC, the designer designs a logic circuit by specifying the inter-connection of logic functions either by schematic entry on a graphics terminal or by a text description called a net-list. Table 1.1 gives a list of logic functions typically available to the designer. From the designer's logic description, the generation of the layout of the interconnecting metal layers is almost completely automatic although it is possible that software cannot complete the inter-connections between the cells because the space available for cell inter-connection is fixed. Generally, though, if 80% of the cells are used, software can complete the layout without human intervention [6].

1.2.2. Standard Cell IC Design

Standard cell IC design is based upon a function library. The function library contains a complete layout description (often referred to as a cell) for several logic functions. As with gate-array design, the designer specifies the circuit in terms of logic functions in schematic or net-list form, and software automatically generates the layout for the design. To generate the layout, the software selects the appropriate cells from the library, places them in a

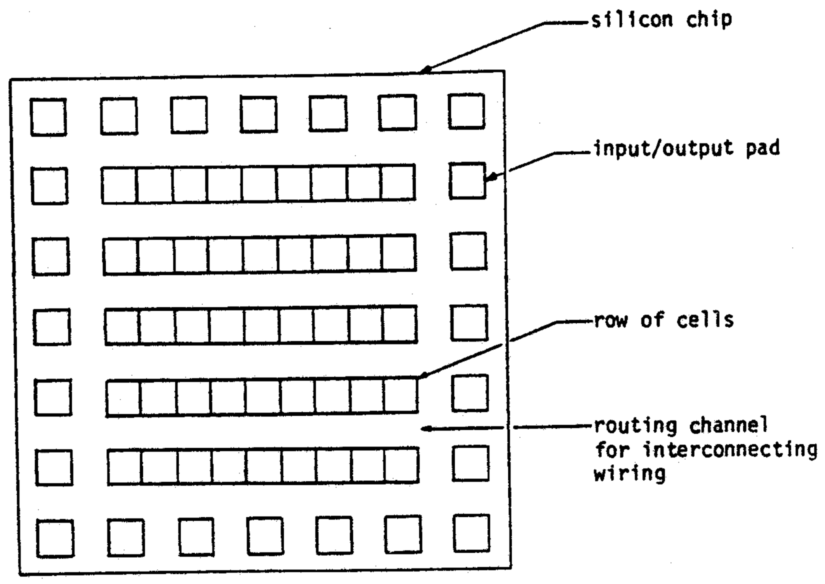


Figure 1-1: Floorplan of a gate array IC.

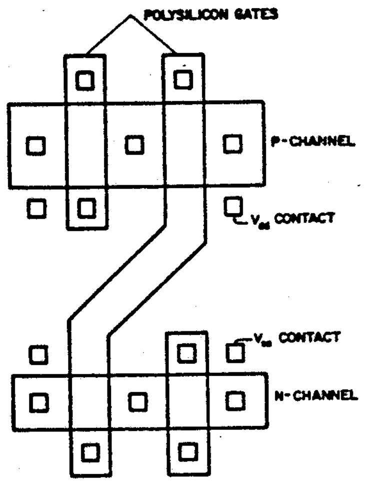


Figure 1-2: Layout of one gate array cell [6].

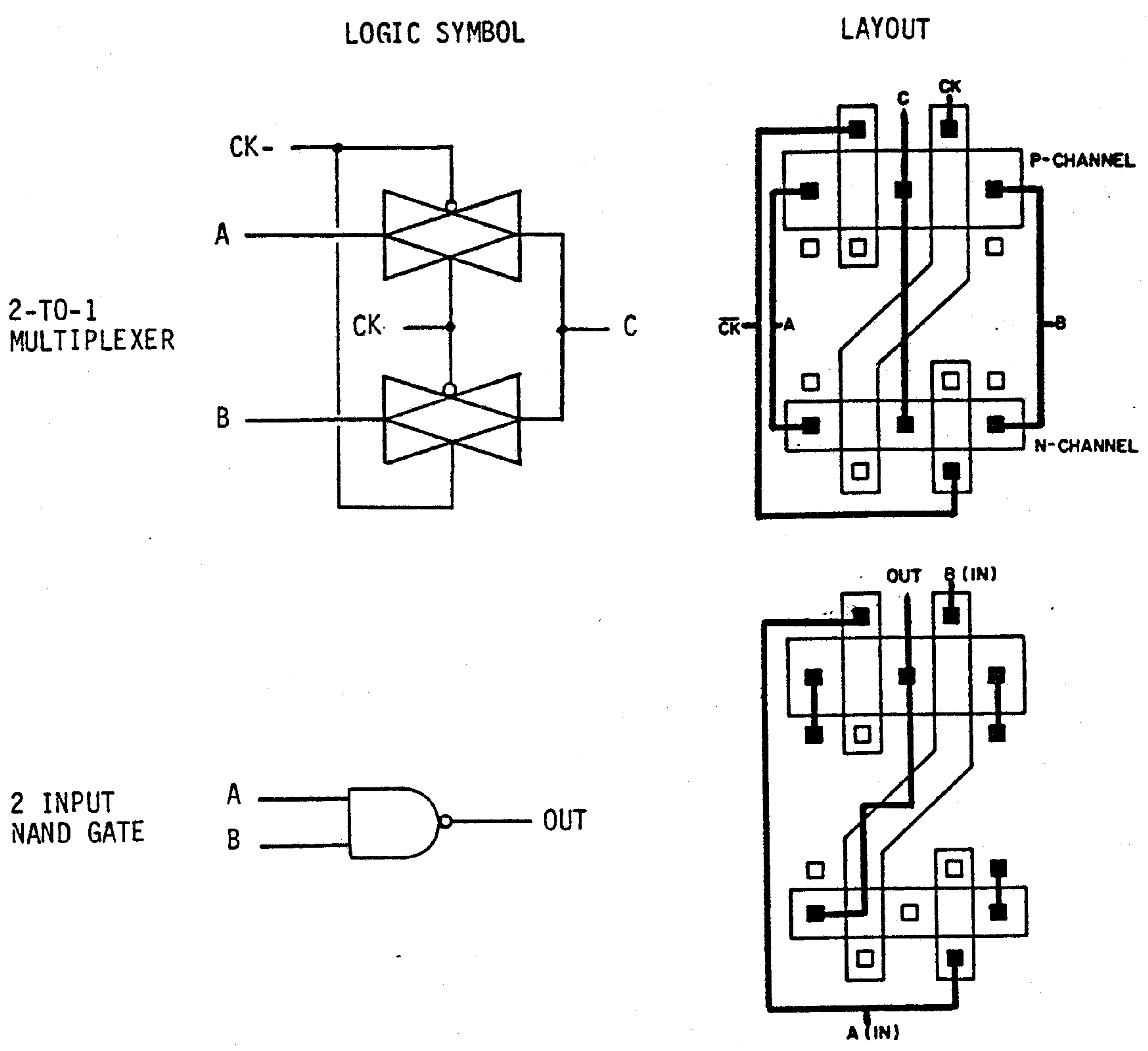


Figure 1-3: Two examples of a logic function created from a gate array cell [6].

NAND,	Triple 2-input
NAND,	Dual 2-input plus complement
NAND,	Dual 3-input
NAND,	Triple 3-input plus complement
NAND,	Triple 4-input
NAND,	5-input
NOR,	Triple2-input
NOR,	Dual 2-input plus complement
NOR,	Dual 3-input
NOR,	Triple 3-input plus complement
NOR,	Triple 4-input
NOR,	5-input
XOR,	2-input
AND-NOR,	2-2 with complement
OR-NAND,	2-2 with complement
Latch,	2-input NAND R-S with 2-I NAND
Latch,	2-input NOR R-S with 2-I NOR
D-flip-flop	
D-flip-flop	with set & reset
Invertor,	Quad
Buffer,	Clock, triple
Buffer,	Dual tristate inverting
Buffer,	Tristate non-inverting
MUX,	2-I

Table 1-1: A list of logic functions available to the gate array designer [6].

manner which minimizes layout area while keeping the routing problem as simple as possible, and then generates the interconnecting wiring to complete the layout of the chip. Unlike gate-array design where only the metal layer layouts are made by the customer, in standard cell design, the layout for all layers is specified by the customer. This makes initial production costs more expensive for standard cell design. However, standard cell design produces smaller designs than gate-array because unneeded functions are not included in the design. Performance is also better than gate-array designs because the layout for each function is optimized for that particular function.

1.2.3. Full Custom Design

The most labour intensive IC design method is the full custom design; here, the designer can specify the size and placement of every transistor by using an interactive colour graphics program to draw the layout.

Even though the designer has the ability to layout every transistor, this does not mean that the full custom layout must be completely generated by hand. Because of the cost of human labour, attempts are made to automate as much layout generation as possible. For example, the program TPLA [7] was used for this IC design to produce the layout for a programmable logic array.

The ideal layout generator would be a silicon compiler. A silicon compiler produces layout from a high-level description that allows the user to specify the IC function in terms of operations like multiplication and addition while hiding the circuit timing constraints and electrical circuit characteristics. One example of a silicon compiler is the FIRST silicon compiler [8]. FIRST provides the designer with primitive functions such as MULTIPLY, DELAY, and MULTIPLEX. Once the designer describes the IC function in terms of these primitives, FIRST develops the layout for the primitives using composition procedures and a cell library, and then combines these primitive layouts with an algorithm which tries to minimize the overall layout area. After the primitive modules have been placed, routing algorithms complete the inter-connections.

Because the full custom design approach has not yet been completely automated and because the automated design tools which have been developed are not yet refined enough to generate layout as well as humans, the full custom design still requires a large amount of manual design. As a result of this large initial design effort, full custom designs have the highest start-up costs among the three design approaches presented.

1.2.4. A Comparison of Three IC Design Approaches

Based upon an actual IC design, a general comparison of the layout area and design times of each of these three design approaches is given in Table 1.2.

		Gate Array	Standard Cell	Hand-crafted Full custom
	Layout Area	100%	75%	40%
Design Time	2200 gates	2.5 months	4.0 months	13.0 months
	6000 gates	3 months	4.5 months	17.5 months

Table 1-2: A comparison of the design time and efficiency for different IC design approaches [9].

As expected, the gate-array design approach required the least design effort, but the resulting chip area consumes over two times the area of a hand-crafted design.

The true cost of an IC chip is [10]:

$$\frac{\text{COST}}{\text{IC}} = \frac{\text{development cost}}{\text{number of IC's}} + \text{unit production cost}$$
$$+ \text{unit test and packaging cost}$$

Therefore selection of the design approach depends on the volume of production as well as development time and performance requirements. The economics of the three different approaches to IC design, gate-array, standard cell, and full custom, are compared to a design based on standard IC's in Figure 1.4.

The approach selected for this thesis is the full custom design approach. At the time this project was initiated, this approach was the most well developed.

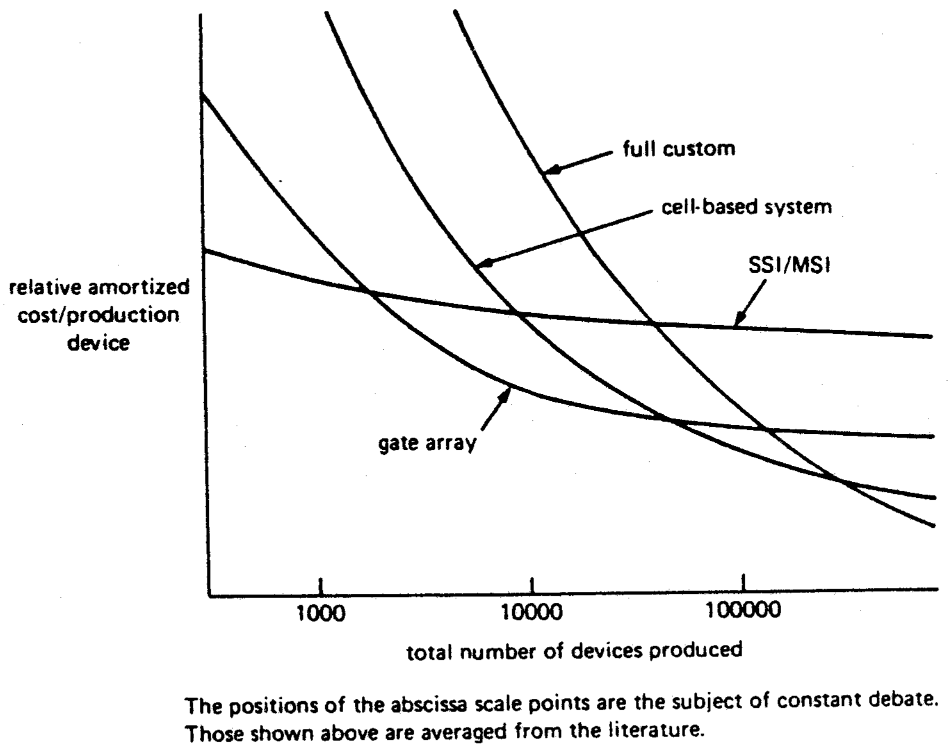


Figure 1-4: The economics of the different IC design approaches [10].

1.3. Advantages and Disadvantages of Custom IC Design

Implementing a function in a custom designed IC has advantages and disadvantages over an implementation using standard off-the-shelf parts. One major advantage of a custom IC design is a reduced part count of the final product. Fewer parts means circuit assembly cost is reduced, the final product is more compact, and less power is consumed. Increased reliability is another reported advantage of custom IC design. Reference [11] reports that large scale IC's have a failure rate one seventh the failure rate of an equivalent number of circuits made from small scale IC's. On the other hand, one disadvantage of custom IC design is that once fabricated, changes to the circuit are expensive and are not done except during the development stage where microsurgery techniques like cutting the metal layer wires with a trace cutting probe and a microscope can be used. This contrasts sharply with a design based on a microprocessor where software modifications are possible at any time.

1.4. Application of IC Technology to Computer-aided ECG Analysis

As mentioned earlier, ECG processing requires a large amount of computation. This computation is highly repetitive; a few functions like square-root, multiplication, and addition are repeated 961 times for each two ECG patterns being compared. This computation can be performed using standard parts such as microprocessors, but such devices are general purpose machines and have a certain amount of overhead in terms of execution speed and hardware. One solution to this problem is to take the approach taken by many special purpose IC's such as direct memory access controllers, math co-processors, and input/output processors whose purpose is to off-load time consuming chores from the central processing unit. The proposed IC design has a similar purpose in that it would relieve the central processing unit of the computationally expensive part of ECG analysis. The proposed IC design would offer a significantly faster (in terms of number of ECG patterns analyzed per unit time) and smaller (as measured by circuit board area) product than one based on a general-purpose microprocessor. An application specific IC can offer these advantages because:

- The function to be performed is built into hardware. There is no need to fetch and decode instructions. ←

- The circuit can be designed to take advantage of concurrency in an algorithm. For example, one part of a circuit may be taking the square-root of a number, while another part is simultaneously doing a multiplication. ←

- IC packaging can occupy several times the area of the IC chip. By combining several standard parts into one IC, the area taken by IC packaging is reduced. As well, the interconnecting wiring on a circuit board is scaled down when it is included on the IC.

1.5. Project Objective

The objective of this thesis is to design an IC which performs the morphology recognition calculations in ECG waveform analysis. The algorithm for this morphology recognition has already been developed and tested in the form of a computer program. The requirements for this IC design are:

- Perform the morphology recognition in a manner described in [5]. This algorithm has already been tested; significant changes should be avoided otherwise further testing of the algorithm might be necessary.
- The ability to process several ECG waveforms (heart beats) in real time. No specific figures are given here because initially the capability of IC technology for this application was not well known. Instead, once the capability of the ECG processor is known, it will be compared to other methods of computing morphology recognition.

The design of this integrated circuit, called the ECG processor, involves the following steps:

Step 1) Learn the use of software tools necessary for IC design. The design of an IC is done almost entirely using computer software. NETLIST [12], RNL [12], and MAX [13] are a few of the computer programs used to design an IC. NETLIST is used to enter the circuit design at the transistor level while RNL is used to simulate the operation of the designed circuit. MAX lets the designer create the layout used to manufacture the IC. The layout produced by the designer using MAX is converted to CIF [14] code, and the CIF code is sent to the fabrication facility where it is used to make the IC.

Step 2) Develop the behavioural specifications of the IC. Behavioural specifications describe how the IC functions. The specifications include defining what outputs will appear for a given set of inputs, restrictions on the timing of the input signals, and expected timing of the output signals. There are no formal rules for generating behaviour specifications, therefore it is possible to make specifications which cannot be realized due to limitations of the IC technology. Generating behavioural specifications involves some feedback from the following design step either in the form of the designer's experience, or in the form of modifications to the specifications as the design progresses.

Step 3) Design of the logic which performs the functions defined in the previous step. The design of the logic circuit usually involves the 'divide and conquer' approach where the circuit to be designed is divided into sub-circuits which are either previously designed or can be designed with little effort.

Step 4) Develop the layout which is the actual physical plan used to fabricate the IC. The transistors and their inter-connections have been defined in the previous step, the designer must now draw the circuit in the form of the layout.

Step 5) Test the IC to verify its operation. For this project, the completed IC design is not tested, but two sub-circuits which were part of this design are tested. To test the IC's, a data generator is used to supply a set of user defined digital inputs to the IC, and the response to the applied inputs is recorded by a logic analyzer. The recorded data are then compared to the expected set of outputs.

1.6. Outline of Thesis

This thesis is divided into six chapters. Aside from this introductory chapter, a brief overview of each of these chapters is given below.

Chapter 2 introduces the terminology of ECG analysis systems and provides the background information regarding the ECG analysis algorithm for which this IC is being developed.

Chapter 3 presents basic information useful for understanding the design and operation of an IC. It covers the procedure for designing large scale IC's as well as discussing the computer programs used for IC development.

Chapter 4 presents the development of the IC for the application in the ECG analysis system discussed in Chapter 2. It describes the flow of data in and out of the IC and the operations the IC performs upon this data. This chapter also provides a description of how the IC would be used as part of an ECG analysis system.

Chapter 5 covers the design of the logic and electrical circuits necessary to perform the functions presented in Chapter 4. It also includes the test results of two fabricated sub-circuits, a discussion of the simulation of the complete IC design, and a comparison of the proposed IC design's performance to a general purpose microprocessor.

The final chapter concludes the thesis and summarizes the results of the thesis work.

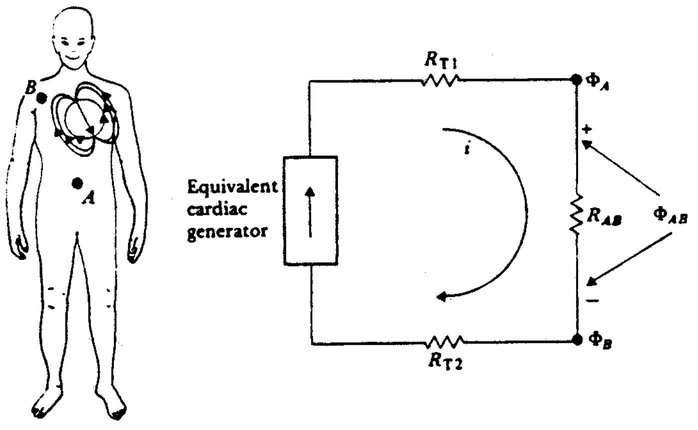
Chapter 2
Morphology Recognition of ECG Waveforms

This chapter introduces the reader to the topic of ECG analysis. It provides an explanation for the operation of a computer ECG analysis system as well as a description of the ECG analysis algorithm for which the proposed IC design is being developed.

The first section of this chapter discusses the origin of the ECG signal and provides a few ECG waveform examples. The second section discusses the general nature of computer ECG analysis, and the third section describes the ECG analysis algorithm which is to be implemented on a custom IC design.

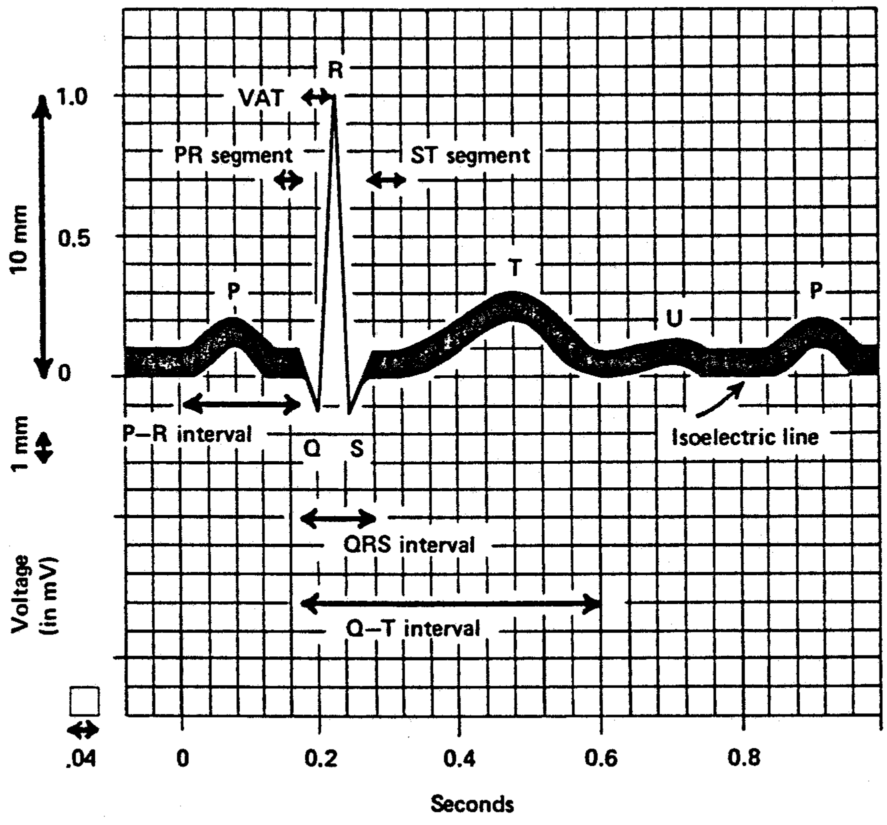
2.1. The ECG Waveform

The electrocardiogram (ECG) is a recording of the electrical signals produced by the beating of the heart. Because muscle contraction is an electrochemical process, the contractions of large muscles like the heart induce small ionic currents throughout the body which can be detected by measuring the voltage between two electrodes placed on the skin of a patient. The simple electric circuit model in Figure 2.1 is an illustration of this process where the heart is considered a current source, and the voltage measured with the electrodes is generated by current flowing through the resistance that exists between two points on the body surface. When plotted along a time axis, this voltage traces a distinctive shape which provides useful information on how the heart is functioning.



what is φ?

Figure 2-1: The electric circuit model of electrocardiography [15].

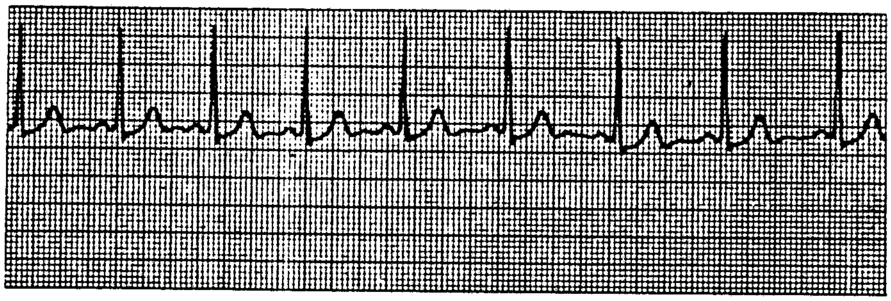


P-R interval?

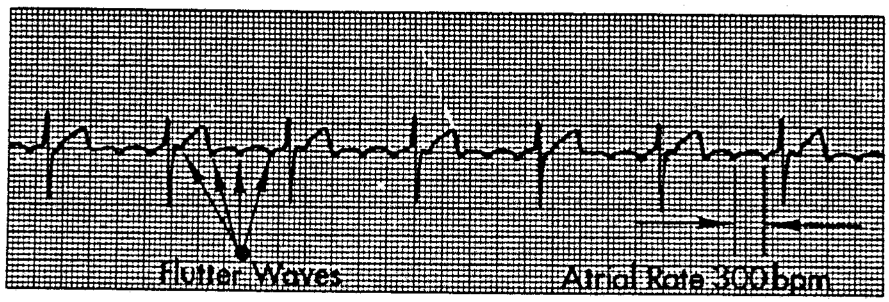
Figure 2-2: Diagram of a normal ECG trace with labelled segments [16].

The wave shape of an ECG signal is due to the fact that all heart muscle cells do not contract simultaneously; the contraction begins at one point on the heart, and through a complex network of conductive tissue, the muscle cells are stimulated and contract in a pre-determined order to produce the characteristic ECG waveform shown in Figure 2.2. This figure also shows the labels given by convention to the various segments of a normal ECG waveform. The P-wave corresponds to the contraction of the atria to pump the blood into the ventricles, and following the P-wave by about 0.07 seconds to allow blood to flow into the ventricles, the ventricle muscles contract to produce the highly visible QRS waveform. The repolarization of the heart muscles in preparation for the next beat causes the T wave.

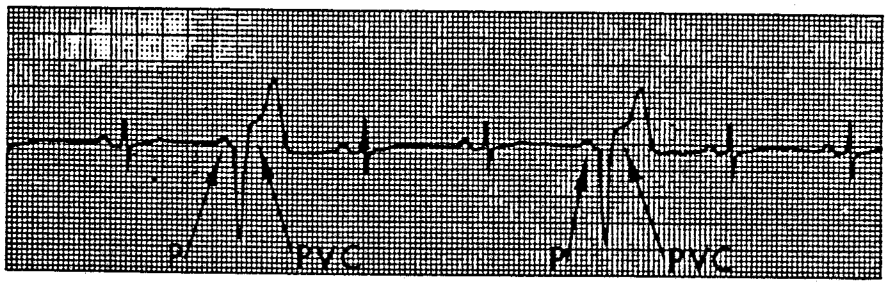
Deviations from the normal order of contraction will show up in the ECG trace as a waveform different from a normal waveform. Some examples of these abnormal heartbeats or arrhythmias are given in Figure 2.3. The atrial flutter in Figure 2.3b is the rapid beating of the atria (typically 300 beats per minute) while the ventricles continue to contract in an almost normal manner at a rate one half to one quarter of the atrial contraction rate. The premature ventricular contraction (commonly called a PVC) in Figure 2.3c is a common arrhythmia which even occasionally occurs in an undiseased heart. However, in a patient with heart disease, PVC's may be a warning sign of serious heart problems. Figure 2.3d is an example of ventricular fibrillation and is fatal if attempts to restore normal heart rhythm are not made immediately. These three examples are only a part of a list of several classes of arrhythmias. Besides the obvious shape or morphology which characterizes some arrhythmias, diagnosis of heart disease by ECG also includes measurements on the time delay between the P and R waves (the P-R interval), the height of the various waves, the duration of the QRS wave, the duration of the RS-T wave (the ST interval) and many other characteristics of the ECG waveform. Experienced physicians use this information from the ECG signal along with other patient data to determine the health of the heart.



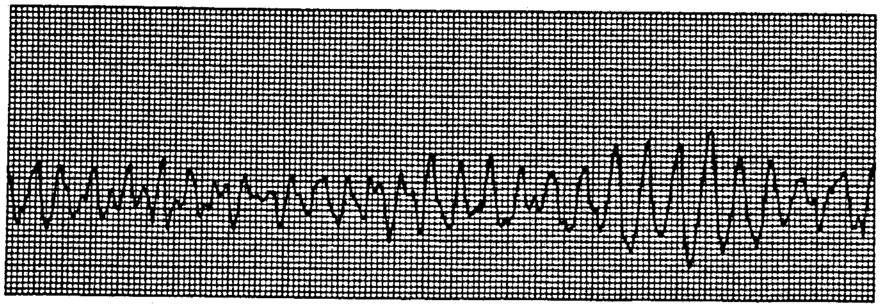
a) ECG of normal sinus rhythm



b) ECG trace of atrial flutter



c) ECG trace of a PVC



d) ECG trace of ventricular fibrillation

Figure 2-3: Examples of ECG traces [17].

2.2. The Automatic Interpretation of ECG waveforms

The analysis of arrhythmias is based upon the observation that arrhythmias may occur frequently or rarely, but certain types of arrhythmias will give approximately the same ECG wave shape from one occurrence to the next. It is this property that ECG morphology recognition programs can take advantage of because once an abnormality is recorded, its next occurrence can be detected almost immediately by maintaining a list of the various waveforms which have occurred in an ECG recording.

One point, though, should be made about the repeatability of ECG waveforms. Gradual changes in the morphology of a normal QRS may occur due to changes in the patient's body position and changes in skin-electrode contact over time. These changes have to be accounted for by continuously updating the reference patterns being used in morphology recognition [18].

The first step in the analysis of ECG's is the detection of a heartbeat. This means locating the QRS waveform using methods such as a search for one or more slopes in the waveform which exceed certain timing and voltage thresholds [4].

Once a QRS waveform has been located, the waveform for that heartbeat must be characterized in a way that allows the separation of normal heartbeats from abnormal heartbeats and possibly the identification of the type of abnormal heartbeat. Two common techniques used to identify the ECG waveform are correlation methods and feature extraction methods. Feature extraction methods are based upon the measurement of certain properties of the ECG waveform such as the QRS duration, the QRS height, and the area under the QRS wave [4]. Correlation techniques, on the other hand, make a direct comparison of an unknown waveform with several different known waveforms called templates.

After the ECG waveform has been characterized, some procedure or set of decision rules must be followed to obtain a classification of the unknown waveform and the results reported to medical personnel. The results are reported in a manner dependent upon the purpose of the ECG analysis system. For example, diagnosis programs may indicate several diagnoses and provide a probability for each diagnosis as does the U.S. Veteran's Administration ECG analysis system while a monitoring system would provide an alarm, an indication of the rate of occurrence of PVC's, and tracings of the waveforms which caused the alarms.

2.3. The Hilbert Transform method for computer-aided ECG analysis

This section introduces the Hilbert transform method for computer ECG analysis. First, an outline of the proposed ECG analysis system is given in the form of a block diagram. This shows which part of the ECG analysis algorithm is to be implemented on the IC and how it is to be connected to the ECG analysis system. Next, a brief summary of the properties of the Hilbert transform is given. In the last part of this section, two important aspects of the ECG analysis algorithm are presented: the recognition matrix and the measure of similarity between matrices. They are important because the recognition matrix is the data upon which the ECG processor is to operate on, and the similarity measure is the function to be performed by the ECG processor.

2.3.1. The ECG Analysis Computer System

Figure 2.4 shows the block diagram for the proposed ECG analysis system based upon the Hilbert Transform method of morphology recognition. The host computer executes the ECG analysis programs and performs the data transfer to and from the peripherals. The graphics terminal displays ECG waveforms and the alphanumeric terminal interacts with medical personnel for the purposes of classifying or verifying unknown waveforms, providing information on the ECG's, and giving warnings of potentially

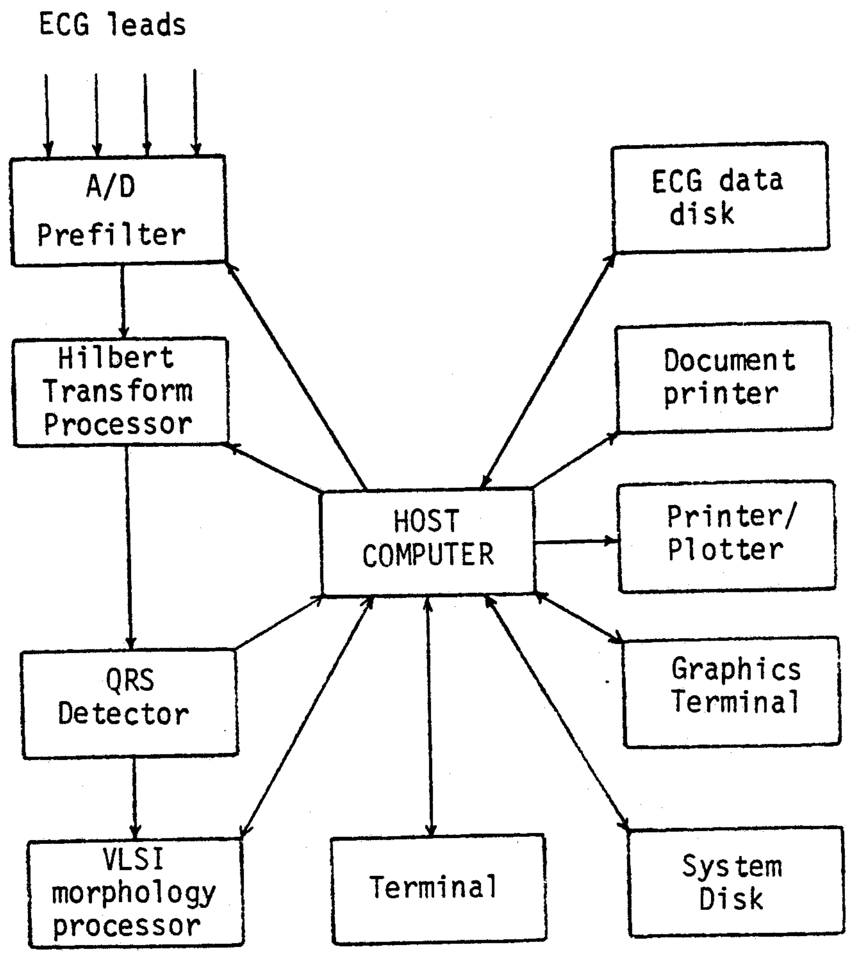


Figure 2-4: Block diagram of a clinical ECG analysis system [5].

serious heart problems as indicated by ECG analysis. The printer and plotter provide hard copies of requested information for documentation purposes. Two disk drives are used to speed up the ECG data storage; when the computer is switching between ECG analysis and other background programs, the use of two disk drives prevents delays due to disk read/write head movement. The analog to digital converter's resolution has not yet been specified, but 8 bits would be reasonable for this system. The sampling rate has not been defined, but a value of 500 samples per second is used in [5] and so it is the value used in this thesis. The pre-filter includes an analog anti-aliasing filter and possibly a digital or analog filter to limit the input bandwidth and remove 60 Hz noise. The Hilbert transform processor, the QRS detector, and the VLSI morphology processor (the morphology processor is the ECG processor being designed) are functions which were performed in software in the research stage, but for a real-time system, they would be performed in hardware because their execution in software would be too slow. The Hilbert transformer (the Hilbert transform will be defined later) calculates the Hilbert transform of the ECG signal. The QRS detector locates the peak of the R-wave for each heartbeat using the Hilbert transformed signal. This signal is used to identify the R-wave peak because the Hilbert transformed signal makes a zero crossing at major inflection points like the peak of the R-wave in the ECG signal (this is true only if the peaks are well separated). The hardware for both the Hilbert Transform processor and the QRS detector are still under development. The morphology processor is the ECG processor being developed in this thesis, and its purpose is to provide the host computer with measures of similarity between each newly recorded ECG waveform and a list of previously classified reference waveforms.

One issue in the computer analysis of ECG's is the selection of the leads used for the analysis. The position of electrodes on the body surface of the patient is important in ECG analysis because their position directly affects not only the morphology of the recording of a normal heartbeat, but also how well certain abnormalities appear in the ECG. For this reason, the number of electrodes used and their placement depends on the purpose of the ECG analysis. A diagnostic system which is used by a physician to help identify a specific heart disease uses 3 to 12 leads (the signal from one lead is the potential difference between two electrodes). Here, many leads are used for diagnosis since certain ECG abnormalities are more visible in one lead position over another lead position and the physician (or diagnostic computer program) wants as much information as possible before making a diagnosis. In contrast to the diagnostic system which does not necessarily operate in real-time and allows computation time to be traded for diagnostic accuracy, a monitoring system must operate in real-time. To keep the hardware costs low for a monitoring system, only one or two leads are monitored per patient. Single lead monitors are the most popular, but dual lead monitors have two advantages. One advantage is a reduced sensitivity to artefact (artefacts are signals which appear in an ECG but are not generated by the heart) since artefacts may appear on one channel and not the other. In this case, analysis continues as long as one good waveform is available from either lead. The ARGUS monitoring system has been developed for both single and dual channel ECG analysis, and experience showed that the single channel version cannot analyze about 5% of the recorded ECG data while the dual channel system was unable to analyze about 2% of the ECG data [19]. This increased rejection of artefacts would be useful in reducing the number of false positive PVC detections. The second advantage of the two lead monitoring system is the ability to recognize a greater variety of arrhythmias [1].

The ECG analysis system in [5] for which the ECG processor IC is being developed is based upon a research system which uses one lead per patient. However, the number of leads monitored per patient does not necessarily affect the similarity calculation between two waveforms and one consideration in the design of the ECG processor is that the number of leads used per patient should not affect the operation of the ECG processor.

2.3.2. The Hilbert Transform

The key function upon which this morphology recognition algorithm is based is the Hilbert transform. It is defined as:

$$\mathrm{h(t)} = \frac{1}{\pi t} * f(t) \qquad \text{(where * denotes convolution)} \tag{2.1}$$

where f(t) is the input signal and h(t) is the Hilbert transform of f(t). The impulse response and the frequency response of a discrete Hilbert transformer are given in Figures 2.5 and 2.6. The ideal Hilbert transformer does not

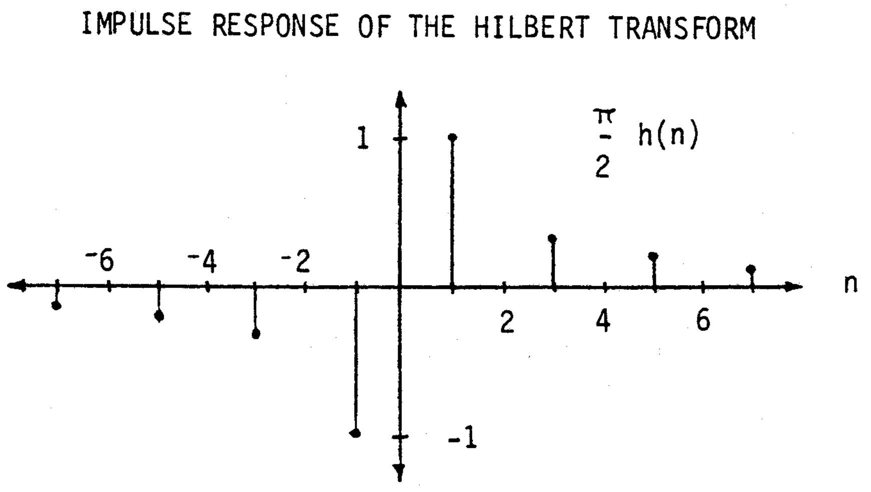


Figure 2-5: Impulse response of the Hilbert Transformer.

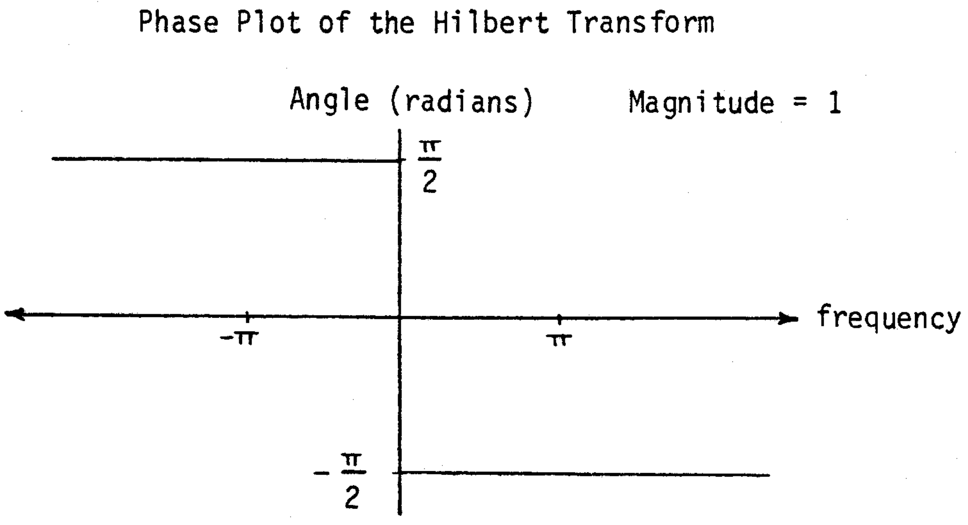


Figure 2-6: Frequency response of a Hilbert Transformer.

exist and practical circuits provide only approximations of the Hilbert transform. For the application of morphology recognition, no tests have been done to determine the tolerance of the morphology recognition algorithm to approximations of Hilbert transforms, however, the frequency response and the phase response should be as flat as possible in order to preserve the time independent representation feature provided by this algorithm.

The most unique feature of the Hilbert transform morphology recognition algorithm is the mapping of the input signal into a representation called the recognition matrix. Other morphology recognition algorithms usually isolate the waveform of one heartbeat and perform correlation calculations between it and several reference patterns called templates. One problem with this technique is that alignment along the time axis is necessary for correlation calculations and the correlation calculations are sensitive to even slight misalignments. Also, the alignment is difficult to do for waveforms of different shapes. The Hilbert Transform technique eliminates the problem of alignment along the time axis by representing the ECG waveform shape in a way that eliminates the time dependence of the data. The principle of this representation is demonstrated by plotting the Hilbert transform of the signal against the signal itself as shown in Figure 2.7. The resulting plot has no time axis and is the same for two waveforms whose shape is the same, but whose durations may be different. For example, two sinusoidal waves of the same amplitude, but of different frequencies both produce identical circles when plotted in a fashion similar to Figure 2.7.

2.3.3. The Recognition Matrix

The plot in Figure 2.7 is just a demonstration and not the true representation used in the morphology recognition algorithm. The actual representation used by the Hilbert Transform morphology recognition algorithm is a 31 by 31 element matrix called the recognition matrix. An example of a recognition matrix is shown in Figure 2.8. Experiments carried

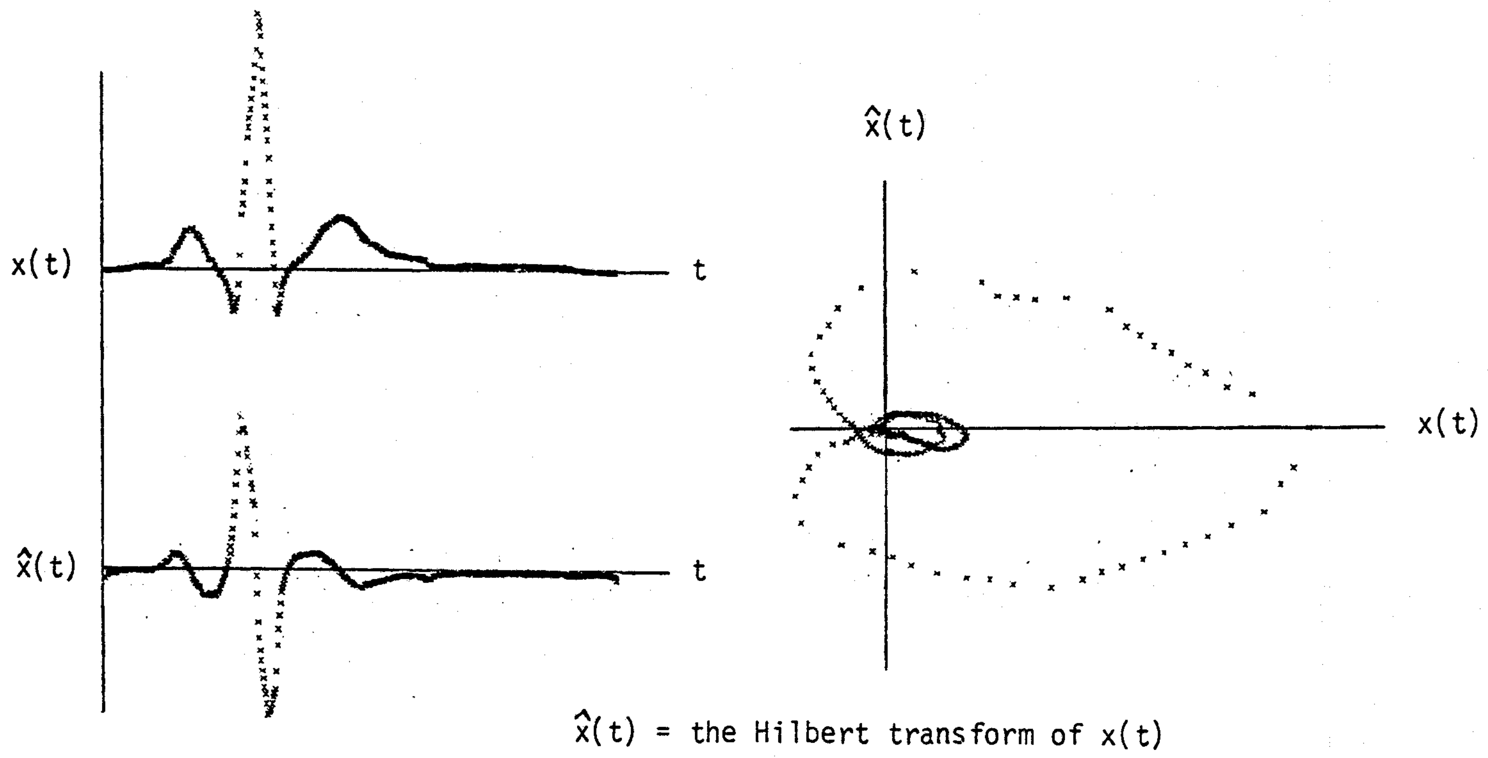


Figure 2-7: Generation of the Hilbert Transform representation.

out in [5] showed that the morphology recognition algorithm worked well for matrix sizes of 15 by 15 to 31 by 31 elements. The larger size matrices provided the greatest accuracy and so were selected for the IC design. It was also found that matrices should have an odd number of rows and columns to prevent dithering when the signal is on the baseline.

Another view of the recognition matrix of Figure 2.8 is that of a two-dimensional histogram as shown in Figure 2.9. In this view, each element of the histogram can be considered to correspond to the number of ECG signal sample points which were mapped into that particular area of the histogram. To create a histogram, all elements are initially set to zero. Then by taking one signal sample point at a time, the elements of certain locations of the histogram are incremented by user defined amounts. The finished histogram is the cumulative effect of the incrementing after all the sample points of one

```
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0                                                                                         0
  0                                                                                         0
  0                                                                                         0
  0                                                                                         0
  0                                                                                         0
  0                                                                                         0
  0                                                                                         0
  0                                                                                         0
  0                                  1  2  4  4  4  2  1                                    0
  0                               1  2  5  9 11 11  7  5  4  3  2  1                        0
  0                               2  5  6 11 15 16 15 13  8  5  5  3  1  1                  0
  0                            1  4  7  8  9  9 14 17 16 11  8  6  5  4  2  1               0
  0                            2  6  9  8  4  2  6 10 12  9  5  5  5  5  4  2  1            0
  0                            2 12 19 29 49 42 30  3  4  4  2  3  4  5  5  4  3  1         0
  0                            3 11 25 44 83 90 40  1           1  2  3  4  4  4  2  1      0
  0                         2  5 12 19 43 84 90 48  5  2              1  2  2  3  3  1  1  0
  0                      1  4  7  7  9 17 30 31 24 10  4                       1  2  2  1  0
  0                      1  6  6  8  6  6  7  9 12 11  6  1                       2  2  1  0
  0                      1  6  8  8  6  4  1  2  6  9  7  2                       1  2  1  0
  0                      1  5  8  9  6  4  2  3  7  8  6  2                       1  2  1  0
  0                      1  3  6  8  7  5  4  5  7  7  4  1                    1  2  2  1  0
  0                         2  4  9  9  7  7  7  8  5  2  1                 1  2  2  1     0
  0                            2  5  8  8  8  9  6  3  1              1  2  2  2  2  1     0
  0                            1  3  5  6  7  5  3  1           1  1  2  3  3  2  1        0
  0                               1  2  3  4  2  1  1        1  1  2  3  3  2  1           0
  0                                  1  2  2  2  2  1  1  1  1  1  2  2  2  1              0
  0                                     1  2  2  2  2  2  2  2  1  1  1  1                 0
  0                                        1  2  2  2  2  2  2  1  1                       0
  0                                           1  1  1  1  1  1  1                          0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure 2-8: An example of a recognition matrix.

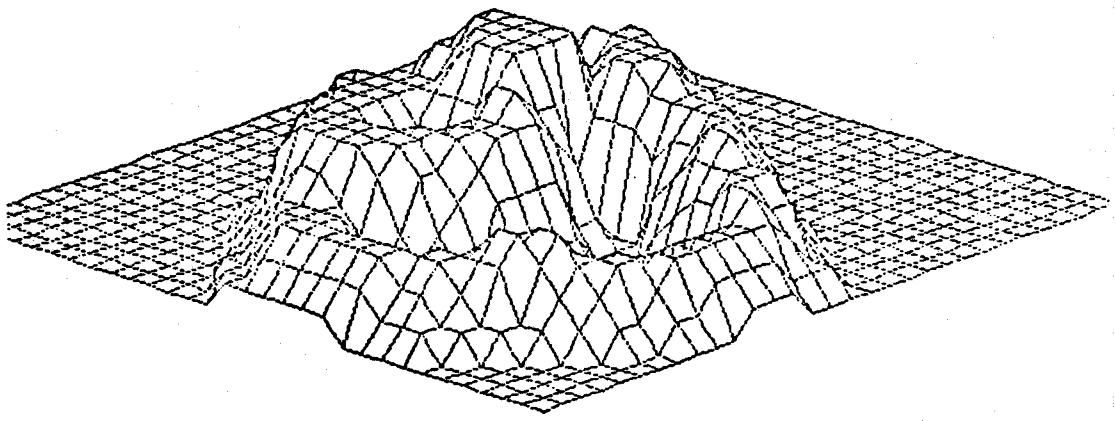


Figure 2-9: A view of a recognition matrix as a histogram [5].

heartbeat have been processed. For a typical heart rate of 70 beats per minute and a 500Hz sampling rate, this is typically 430 sample points. The locations on the histogram to be incremented are determined by the value of the digitized ECG signal, the corresponding Hilbert transform value, and the mapping function. The amounts by which the elements are incremented by are given by the potential function shown in Figure 2.10.

		COLUMN OF RECOGNITION MATRIX		
		j-1	j	j+1
ROW OF RECOGNITION MATRIX	i-1	1	1	1
	i	1	2	1
	i+1	1	1	1

Figure 2-10: The potential function used in this thesis.

The mapping function is performed after all the sample points have been collected for one heartbeat. The mapping function as defined in [5] is based upon the following definitions and equations:

$$x(n) = n^{th} \text{ sample point of ECG signal} \tag{2.2}$$

$$\chi(n) = n^{th} \text{ sample point of the Hilbert transformed signal} \tag{2.3}$$

$$x(n) + j\chi(n) = \text{pre-envelope of } x(n) \tag{2.4}$$

$$z(n) = \sqrt{x(n)^2+\chi(n)^2} = \text{ envelope or magnitude of } x(n) \tag{2.5}$$

$$z_{max} = \text{the maximum magnitude for one heartbeat} \tag{2.6}$$

$$\phi(n) = \text{atan}\left(\frac{\chi(n)}{x(n)}\right) = \text{angle of the pre-envelope} \tag{2.7}$$

$$2N_x + 1 = \text{number of rows in the recognition matrix} \tag{2.8}$$

$$2N_y + 1 = \text{number of columns in the recognition matrix} \tag{2.9}$$

$$K(i,j) = \text{the potential function centered on row i and column j (see Figure 2.10)} \tag{2.10}$$

$$A_{i,j} = \sum_{\text{all n}} K\left(\left(N_x+1+N_x \frac{z(n)}{z_{max}} \cos(\phi(n))\right), \left(N_y+1+N_y \frac{z(n)}{z_{max}} \sin(\phi(n))\right)\right)$$

$$= \text{element of the i}^{\text{th}} \text{ row and the j}^{\text{th}} \text{ column of the recognition matrix } \mathbf{A}. \tag{2.11}$$

The purpose of changing the ECG data and its Hilbert transform into the polar coordinates $z(n)\angle\phi(n)$ is to scale the data by z_{max} and then convert the scaled polar coordinates into the rectangular coordinates of the recognition matrix.

While Equation 2.11 is useful for a research system, a system operating in real-time would benefit by a mapping procedure which did not use the sine or cosine functions. In order to provide a real-time implementation of Equation 2.11 two methods of mapping the ECG data into a recognition matrix are presented here: the first method will give the same results as Equation 2.11 while the second method will give approximately the same results and is faster than the first method.

Method 1.

1. Find the square of the maximum magnitude of the envelope, z^2_{max}, by finding the largest value of $x(n)^2 + \chi(n)^2$.

2. Calculate the scale factor

$$a = \frac{\text{maximum possible representation of x(n)}}{\sqrt{\frac{z^2_{max}}{4}}}$$

The maximum possible representation of x(n) depends on the number of bits used to represent x(n); if 8-bits are used, then the maximum value would be 255. The purpose of dividing z^2_{max} by 4 is to speed up the calculation of the square-root. If $x(n)^2$ and $\chi(n)^2$ are 8-bit values, z_{max} could be a 9-bit value (see Equation 2.5) which could require the use of 16-bit arithmetic on some microprocessors, but dividing by 4 permits the use of 8-bit hardware or software math routines for the scale factor calculations.

3. Multiply each sample point by the scale factor.

$$x_s(n) = a\ x(n)$$

$$\chi_s(n) = a\ \chi(n)$$

Each scaled sample point is now ready to be used as a pointer to the row and column of the recognition matrix to which potential functions are to be added.

4. For each sample point in the ECG waveform, add the potential function to the location determined by

row = 5 most significant digits of the scaled value $x_s(n)$

column = 5 most significant digits of the scaled value $\chi_s(n)$.

The phrase *5 most significant digits* is used because it does not assume to resolution of the sample points is known. It is another way of saying scale the highest value of the scaled values, say, for example 255, to the largest row or column of the matrix (that is 31 in this case). This method applies only to unsigned integers, and the system designer must take care when applying the above algorithm to two's complement numbers.

Method 2.

1. Find the maximum value of both $x(n)$ and $\chi(n)$ and call it x_{max}.

2. Calculate the scale factor

$$a = \frac{\text{maximum possible representation for } x(n)}{x_{max}}$$

3. Follow through steps 3. and 4. of method 1.

This method will scale all waveforms into the recognition matrix, but not in the same way as Equation 2.11. While method 1 scales according to the maximum magnitude of the envelope $z(n)$, method 2 scales according to the maximum value of the sample points in $x(n)$ and $\chi(n)$. The selection of these two mapping algorithms is the decision of the person programming the host computer, and does not affect the design of the ECG processor.

2.3.4. Computing the Similarity between Matrices

Once a recognition matrix is created from newly acquired ECG data, it must be compared to several reference matrices in order to be classified. The function selected for the comparison is the Bhattacharyya distance function [20]:

$$\min : \quad Q = -\log\left[\int^{X} \sqrt{P_c(x)\, P_u(x)}\, dX\right] \tag{2.12}$$

where

Q = distance between the probability distribution $P_c(x)$ and $P_u(x)$

$P_c(x)$ = probability distribution for class c

$P_u(x)$ = probability distribution for class u

X = the pattern space or in this case the recognition matrix.

This distance function has been modified for use with recognition matrices. The first change is the elimination of the *-log* operator because it does not affect the decision on which two matrices are the best match. The new function is now a similarity measure:

$$\max : \quad Q' = \int^{X} \sqrt{P_c(x) P_u(x)}\, dX \tag{2.13}$$

where Q' = the degree of similarity between class c and class u

The above equations are for continuous distributions; for this ECG processing application, a discrete form is required:

$$\max : \quad Q_d = \sum_{i=1}^{2N_x+1} \sum_{j=1}^{2N_y+1} \sqrt{P_c\, P_u} \tag{2.14}$$

where

Q_d = the degree of similarity between the unknown class u and the reference class c

$P_u = \frac{U_{i,j}}{N_u}$

$P_c = \frac{M_{i,j}}{N_c}$

U = the unknown recognition matrix

M(c) = the reference recognition matrix for class c

N_u = the number of points in the unknown matrix

N_c = the number of points in reference matrix c.

Q_d is calculated for several reference matrices and the unknown matrix is assigned to the class which produces the largest value of Q_d provided that a minimum level of similarity is met. If the unknown matrix does not fit any of the reference matrices to a reasonable degree, the unknown matrix itself may be stored as a reference matrix for future classification calculations.

Because at first the square-root function appears to be a complex function whose implementation in IC circuitry would consume a large area on the IC, other distance measures for the classification process were considered. One example that appears to require less computational effort than the Bhattacharyya distance measure is the Patrick-Fisher distance measure [20]:

$$\min : \quad Q^2 = \int^{X} [p_1 p_1(x) - p_2 p_2(x)]^2 \, dX \tag{2.15}$$

When modified for use with recognition matrices, this appears to require only one subtraction and multiplication for each matrix element. However, computational complexity depends on the number of bits being operated on as well as the functions being performed, and consideration of this must be made when comparing these two distance functions. For example, the square-root function compresses the dynamic range of the input values whereas the square function of the Patrick-Fisher measure expands the dynamic range. This means that if the recognition matrix elements are 8-bits, their square-root will be 4-bits (more bits can be added to increase the precision, but the dynamic range is still covered by 4-bits). As the square-root is done first, all remaining functions need only operate on 4-bit data (for example, a 4 x 4 multiplier). The square function, on the other hand, will produce a 16 bit result which must later be operated on and stored. As the chip design progresses, it will be seen that the chip area required for the square-root extractor is offset by savings produced by the smaller data size of the square-root output.

One idea which was tested while looking for other similarity measures was simply to eliminate the square-root function and complete all other calculations as before. While no mathematical justification could be made for doing this, this test would ensure that the square-root extractor was a necessary part of the ECG processor. In one test, the similarity between various one-dimensional continuous probability distributions was computed. The test showed that in the worst case, the similarity between two obviously different distributions had a computed similarity measure equal to that of identical distributions.

Because this did not prove that eliminating the square-root function would always cause classification errors, another test closely resembling the ECG analysis application was made. In this second test, all combinations of 2x2 matrices and 3x3 matrices were compared to each other. The results of this test depended upon the sum of all elements in the matrices. If the sum of the elements was greater than two, there were misclassifications. If the sum of the elements was equal to two, then there were only ties, that is, the similarity between different matrices could equal that of a perfect match. A study of the matrices which were classified correctly and those which were not showed that errors occurred when a sharply peaked distribution was compared to a more even distribution of numbers.

The results of these tests showed that under certain conditions, there is a possibility that the square-root extractor could be eliminated, but further tests on recognition matrices used in ECG analysis would be necessary before taking this step in the ECG processor design.

2.4. Summary

In the first section of this chapter, the origin of the ECG signal was presented along with a few examples of normal and abnormal ECG waveforms. Next, a brief description was given on the operation of a computer ECG analysis system. In the final part of the chapter, the Hilbert transform method for ECG analysis was discussed and the two main features of this algorithm, the recognition matrix and the similarity measure, were described in detail. This description included two methods for generating a recognition matrix and the development of a similarity measure from the Bhattacharyya distance function.

Chapter 3
Integrated Circuit Design

This chapter covers two aspects of IC design. First, the techniques used to manage the complexity of the design of a large digital circuit are discussed, and then the computer programs used to design an IC are examined.

3.1. IC Design Procedure

VLSI design requires all of the complexity management discipline associated with complex software systems, but without the underlying simplicity of a single sequential machine. Not only must we deal with the problems of enormous concurrency, but we must map the entire design onto a physical medium, with constraints on space, time, and energy imposed by the laws of physics. [21]

As indicated by this quote, VLSI design requires some methodology to handle the large amount of data and the high number of decisions involved with IC design because for large IC's, the complete freedom to 'draw' anything on the silicon chip overwhelms the average human designer. This complexity of IC design arises from the number of factors which must be considered in IC design including:

1. Keep chip area to a minimum. Yield, defined as the number of good chips produced per total number of chips fabricated, decreases rapidly as area increases because particle contamination is the main cause of fabrication failure (see Table 3.1) and the probability of particle contamination is an exponential function of the chip area.

Particle contamination	30%
Design margin	6%
Photolithographic errors (e.g., alignment error)	9%
Material defects	6%
Process variation	9%
Total die loss	60%
Die yield (100% − die loss)	40%

Table 3-1: Typical sources of chip fabrication failure for a 2μ process [22].

2. Design circuits with some tolerance of circuit parameters. Transistor characteristics can vary from wafer to wafer, from one point on a wafer to another, and even across a large chip. To improve yield, circuits should be designed to allow for variations from planned circuit speeds. Another advantage of allowing for parameter variations is the possibility of using more than one vendor to supply parts.

3. Design for testability. Testability is a measure of the fraction of the total number of transistors in the IC which can be tested in a reasonable length of time. Because yield is low, every chip must be tested to separate good chips from bad, but a complete test which provides 100% confidence in the final product is sometimes difficult. On the other hand, testing a circuit insufficiently could mean a field repair costing hundreds of times more than the cost of the IC.

4. Make the chip easy to use. Like a programmer using a subroutine written by another programmer, circuit designers using the chip want to know how to use the chip while having to learn as little as possible.

5. Balance between a complex design which provides every function the chip consumer wants, and a simple design which is easier to design and test, or in other words, the IC designer must decide what functions are a worthwhile part of the design and which are not.

This section presents two common methods available to manage the complexity of IC design: the use of hierarchical abstraction, and the use of a global two-phase clock signal. Following this, another issue of IC complexity, the ability to test what has been designed, is discussed briefly.

3.1.1. Hierarchical abstraction

Abstraction means replacing an object with a simpler one that keeps only the most important information about the original object's interactions with the environment [23]. The value of abstraction is a data reduction which allows the designer to concentrate on the important aspects of the object during the design process. Since the average person cannot consider all the details of an IC design at one time, abstraction is important, and, as illustrated in Figure 3.1, it has been applied to three representations in IC design: the physical or layout description, the behavioural or functional description, and the structural or circuit description.

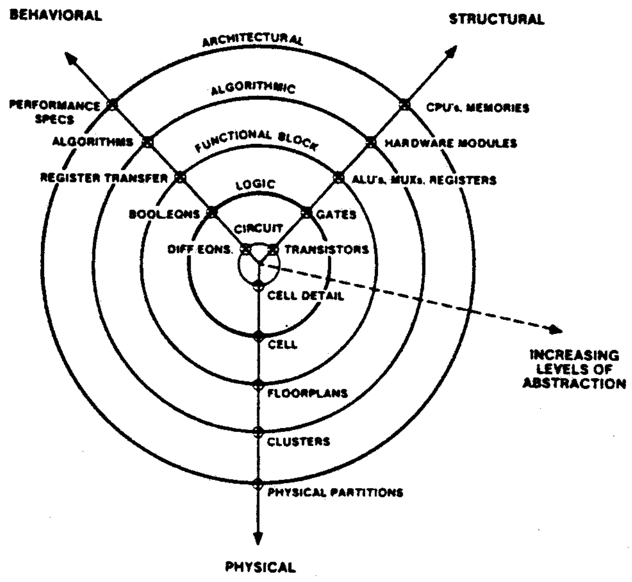


Figure 3-1: Levels of abstraction commonly used in IC design [9].

From this figure, it also can be seen that abstraction is not restricted to one level; it may be applied several times to obtain a hierarchical abstraction with the purpose of allowing the designer to comprehend all the necessary detail at each level in the design. For IC design, four or five levels of abstraction have been found sufficient.

Another name for this hierarchical design approach is 'divide and conquer'. Although the idea of a hierarchy is clear, the procedure for 'dividing' or partitioning a problem is not obvious. For instance, suppose the problem is to design a circuit to calculate $\sqrt{a\,b}$. The first step, defining the inputs and outputs is not difficult; the inputs are a and b, and the output is $\sqrt{a\,b}$. Unless a circuit exists to calculate $\sqrt{a\,b}$, the designer would attempt to do the obvious, that is, build the circuit up from a multiplier and a square-root extractor. The designer then develops a circuit where two 8-bit inputs are multiplied to give a 16-bit result and the 8-bit square-root extracted from the intermediate 16-bit result. This achieves the desired value, however, the problem could have been divided into $\sqrt{a}\sqrt{b}$. In this case, two 8-bit square-root extractors provide two intermediate 4-bit results which are multiplied to give $\sqrt{a\,b}$. Though it was originally not obvious, this division yields a more compact design than the first approach at the expense of precision in the resulting output. This example demonstrates the problem of partitioning in a hierarchical design.

3.1.2. System timing

Co-ordinating the flow of data through the various parts of the IC is another complex IC design problem. To help the designer organise the on-chip data communication, methods, such as, 2-phase clocks, 4-phase clocks, and self-timed signalling [14] have been developed. Synchronous systems which are based upon a global clock signal are used extensively in IC design; every microprocessor known by the author requires some type of clock signal.

The 2-phase clocking scheme is selected for the IC design in this thesis.

Its advantages include: it is a simple and easy-to-understand clock which means the designer should make few implementation errors, it prevents race conditions and closed feed-back loops, and it is useful for both static and dynamic circuits. But there are a few limitations of the two-phase clock. One problem is that the clock frequency is limited by the slowest circuit on the chip. For example, 99% of the logic functions may be complete within one tenth of a clock cycle, but a new clock cycle cannot begin until all logic functions have been computed. Another problem is distributing the clock throughout the chip so that the clock signal is the same everywhere in the system. This signal is used as a reference for on-chip data communication, and if the clock signal in one area on the chip is delayed relative to another area on the chip, communication problems may arise when data is being transferred between these two areas. One more problem, although minor, is synchronization failure which occurs when trying to read information into a synchronous system from an asynchronous source. This problem arises from the condition where it is possible for a flip-flop to exist in an unstable equilibrium or metastable condition for an indefinite length of time. In this state, circuit voltages are neither logic '1' nor logic '0', and although such a state has a small probability of occurring, in an example given in [14] a value of $3x10^{-12}$ is given which translates into an error every three days at a data transfer rate of 1 MHz. The only solution to this problem is to detect the metastable state and wait for it to end before continuing with the data.

The timing characteristics of the 2-phase clock are presented in Figure 3.2. ϕ_1 must be held high long enough to allow all inputs nodes to reach their appropriate state while ϕ_2 must remain high until all logic element outputs have attained their final state. Because both ϕ_1 and ϕ_2 must never be high at the same time, t_{12} and t_{21} are greater than zero to allow for clock skew between ϕ_1 and ϕ_2 and to allow for clock time differences among the separate areas on the chip.

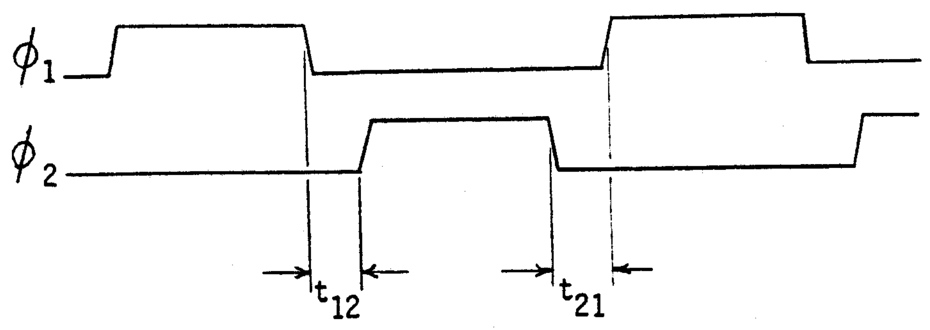


Figure 3-2: A two-phase clock signal.

3.1.3. Design for testability

The techniques for designing for testability in IC's are not presented here, but rather this is a brief introduction to testability to allow a further discussion in later chapters.

Designing for testability has been found to reduce the production cost of IC's [24]. Because of the low yield in IC production, an example of 40% is given in Table 3.2, all IC's must be tested as thoroughly as possible to ensure they work even though complete testing of an IC is difficult. The problem is trying to control the logic state of tens of thousands of nodes and observing their changes through a small number, say 64, pins. Unless consideration is given to testability at the design stage, test time, and so IC cost, grows exponentially with the number of gates in the IC design. Designing for testability, on the other hand, can make this test time growth almost linear [25].

Not many concrete figures on test times could be found in literature, but one article did briefly describe a designer's experience in designing for testability. In a report of one presentation at the 1985 IEEE International Test Conference [26], the test time reported for a 4,000 circuit (gate) IC was 5 seconds while the test time for a 32,000 circuit (gate) IC was 100 seconds (this time growth is not linear, but test time also depends on the type of circuits, and these are specific examples). Another benefit of testable circuits cited in [26] was the ability to debug parts of the circuit through the use of circuits designed specifically for circuit testing; in one given example, the design of an exclusive-or gate was found to be faulty. Aside from this report, a further illustration of the value of designing for testability is the observation that a 5 to 10% increase in chip area is considered reasonable for a reduction in test time [25].

3.2. Software for IC Design

For the development of the ECG processor, IC design software can be divided into three classifications: simulation software, layout software, and circuit extraction software. The available programs for each of these classifications and their use in this project are reviewed in the following sections.

3.2.1. Simulation software

Simulation is important in IC design because the high start-up costs for fabrication do not permit several prototype circuits. Even if cost were no object, testing these circuits is difficult because of the small size of the components and because of the limited drive capability of the IC transistors. On the other hand, a simulator is not subject to these limitations, and it can provide information on every node of a circuit.

The function of a simulator is to predict how a given circuit will act before it is actually fabricated. A simulator is based upon a model which approximates the basic components used in the design. The detail of the model generally determines the accuracy and speed of the simulator, and usually speed and accuracy are opposing forces. For this reason, different levels of simulation have been developed; high-level simulators offer fast calculations for circuits described as functions such as addition, shifting, and register storage while low level simulators provide voltage and current calculations for circuits described at the transistor level. High-level simulators will not check that the adder circuit actually will work; only that if it does work, the circuit will function a certain way. To check that a designed circuit does work, low-level simulators must be used. Figure 3.3 lists several simulators for various levels of simulation of which all but two, CADDET and SUPRA, are available at the University of Saskatchewan.

	high level						low level
CLASSIFICATION :	BEHAVIORAL	LOGIC		CIRCUIT		DEVICE	PROCESS
PROGRAM :	ISPS	DABL	DED/DLS	RNL	SPICE	CADDET	SUPRA
ELEMENTS MODELED :	*adder/ subtractor *multiplier *logic functions *memory	*adder/ subtractor *multiplier *logic functions *memory	*logic gates *transistor	*MOS transistor *capacitor *resistor	*MOS transistor *bipolar transistor *diode *capacitor *resistor *inductor	*MOS transistor	*impurity profile

Figure 3-3: Simulators of IC development ordered according to the level of simulation.

Some of the simulators available for the ECG processor project are listed below along with some of their advantages and disadvantages.

3.2.1.1. ISPS

This register transfer simulator was developed to specify the behaviour of digital systems and to evaluate computer architectures [27]. ISPS compiles the user's description into machine code resulting in fast simulations, but specifying circuits at the gate level can be tedious, so ISPS is generally used to model circuits only at a high level.

ISPS was studied as one possible way to simulate the circuits for IC design, but it was not used because while descriptions of circuits made up of adders, registers, and multiplexers are readily expressed in this language, unusual functions like the square-root extractor are difficult to express.

3.2.1.2. DABL

DABL is a Pascal-like logic simulation language supplied as part of the logic simulation package for the Daisy workstation. The Daisy workstation is a computer system designed specifically for digital circuit design and IC layout. The levels of simulation provided by DABL range from the register transfer level to gate level. One sample program supplied with the Daisy models the functions of an 8085 microprocessor and is one example of this simulator's versatility. To use DABL, the circuit designer writes procedures for the required logic functions and connects these procedures through signal variables. The designer must also provide the expected time delays for each function. This is a good program to begin an IC design because the user can start with a high-level description and work towards a gate level description, but the DABL simulator was not used for the ECG processor design because it was not available until mid-way through the project.

3.2.1.3. Logician Design Editor

This is a logic simulation package for the Daisy workstation which requires the user to enter the schematic of the circuit to be simulated. Schematic entry has the advantage of documenting the design while it is being developed; updates and changes are easy to make. To investigate the usefulness of this system for IC design, both the multiplier and the square-root extractor were entered and simulated. Figure 3.4 shows the DED schematic for the multiplier as a connection of several major function blocks. Each function block, in turn, is also defined by a schematic diagram; Figure 3.5 is an expanded schematic showing a circuit at the gate level for the CONTROL unit of the multiplier. This multiplier circuit was tested by multiplying two sets of numbers, and the results are illustrated in Figure 3.6.

The complete ECG processor design was not entered using DED because the following problems suggested another approach would be better.

- The schematic entry system is suited for circuit board layout, not IC logic layout. IC layouts often take advantage of repetition of

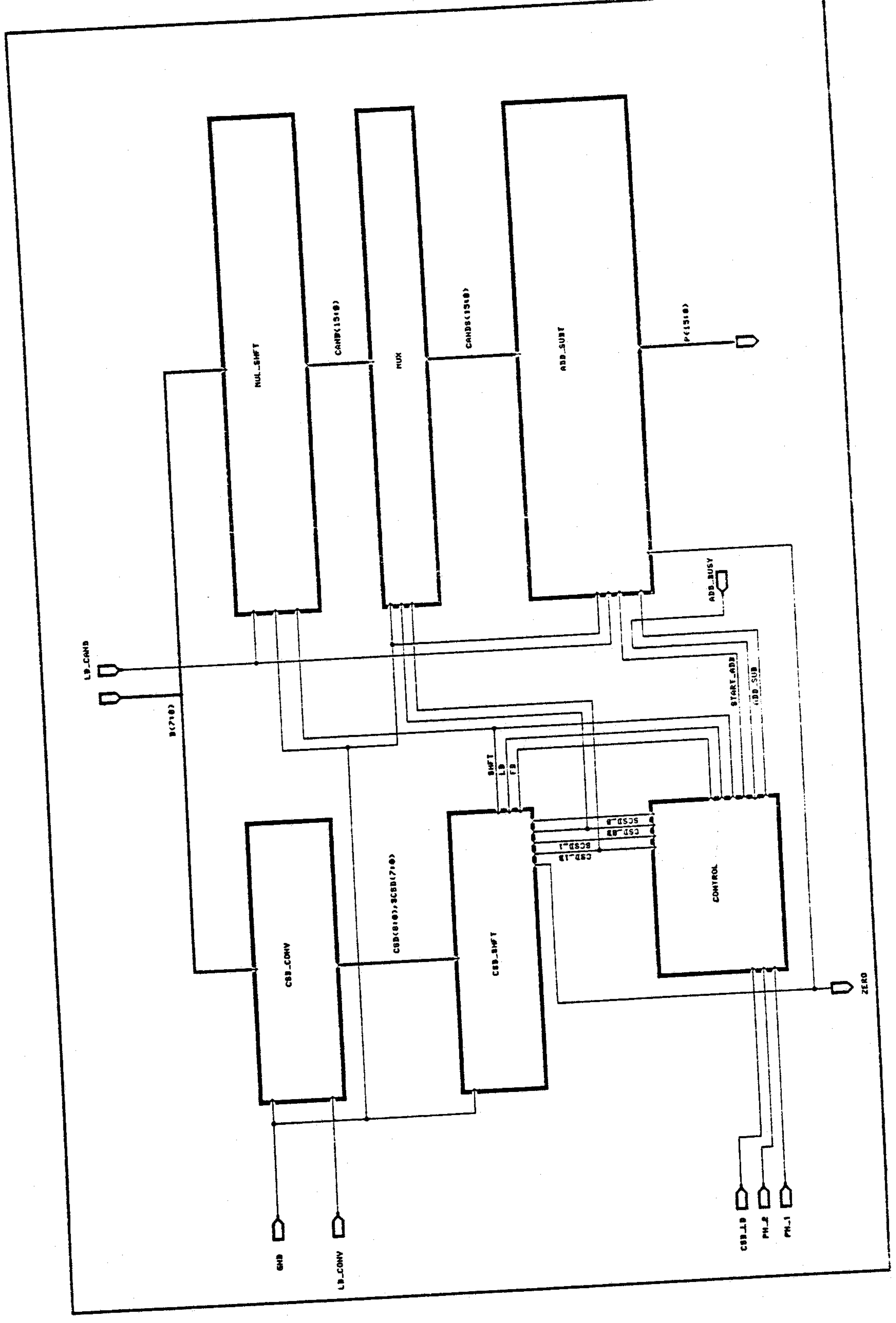


Figure 3-4: DED schematic for an 8x8 bit signed multiplier.

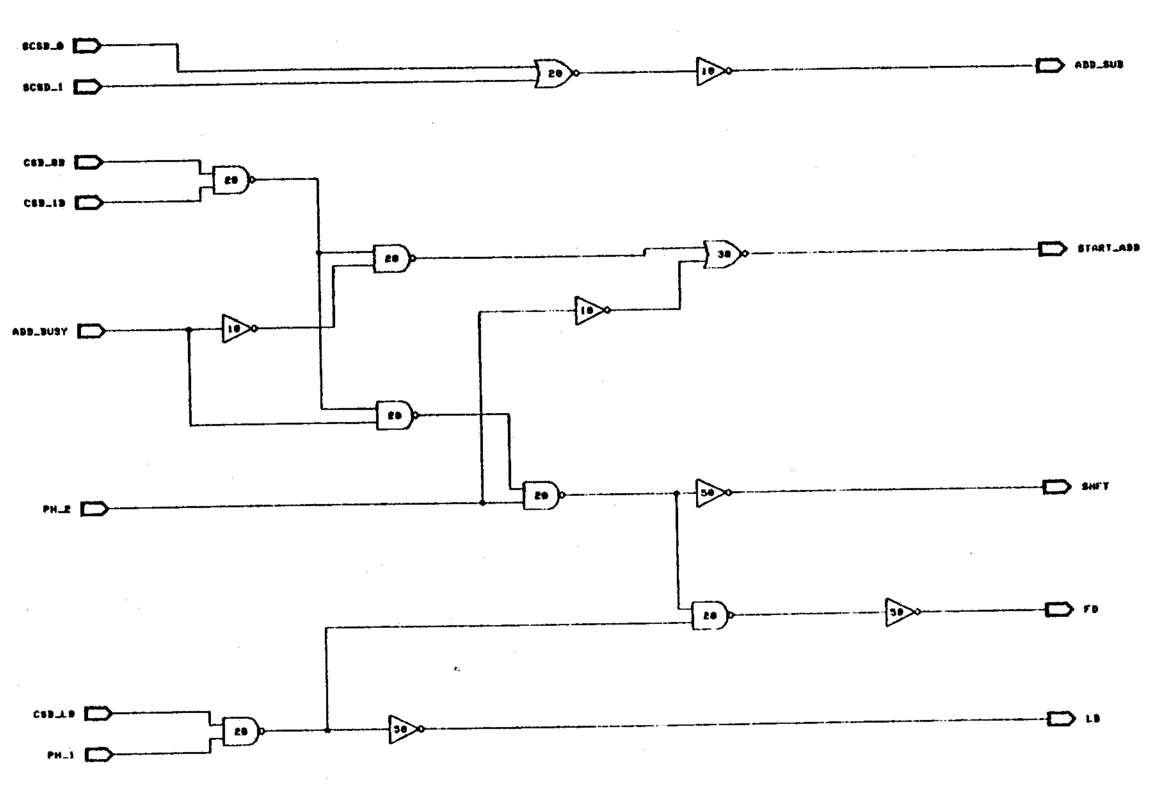


Figure 3-5: The CONTROL unit schematic for the multiplier.

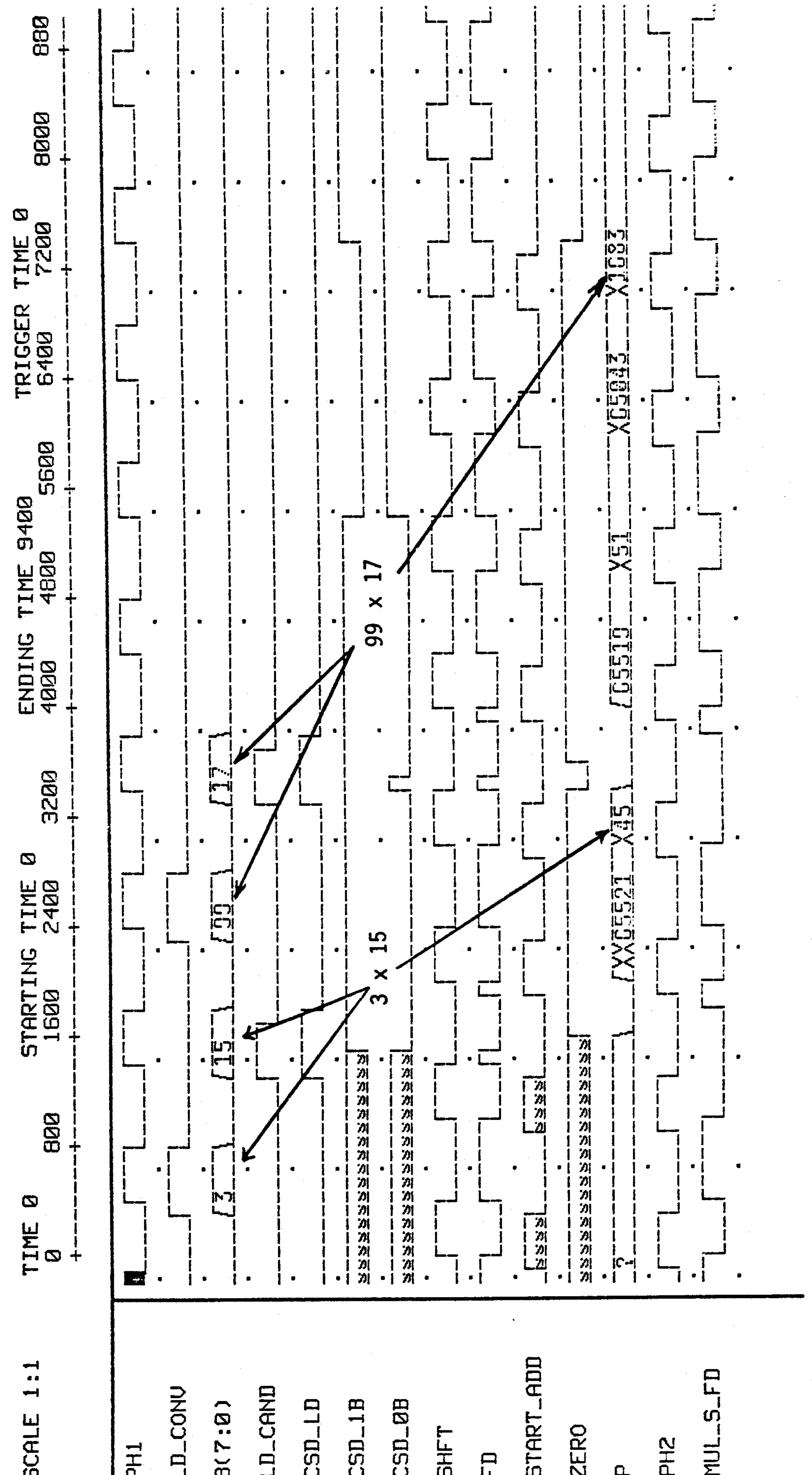


Figure 3-6: The multiplier simulation results.

logic blocks, but the schematic entry did not provide a means of quickly replicating logic blocks.

- The transistor model did not support bi-directional signal transfer. The transmission gate, a common element in CMOS IC design, is a bi-directional element. A method for modelling a bi-directional element was provided, but it was complex.
- The user must specify logic delays. This approach is reasonable for initial calculations, but does not provide a positive indication of a working circuit.
- The schematic could not be compared to the layout to ensure that the circuit represented by the layout actually corresponded to the circuit defined by the schematic diagram.

3.2.1.4. NETLIST-PRESIM-RNL

These programs are part of the University of Washington/Northwest VLSI Consortium software package [12] developed specifically for IC design. NETLIST is a LISP-like language which allows the user to specify circuits at the transistor level. NETLIST supports the definitions of macros and loops in the user's circuit description which enables it to specify large regular circuits in a compact way. Once the circuit designer has specified a circuit, the NETLIST program expands the user's macros and loops into a file which lists every transistor in the circuit. Using this file as well as a file describing the characteristics of the IC technology, PRESIM generates the binary circuit representation used by RNL for circuit simulation.

In the RNL simulation, switching delays in the circuit are calculated using RC time constants. Each node has a capacitance and each transistor is modelled by a switch in series with a resistor. When a transistor turns on to change the state of a node, the node's capacitance must discharge through the transistor's resistance. The time to change from one state to another equals RC. The RNL model has been found to be useful, but attention must be given to its limitations. One limitation is the existence of only three states for a node: 0, 1, and X. Because some circuits, like 6-transistor RAM cells, are sensitive to actual circuit voltages, RNL should not be used for their development. This does not mean RNL cannot model such circuits, but the circuit must be set up carefully so that RNL can model them. Charge sharing is another property where the RNL circuit model is limited. Normally, if nodes of different voltages are connected by a transistor turning on, time would be required for charge to flow between nodes, but in RNL, charge sharing is assumed to occur almost immediately. This has been found to be a problem in the simulation of RAM cells.

3.2.1.5. SPICE

SPICE is a general purpose electronic circuit simulator developed at the University of California. SPICE is used for circuit level simulation. Because the simulation is slow, SPICE is generally used to develop circuits with 100 transistors or less.

A modified version of SPICE called SPICE-PAC was used for the ECG processor design project. SPICE-PAC uses the same circuit models as SPICE, but while SPICE is executed as a batch job, SPICE-PAC is an interactive program. SPICE-PAC was further modified by the author to create another program called PEPPER which allowed the output data to be directed to a file for plotting purposes. This file was in a form which could be used by the system plotting service TELLAGRAF. As well, a program called SAGE was written which would plot the results of a circuit simulation in logic analyzer format. The PEPPER - SAGE programs were useful for developing small circuits.

3.2.1.6. SIMPSIN

SIMPSIN was a circuit simulator developed by the author specifically for IC design. The reason for developing this simulator was due to simulation problems encountered with SPICE; SPICE was slow with a typical simulation time of 20 minutes for a 50 transistor circuit, and in addition to this inconvenience, this version of SPICE had an occasional problem with 'convergence' or finding the solution to some circuits. It was believed that a faster simulator could be developed because SPICE modelled circuits with more accuracy than was required for IC circuits and because it was a general purpose program with more features than necessary for IC design.

To create a faster simulator, the following characteristics of IC circuits were taken advantage of:

- a limited voltage range of 0 to 5 volts, and a wider but still limited range of circuit currents and capacitances would permit integer representation of these quantities,
- only a few components, transistors, resistors, and capacitors need to be modelled,
- a look-up table for transistor characteristics would eliminate the repeated calculation of complex models.

A substantial increase was thought possible by using integer arithmetic and look-up tables.

The algorithm for this simulator was much like that for MOTIS [28]. Each node in the circuit had a capacitance, C_n, and the change in node voltage, V_n, with respect to time depended upon the net current flow, I_{net}, in or out of the node. That is,

$$\Delta V_n = \frac{\Delta t}{C_n} I_{net} \tag{3.1}$$

Δ t, C_n, and I_{net} had their units scaled so that Δ t equalled one. I_{net} was the summation of currents in all transistors connected to the node. The transistor currents, in turn, were calculated from two values, V_{DS}, the drain to source voltage, and V_G, the gate voltage, by using V_{DS} as a row pointer and V_G as a column pointer to a 50 by 50 matrix. The 50 by 50 matrix size was chosen arbitrarily.

The SIMPSIN simulation package, as set up for the the Engineering VAX 11/780 computer system, is shown in Figure 3.7. PEPPER converts the transistor parameters into characteristic curves. PEPPER uses the SPICE transistor model, but any model could be used. CONV scales the

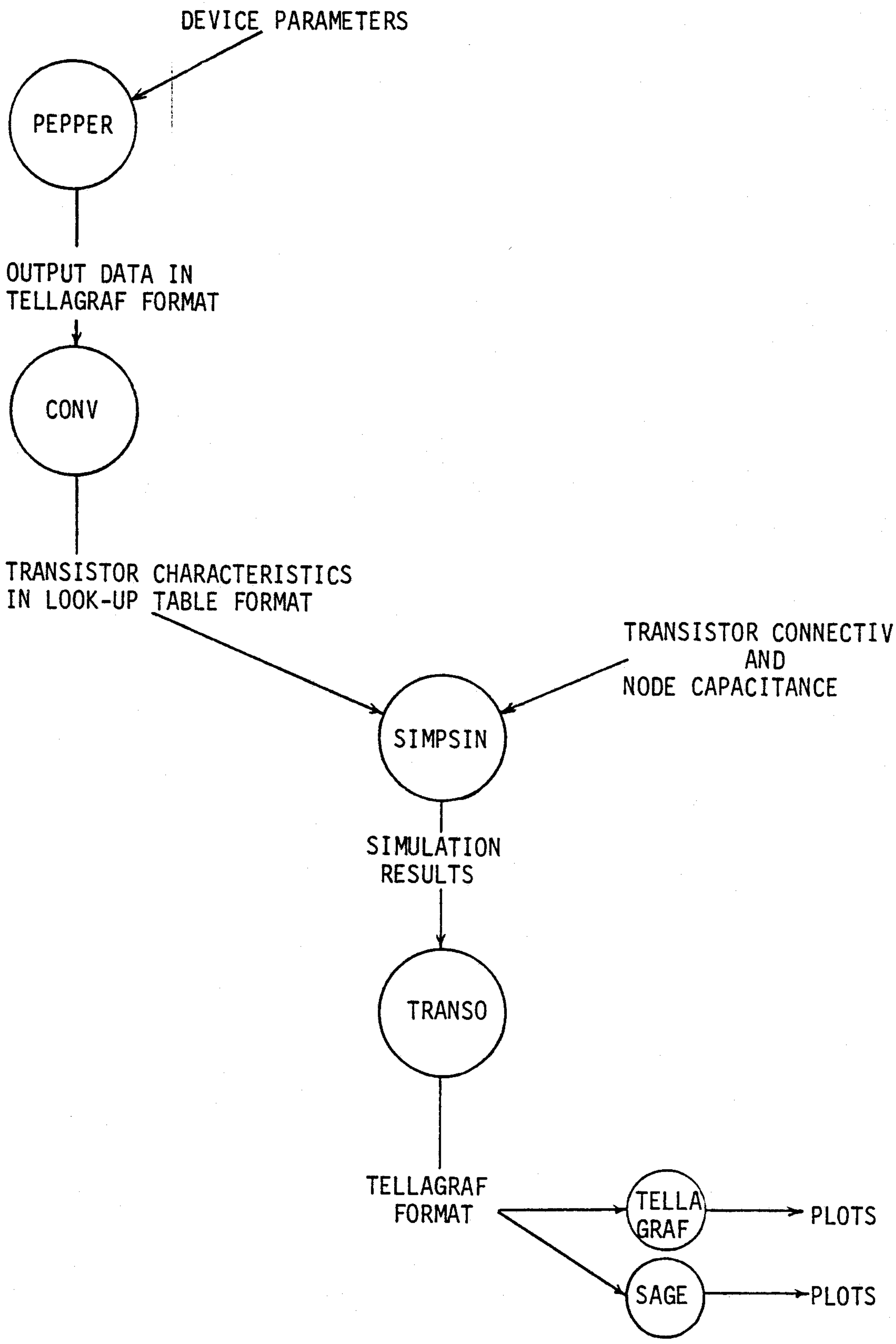

DEVICE PARAMETERS
PEPPER
OUTPUT DATA IN TELLAGRAF FORMAT
CONV
TRANSISTOR CHARACTERISTICS IN LOOK-UP TABLE FORMAT
TRANSISTOR CONNECTIVITY AND NODE CAPACITANCE
SIMPSIN
SIMULATION RESULTS
TRANSO
TELLAGRAF FORMAT
TELLA GRAF
PLOTS
SAGE
PLOTS

Figure 3-7: Programs for the SIMPSIN simulation package.

voltages and currents of the PEPPER data into units used by SIMPSIN and then writes the scaled data into a look-up table format. SIMPSIN performs the actual simulation and records the results to a file. TRANSO reformats the simulation results of the files into a TELLAGRAF plot file format which can be plotted by either TELLAGRAF or SAGE.

The results of a SIMPSIN and PEPPER (equivalent to SPICE) simulations for the circuit in Figure 3.8 are given in Figure 3.9.

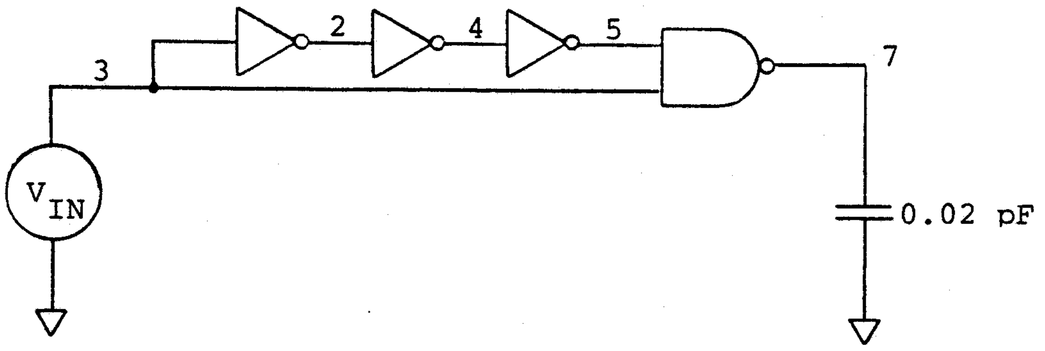
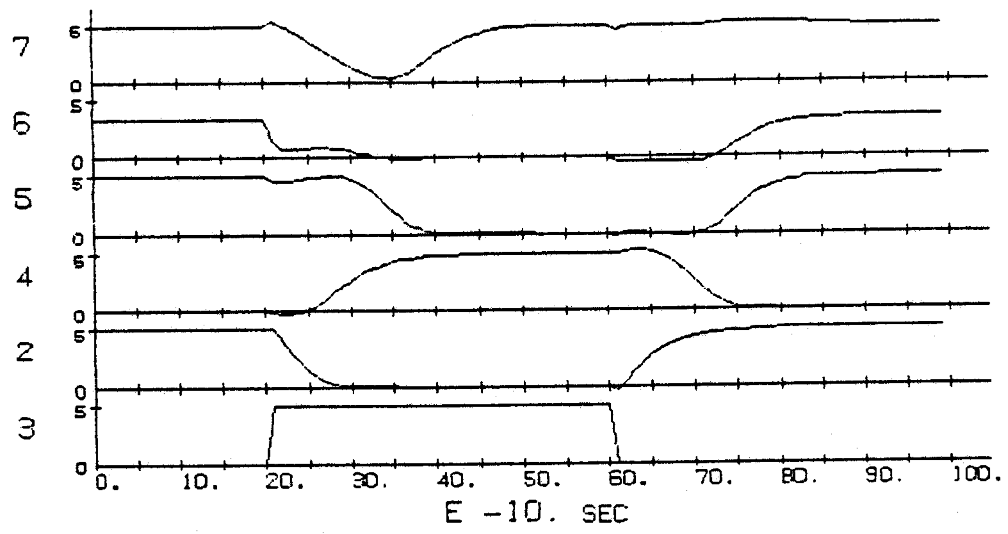


Figure 3-8: Circuit for the SIMPSIN simulation test.

Several node voltage waveforms agree quite well, but the output voltage is notably different. This may be due to a different capacitance value for the output node since SPICE calculates its node capacitances, but SIMPSIN has its node capacitances supplied by the user.

SPICE used 25 seconds of CPU time to calculate the transient response while SIMPSIN used 430 milliseconds. This is a speed improvement of 50 times, but this comparison is not fair. SIMPSIN, as written for this demonstration, modelled only one size of transistor and an extra multiplication would be needed to model different sizes of transistors. As well, SIMPSIN is not as accurate as SPICE and so the specified accuracy for a SPICE run should be reduced to allow SPICE to run faster. On the

SPICE SIMULATION RESULTS



SIMPSIN SIMULATION RESULTS

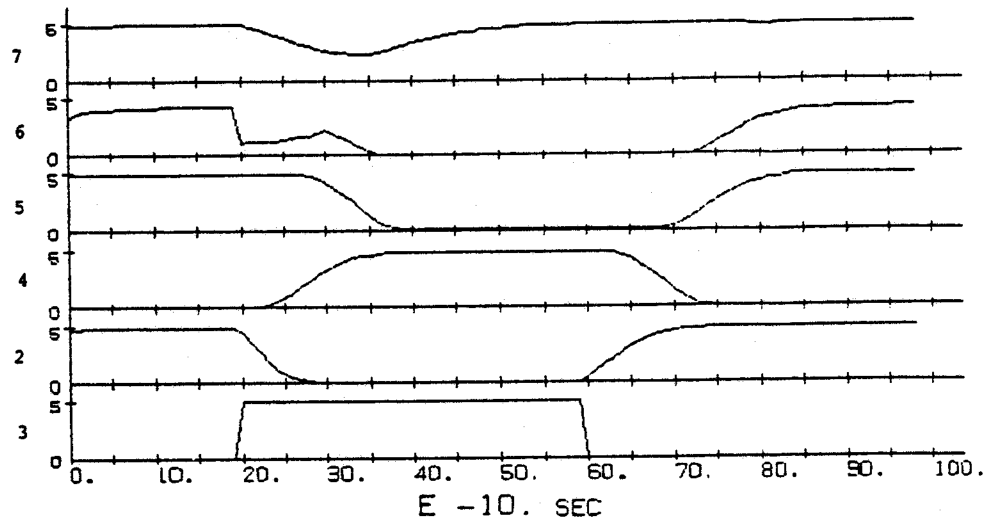


Figure 3-9: SIMPSIN and SPICE simulation results.

other hand, SIMPSIN could be modified for faster simulation. A variable time step would eliminate iterations where circuit voltages do not change quickly, and the division by C_n could be changed to a multiplication by storing $1/C_n$ rather than C_n. Writing the node voltages to a disk file after every iteration may also have slowed SIMPSIN run times, but this would depend upon whether this time is measured by the CPU timer variable 'sys$timer' and this information is not mentioned in VAX documentation.

The results of the SIMPSIN program were not as good as expected. Even with a speed improvement of 50 times, SIMPSIN could not compete with event-driven logic simulators like RNL, and for cases where accuracy is required, the proven models and flexibility of SPICE are more desirable than those for SIMPSIN. SIMPSIN also has problems with its integer quantification of node capacitance and current values; since the dynamic range of current extends from leakage current, say 0.1nA, to power supply current, say 1A, either a large word size or an exponential representation should be used, but this would eliminate one part of SIMPSIN's speed advantage. Despite these drawbacks, SIMPSIN may be useful for IC development on a microcomputer where floating-point hardware is not available.

3.2.2. Layout Software

With the exception of the PLA, the layout of the IC was completed using the MAX layout editor on the Daisy workstation. To use MAX, the user entered his/her design with a graphics tablet and observed a colour graphics display to view the layout. Once the layout was completed, the CIF_OUT program converted the graphical information into a text file that was sent to the Canadian Microelectronics Corporation for fabrication into an IC.

The DRACULA program is used to check the layout for design rule violations. A design rule check takes two and one half hours for a small design, and up to four hours for a large design. An interactive design rule checker is built into the MAX layout editor, but it is not capable of doing a complete check.

The PLA is generated from a truth table by a program called TPLA. TPLA pieces together 'tiles' or sections of layout according to values in the truth table. The Caesar layout editor is required to define the tiles used by TPLA, but unfortunately it was not available at the time the controller was being developed. Fortunately, example tiles were already defined for another 3 micron CMOS technology, and with some modifications such as changing the layer names and adding two additional layers, the resulting layout passed a CMOS3 design rule check. However, most of the geometric sizes were 20% larger than necessary because the design was not based on CMOS3 rules.

3.2.3. Layout Circuit Extractors

Once a layout is completed, a check of its functionality is desired, and to do this a layout extractor is used; it reads the geometrical description of the layout and creates an electronic circuit description at the transistor level. The circuit extractor used for the ECG processor design was MEXTRA. MEXTRA generates a circuit description which is compatible with the PRESIM-RNL simulator to allow a simulation based upon the layout. In addition to extracting transistors and their interconnections, MEXTRA also finds the gate dimensions of the extracted transistors and finds the capacitance of each node due to layers attached to the node. These features make the simulation based upon the layout more accurate than a simulation based up on a NETLIST circuit description.

Other capabilities which could increase the accuracy of simulations would be extractors which included the resistance of long wires (especially polysilicon) and internode capacitance (caused by overlapping layers) although measurements taken on the CMOS3 process did not show measurable cross-talk between long wires or overlapping metal and polysilicon lines.

3.2.4. The University of Saskatchewan Design System

Figure 3.10 is a description of the IC design environment used for this project. Most of the programs have been mentioned previously, but those which have not are briefly described here:

- GEN_CONTROL - provides a fast way to specify the input and output signal to the RNL simulator.
- GEN_TIME - specification of the input signals to RNL can be time-consuming and difficult to read. GEN_TIME provides a better way to specify these signals, especially repetitive signals.
- SIM2SPICE - translates the transistor list of a .sim file to a SPICE compatible transistor list.
- STREAMOUT - converts the layout representation used by MAX into a format called GDS.
- STREAMIN - converts the GDS layout representation into the format used by MAX.
- CIF_OUT - converts the MAX layout representation into CIF format.
- CIF_IN - converts the CIF layout representation into the MAX format.
- CIF2KIC - converts the CIF layout representation into the KIC format.
- KIC2CIF - converts the KIC layout representation into CIF format.
- KIC - a layout editor available on the engineering VAX 11/780 computer.
- PEG - a program which translates a user's high-level description of a Moore machine into a set of logic equations.
- EQNTOTT - generates a truth-table from a set of logic equations.
- PLA2NET - generates a transistor netlist for a PLA from a truth-table.

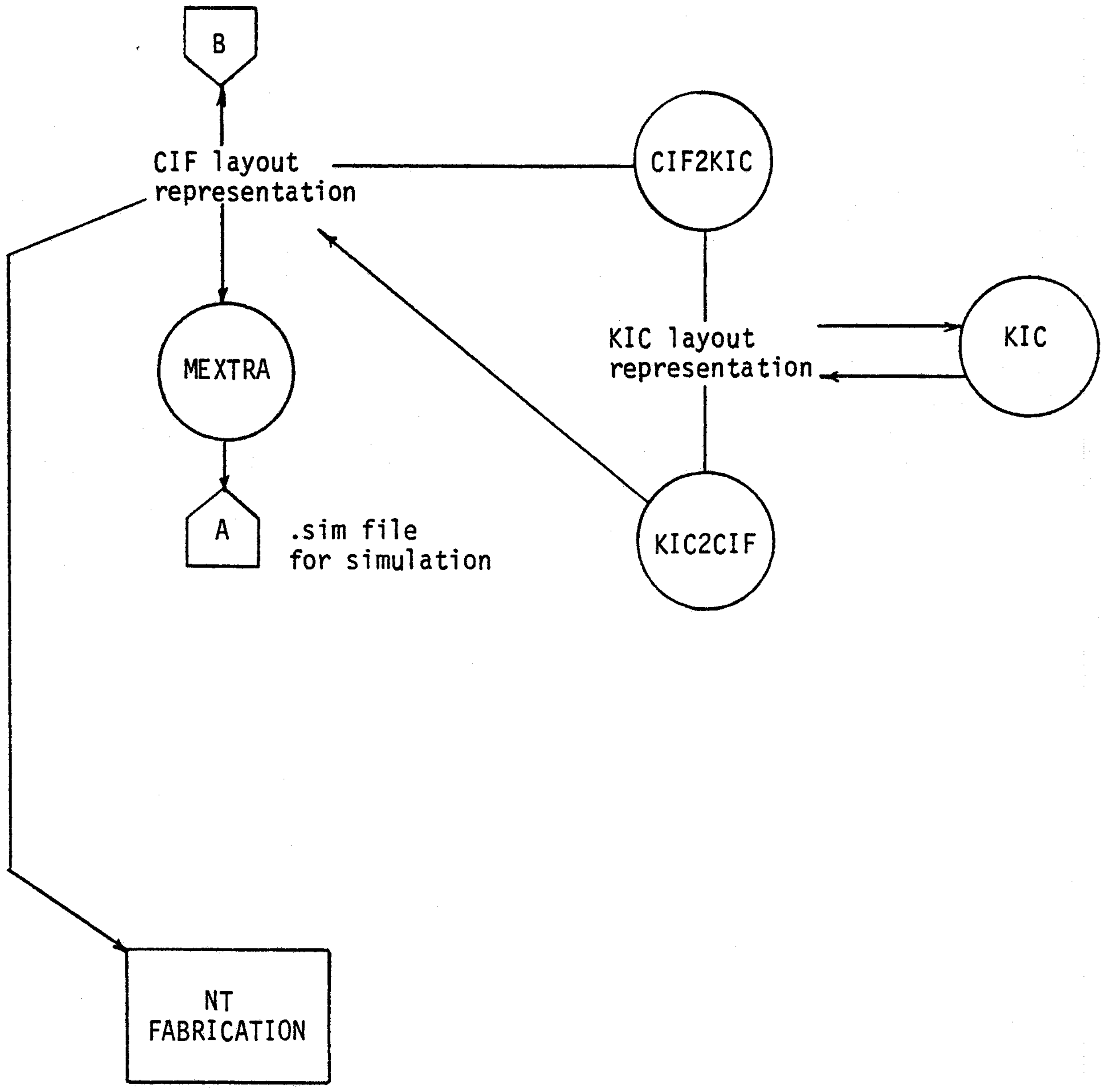
Not all programs reside on the same computer; MAX is part of the Daisy workstation, NETLIST, PRESIM, and RNL are run on a MicroVAX II, and PEPPER is run on the Engineering department's VAX 11/780. The transfer of data between the MicroVAX and the VAX 11/780 is possible through an Ethernet communication link while data transferred from the Daisy to the VAX 11/780 is performed using magnetic tape.

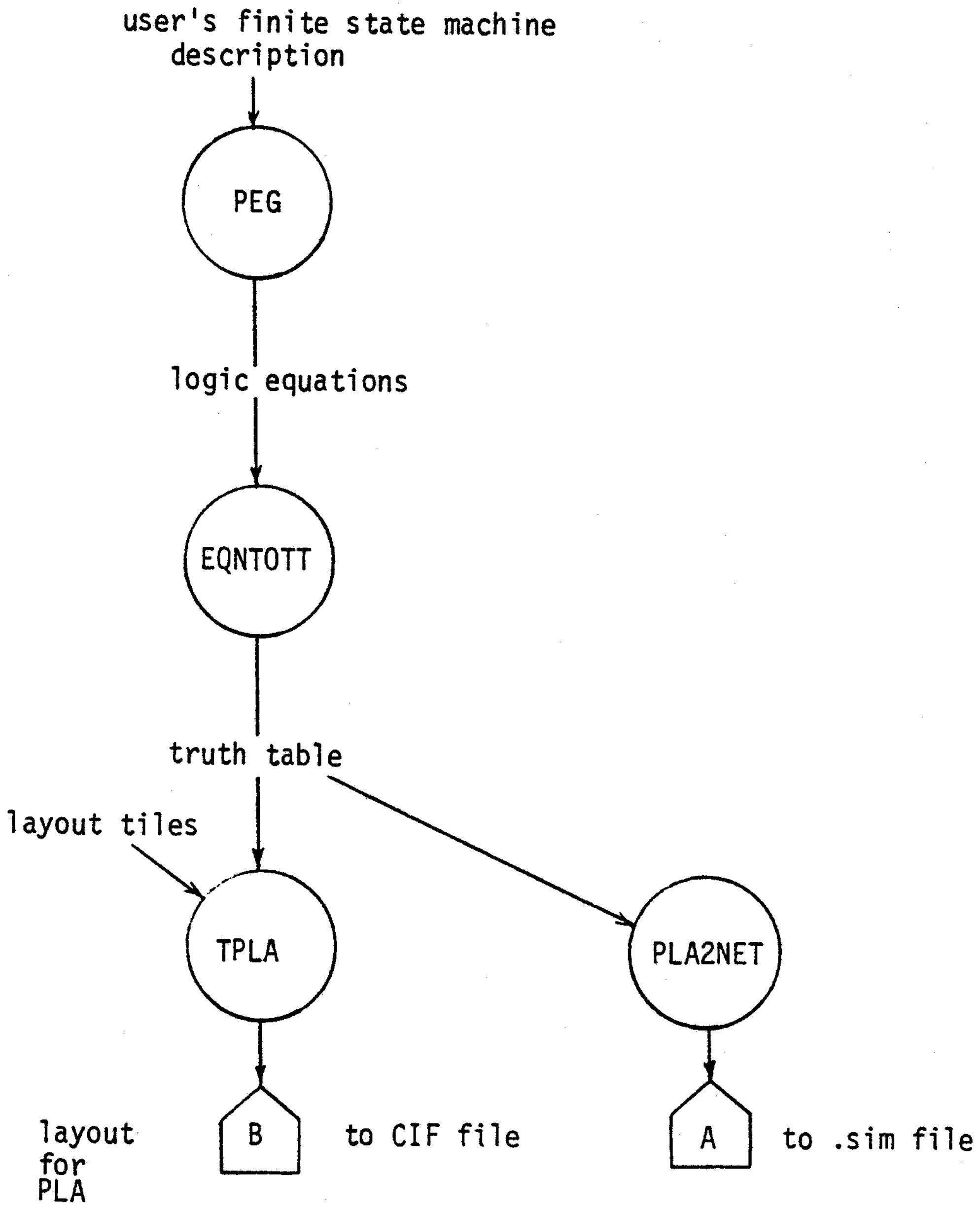
3.3. Summary

This chapter covered the techniques used to manage the complexity of IC design and some of the software used for this project.

Two methods were presented which allowed a designer to develop a large IC circuit; they were hierarchical abstraction and a global two-phase clock signal. Hierarchical abstraction reduces the amount of information the designer needs to comprehend at one time by allowing him to concentrate on only the most important aspects of IC design. The global two-phase clock signal is a simple method of organizing on-chip data communication.

The final section of this chapter listed the software available for this IC design. Examples of high-level and low-level simulators were given, and a program for circuit simulation along with the simulation results was presented. Finally, the organization of the software used for the design of the ECG processor was depicted in diagram form; 23 separate programs for design rule checking, logic simulation, circuit simulation, and layout generation are part of the IC design package.


B
CIF layout representation
CIF2KIC
MEXTRA
KIC layout representation
KIC
A
.sim file for simulation
KIC2CIF
NT FABRICATION


user's finite state machine description
PEG
logic equations
EQNTOTT
truth table
layout tiles
TPLA
PLA2NET
layout for PLA
B
to CIF file
A
to .sim file

Chapter 4
The Integrated Circuit Implementation of the Morphology Recognition Unit

This chapter presents the development of an IC for the morphology recognition algorithm described in Chapter 2. The development is presented here in a hierarchical fashion beginning with a description of the algorithm to be implemented, and then describing, at the register transfer level, the functions necessary to carry out the morphology recognition algorithm. The complete circuit description at the transistor level is left for Chapter 5 because it is not necessary for understanding the operation of the ECG processor and because the circuit simulation results can be discussed in more detail there.

The first section of this chapter presents the architecture of the ECG processor including a description of the actual algorithm to be implemented as well as a description of how the ECG processor would be connected to an ECG analysis system. The next section reveals how each of the functional blocks required by the ECG processor perform their intended function. Following a description of the ECG processor operation, two sections, one describing support hardware and the other describing support software, are presented to clarify how the ECG processor is to be used in an ECG analysis system. The last section is important for the production of the ECG processor; it tells what test patterns should be used in the production testing of the IC.

4.1. Architecture of the ECG Processor

The design of the ECG processor begins with this section. First, the ECG morphology recognition algorithm is simplified to shorten the design time and to increase the probability of a successful IC design. This is followed by a discussion of the interface between the ECG processor and the ECG analysis system.

4.1.1. The Similarity Measure Algorithm

For a first time design, the complete morphology recognition algorithm is too complex to be implemented on a single IC, and therefore, as the first step in the design, the functions to be carried out by the ECG processor are simplified as much as possible. The simplification is accomplished by moving seldom performed calculations and operations from the ECG processor to software whenever possible. For example, the morphology recognition algorithm can be divided into two operations: the mapping function where the ECG data and its Hilbert transform are mapped into a recognition matrix, and the Bhattacharyya distance calculations for comparing the recognition matrices. Each of these functions could be implemented in either hardware or software, but the similarity calculations are more complex than the mapping calculations and are expected to be the bottleneck in the morphology recognition calculations. Therefore, the IC design will concentrate on the Bhattacharyya distance function while the mapping function is treated as a separate function that is performed by the host computer.

Calculating the distance between two recognition matrices using the complete Bhattacharyya distance function is still a complex task. The task was reduced in complexity in Chapter 2 by removing the $-\log()$ function to produce a similarity measure based upon the Bhattacharyya distance:

$$\text{max: } Q = \sum_{i,j} \frac{\sqrt{A_{i,j}}\sqrt{B_{i,j}}}{\sqrt{A_T B_T}} \tag{4.1}$$

which can be further simplified by noting that the division by $\sqrt{A_T B_T}$ does not have to be part of the summation function. The ECG processor could compute the value after the Σ in

$$\max:\ Q = \frac{1}{\sqrt{A_T B_T}} \sum_{i,j} \sqrt{A_{i,j}} \sqrt{B_{i,j}} \qquad (4.2)$$

leaving the value $(A_T\ B_T)^{-1/2}$ to be computed by the host computer. The reason for having the host computer do this calculation is because it would add considerably to the IC complexity to perform the required divide function which otherwise would not have to be included, and since the divide operation is calculated relatively infrequently, once every heartbeat, there would be no substantial speed increase even if it were made part of the ECG processor.

4.1.2. The ECG Processor Interface

So far, the function of the ECG processor has been simplified to $\sum^{i,j} \sqrt{A_{i,j}} \sqrt{B_{i,j}}$, but the interface between the ECG processor and the device using the ECG processor has yet to be defined. Through this interface must pass the recognition matrix data for the ECG processor calculations and the similarity measures generated by the ECG processor.

Several options were considered for the host computer - ECG processor interface. Two options were the use of a standard serial port like the RS-232, or the use of a parallel port. The advantage of using such interfaces would be the ease with which the ECG processor could be connected to a host computer. However, the rate of data transfer could be too high for a serial or parallel port. A preliminary check was made by assuming that the memory cycle time was the limiting factor in the similarity calculations. With this assumption, to compare two matrices, 2x961 or 1922 memory accesses would be made with, say, a conservative memory cycle time of 500 ns, and the total calculation time would be about 1 ms. Two more assumptions, eight comparisons for each recorded heartbeat and a heartbeat every second, would give a preliminary number of ECG leads at $\frac{1s}{8ms} = 125$ leads. With a sample rate of 500 Hz per lead, the rate for ECG data transfer would be 62,500 bytes per second, and the rate of recognition matrix data transfer would be 120,000 bytes per second. These rates would be too high for a serial port, but a parallel port remains one possible alternative.

A third option for the ECG processor interface was a shared memory configuration. In this case, the host computer which created the recognition matrices would write its data to a memory to which the ECG processor also had access. The memory for this approach would have to be external to the ECG processor because the maximum on-chip data storage (based upon a 20μ x 20μ memory cell) is 600 bytes and would not be sufficient for even one recognition matrix. The main advantage of the shared memory approach would be the elimination of the time needed to transfer information, but this would come at the expense of an interface that would be more complex and more dependent on the type of host computer.

The interface selected for the ECG processor is based upon the shared memory approach. This approach offers the highest morphology calculation rate because the time necessary to transfer the matrix data between the ECG processor and the host computer is eliminated. One problem with the shared memory approach, in addition to the previously mentioned problem of a complicated interface at the hardware level, is that the IC is made more complex because a memory address unit must be included. However, by giving the ECG processor direct access to the recognition matrix data, it may also perform other functions that would not be possible through a parallel port. For example, the elements of an unknown matrix must be reset to zero before starting the recording of a new heartbeat. This highly repetitive and simple task can now be moved from the host computer to the ECG processor. Another task which can now be executed by the ECG processor is the updating of the reference matrices. This task involves adding the elements of a newly classified unknown matrix to a reference matrix to improve the definition of the reference matrix and to account for slight changes of the ECG wave form with time as mentioned in Chapter 2.

The shared memory arrangement could be approached in one of four ways, each with its own advantages and disadvantages.

Shared memory approach 1. One way to provide shared memory would be to use a switched memory block for each recognition matrix, that is, several 1K memory chips could have their address and data lines multiplexed between the host computer and the ECG processor. The memory blocks would be connected to the host computer while a recognition matrix was being constructed, and when finished, the memory address and data lines would be switched to the ECG processor for the similarity calculations. This approach was not considered further because each ECG lead would require a switched memory block resulting in a circuit with several small 8K bit memory chips and their associated switching circuits.

Shared memory approach 2. Another method for sharing memory would use time division multiplexing where the memory would be available to the ECG processor every other clock cycle and available to the host computer on the alternate clock cycles.

Shared memory approach 3. Yet another technique for providing shared memory would be to multiplex the address and data lines to a large, say 256K, block of memory and allow access to the memory as each device made a request. Some type of arbitration unit would be necessary to resolve conflicts when both devices tried to access memory at the same time, and there would be a performance penalty since if the memory were being accessed by one system, the other system would have to wait. One IC is available which performs these functions. It is the the Intel 8207, and it combines the refresh functions of a dynamic RAM controller with those of an asynchronous dual-port arbiter and multiplexer [29].

Shared memory approach 4. The least complex configuration would be based upon dual-port RAM. This RAM has two sets of address and data lines and memory accesses may be made independently from either set (there could be some restrictions when writing to the same location). The problem with dual-port RAM would be that it is currently limited to small (8K-bit) sizes and is expensive, but this could change as 256K bit dual-port RAMs have been successfully fabricated [30].

4.1.3. Data Structures

Once the shared memory configuration interface has been selected, the format of data storage or a memory map must be defined. A first step in this definition is the selection of a word size for the ECG processor. The word size is based upon the most common unit of data which, in this case, would be the element of the recognition matrix. In [5], the matrix elements were stored as floating point variables, but the word size selected for this project was 8 bits because:

- development of the ECG processor would be easier for 8-bits since the design would be smaller,
- 8-bits is a common word size for many computer systems,
- as a total of 500 sample points typically make up a recognition matrix, the largest possible element value should be about 1000 which would occur only for a featureless signal. A time varying signal would spread this total over several elements. Therefore, the limit of 255 provided by an 8-bit word appears to be a reasonable choice.

The choice of an 8-bit word is a compromise. While the last point mentioned above is true for the unknown matrices, the reference matrices are the sum of several recorded heartbeats, and therefore when adding data to a reference matrix, care must be taken not to exceed 255. Another factor in the consideration of word size is the required precision of the similarity measures. As the measure in this design is the sum of several 8-bit values, its precision should be greater than 8-bits. However, no experiments were conducted which tried to determine how the precision of the similarity measure affected the accuracy of the ECG classification program, and since there is no evidence that a higher precision is necessary, this leaves us to assume that using 8-bits for a matrix element is sufficient. Once experience with actual ECG data is acquired, the word size may be changed for future versions of the chip.

The next point to be considered is how the two types of recognition matrices, the unknown or unclassified matrix, and the reference matrix, are to be stored in memory. The memory map for the proposed ECG processor is given in Figure 4.1.

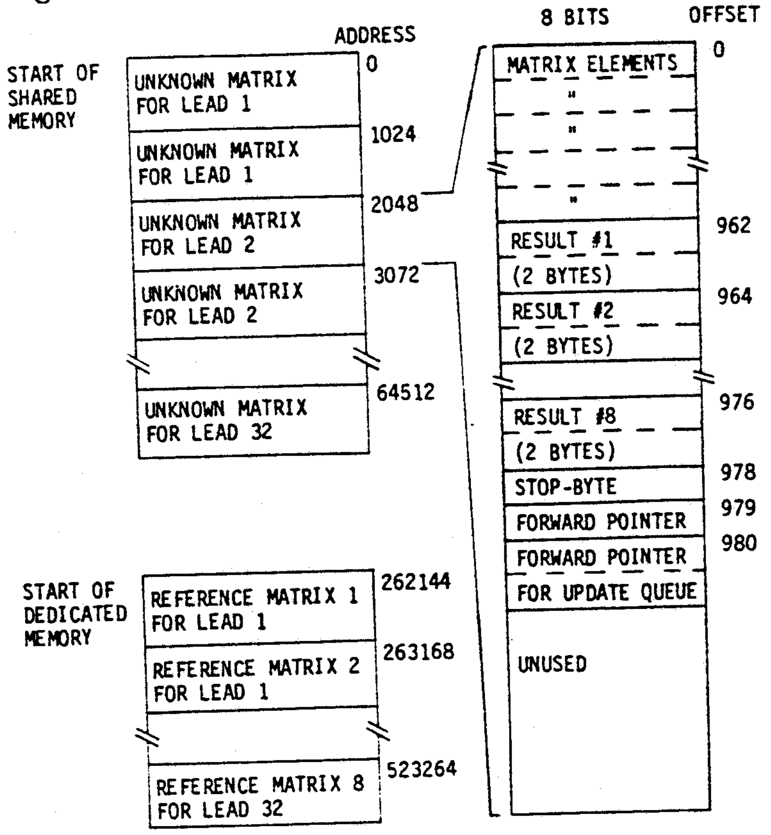

ADDRESS
START OF SHARED MEMORY
UNKNOWN MATRIX FOR LEAD 1
0
UNKNOWN MATRIX FOR LEAD 1
1024
UNKNOWN MATRIX FOR LEAD 2
2048
UNKNOWN MATRIX FOR LEAD 2
3072
UNKNOWN MATRIX FOR LEAD 32
64512
START OF DEDICATED MEMORY
REFERENCE MATRIX 1 FOR LEAD 1
262144
REFERENCE MATRIX 2 FOR LEAD 1
263168
REFERENCE MATRIX 8 FOR LEAD 32
523264
8 BITS
OFFSET
MATRIX ELEMENTS
0
"
RESULT #1 (2 BYTES)
962
RESULT #2 (2 BYTES)
964
RESULT #8 (2 BYTES)
976
STOP-BYTE
978
FORWARD POINTER
979
FORWARD POINTER FOR UPDATE QUEUE
980
UNUSED

Figure 4-1: The ECG Processor memory map.

This arrangement was developed during the design of the ECG processor and so reasons for the layout of this memory map do not follow from the information presented so far. Each matrix uses a full 1024 byte block even though each matrix has only 961 elements. The remaining 63 bytes in shared memory are used to transfer information between the host computer and the ECG processor. The reason for this choice is a considerable simplification of the address unit. By keeping the matrix data aligned on 1024 byte borders, a 10-bit register and a 10-bit counter are eliminated. Whether this is an economically good choice is difficult to determine without knowing the fabrication cost of the circuits, but for a first time design, it was felt this simplification was worthwhile.

The amount of memory required for data storage depends upon the number of ECG leads being analyzed and the number of matrices stored per lead. Each lead of ECG data requires 10 recognition matrices to be stored in memory. For each lead, there are two unknown matrices, one which is being created as the ECG signal is being digitized, and the other which is in the process of being classified by the ECG processor. As well, there are eight reference matrices associated with each lead. The suggested minimum number of reference patterns per lead is recommended in [5] to be ten, but this was reduced to eight because it simplifies the construction of the ECG processor address unit and reduces the amount of on-chip storage. This is not a serious limitation, though, because by creating two identical unknown matrices from one lead of ECG data, 16 reference patterns are effectively available, and this may be extended to 24 reference patterns by creating 3 identical matrices.

The total number of leads which can be analyzed has been found to depend upon the speed of the host computer. For this reason, the memory addressing capability of the ECG processor has been limited to the ability to access data for 32 leads. The ECG processor would be capable of processing more leads if its address bus were extended, but the host computer which must map the ECG data into the recognition matrices would not be able to map data fast enough to take advantage of this.

Because the reference matrices are not modified as often as the unknown matrices, reference matrices do not need to be in shared memory. As shared memory is more expensive that standard memory, the reference matrices are to be stored in a memory called dedicated RAM that is accessible only by the ECG processor. Besides lowering memory costs, this also reduces the address range of the shared memory block. This can be important because otherwise the address range would be large, about 256K bytes for 30 leads, and it may be difficult to design into some computer systems.

The right half of Figure 4.1 shows the details of one 1024-byte shared memory segment. The first 961 elements are the values of the matrix elements. How the matrix elements are ordered within this 961 byte block does not affect the result of the similarity calculations and therefore may be done in a way that is most convenient to the mapping algorithm. Following this are 16 bytes which hold the results of the ECG processor's calculations. After the unknown matrix has been compared to eight reference matrices, the resulting 8 16-bit similarity measures are written by the ECG processor to these locations for use by the host computer. The next byte, called the 'stop-byte', is used to synchronize the ECG processor to the host computer. When a zero is stored in this location and the ECG processor reads this zero value, it will continuously poll this 'stop-byte' until a non-zero value is detected. When the host computer is ready for the ECG processor to continue, the host computer writes a non-zero value to this location, and when the non-zero value is detected, the ECG processor resets the 'stop-byte' back to zero and continues reading sequentially from the next shared memory location. The next four locations are forward pointers which identify the address of the next unknown matrix to be classified. Once these pointers are loaded into the ECG processor's address registers, the ECG processor 'jumps' to the appropriate memory segment to begin similarity calculations for another matrix.

The recognition matrices in shared memory form a linked-list with the forward pointer being located at the end of each matrix. The ECG processor reads the data from one matrix, writes its calculated values back to memory, polls the 'stop-bytes', and then reads the address for the next matrix from the forward pointers at the end of the matrix. By using the 'stop-byte' and the forward pointers, the ECG processor is completely controlled by the host computer through shared memory. The host computer does not have any indication of what the ECG processor is doing at any given moment, and for this reason, the host computer must keep a list of matrices being classified. It must also poll the 'stop-byte' locations to determine when the similarity calculations are complete. This list of matrices may be thought of as a queue. A matrix for classification is added to the queue by putting its address at the end the matrix which was most recently placed on the queue and changing the 'stop-byte' to a non-zero value. By checking the 'stop-byte' for a zero value of matrices already placed in the queue, the host computer can determine which matrices have been classified.

4.1.4. ECG Processor Timing

The ECG processor is based upon a 2.5 MHz two-phase non-overlapping clock. The frequency of 2.5 MHz was chosen as a target frequency because the clock period would approximately correspond to a memory access cycle time. Depending on the memory cycle time and the shared memory arrangement used, the frequency could be changed. For example, if the time multiplexed shared memory arrangement was used, the clock period might be increased to twice the memory cycle time. In this arrangement, the shared memory would be available to the ECG processor for one half of a clock cycle, and available to the host computer for the other half.

The maximum clock frequency may be higher than 2.5 MHz since, as mentioned in Chapter 3, the period needs only to be long enough for all logic operations to finish. The function which will take the longest time to complete in the ECG processor will be the accumulation function. For this operation, a read operation is made from the register file, the data added to the multiplier output, and the result written back to the register file all within one clock cycle. An estimate of the maximum clock frequency can then be made by using data presented in the next chapter. The read and write time to the register file is 60ns, and the add time is approximately 52ns giving a total time to completion of 172ns. This is the minimum clock period which equals a clock frequency of 5.8MHz.

The memory read/write accesses are similar to those for the Motorola 6800 and MOS 6502 microprocessors except that the roles of ϕ_1 and ϕ_2 have been interchanged. As shown in Figure 4.2, the address lines begin to change on the leading edge of ϕ_2, and depending on the loading, they are guaranteed to be valid t_{AV} after this edge.

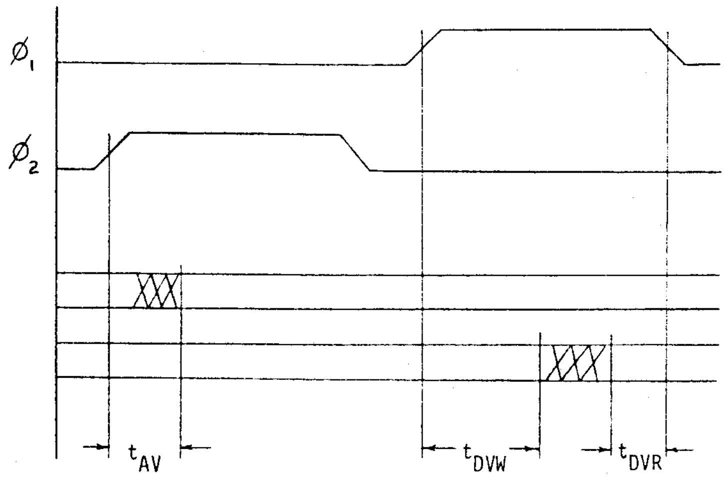


Figure 4-2: Memory read/write timing diagram.

For a read cycle, the data must be valid t_{DVR} before the falling edge of ϕ_1, and for a write cycle, the data is made available t_{DVW} after the rising edge of ϕ_1. From an RNL simulation using 10pF loads on the IC pins, the t_{AV} propagation delay time is 29ns, the t_{DVW} propagation delay time is 57ns, and the t_{DVR} set-up time is 17ns.

4.1.5. Summary of the ECG Processor Functions

The flowchart in Figure 4.3 outlines the functions performed by the ECG processor. After the ECG processor is released from its reset state, it writes a zero to the 'stop-byte' and continues to poll this location for a non-zero value. This gives the host computer time to set up the matrix data and the forward pointers following the 'stop-byte'. It should be noted that the 'stop-byte' is polled only every other cycle to allow the host computer access to shared memory. Once the host computer writes a non-zero value to this location, the ECG processor resets it back to zero and reads the address of the next matrix to be classified. Then the next few steps are repeated 961 times. For each cycle through this loop, an element is read from the unknown matrix. If this value is non-zero, then this cycle is followed by eight read cycles from dedicated memory to retrieve the corresponding element from each of reference matrices, otherwise, if this value is equal to zero, then only one read is made from dedicated memory. The value from this one read is discarded and is only necessary because, as the unit was designed, one cycle is needed to detect a non-zero value. Following this are the cycles taken up by the update functions; these cycles are always taken whether or not the update function performs any useful function. Functions carried out during the update cycles include resetting the elements of unknown matrices back to zero after they have been classified, and adding the elements of a newly recorded matrix to a reference matrix. After this loop has been completed 961 times, the ECG processor completes the similarity calculations, writes its results to shared memory, and the begins the process over by polling the next 'stop-byte'.

4.2. Function Blocks of the ECG Processor

The algorithm for the ECG processor has been presented and now in this section, the function blocks necessary to carry out the algorithm are described. A general block diagram for the ECG processor is presented in Figure 4.4. The main components of the ECG processor have been identified

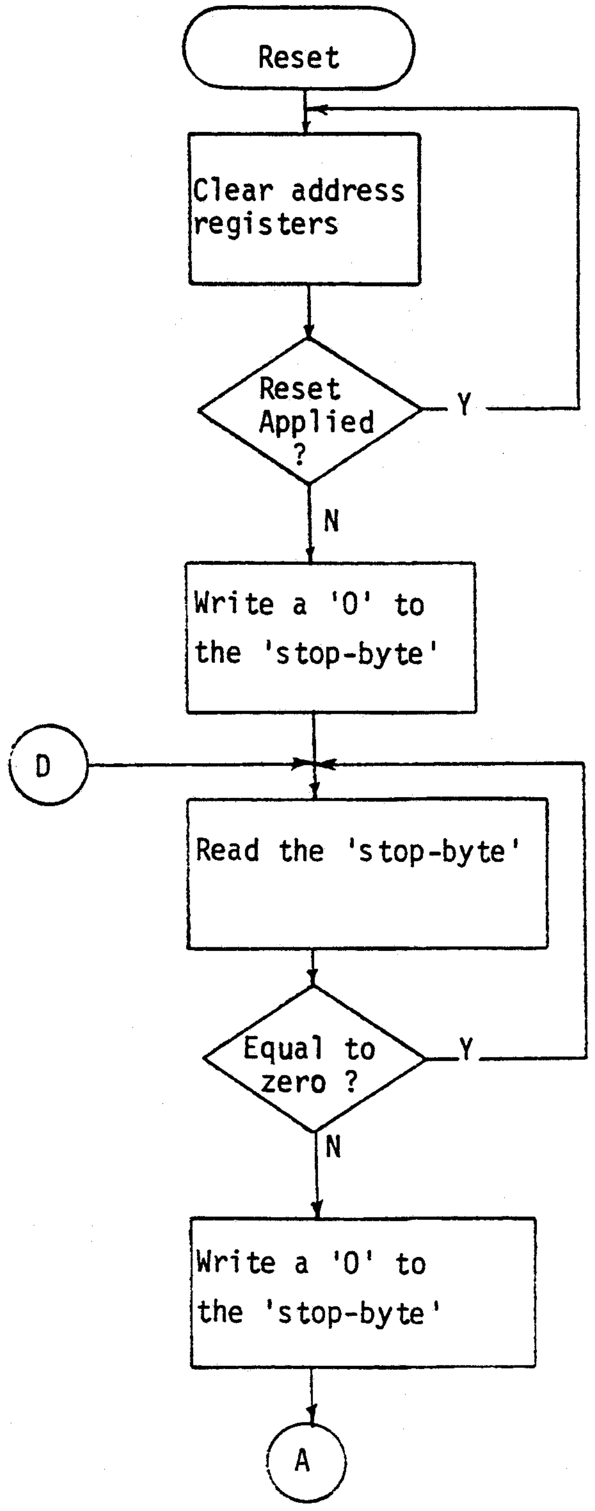


Figure 4-3: Flowchart of the ECG Processor algorithm.

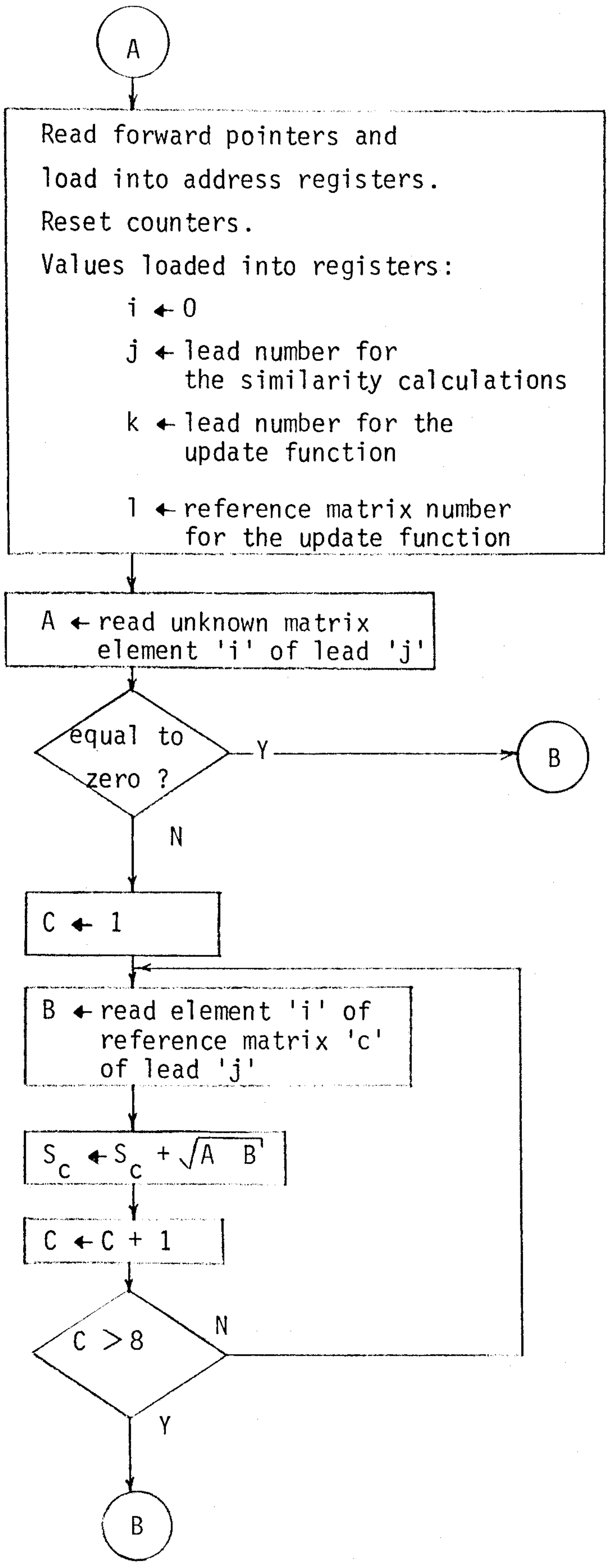

A
Read forward pointers and
load into address registers.
Reset counters.
Values loaded into registers:
i ← 0
j ← lead number for the similarity calculations
k ← lead number for the update function
l ← reference matrix number for the update function
A ← read unknown matrix element 'i' of lead 'j'
equal to zero ?
Y
B
N
C ← 1
B ← read element 'i' of reference matrix 'c' of lead 'j'
$S_c \leftarrow S_c + \sqrt{A\ \ B}$
C ← C + 1
C > 8
N
Y
B

Figure 4-3 continued

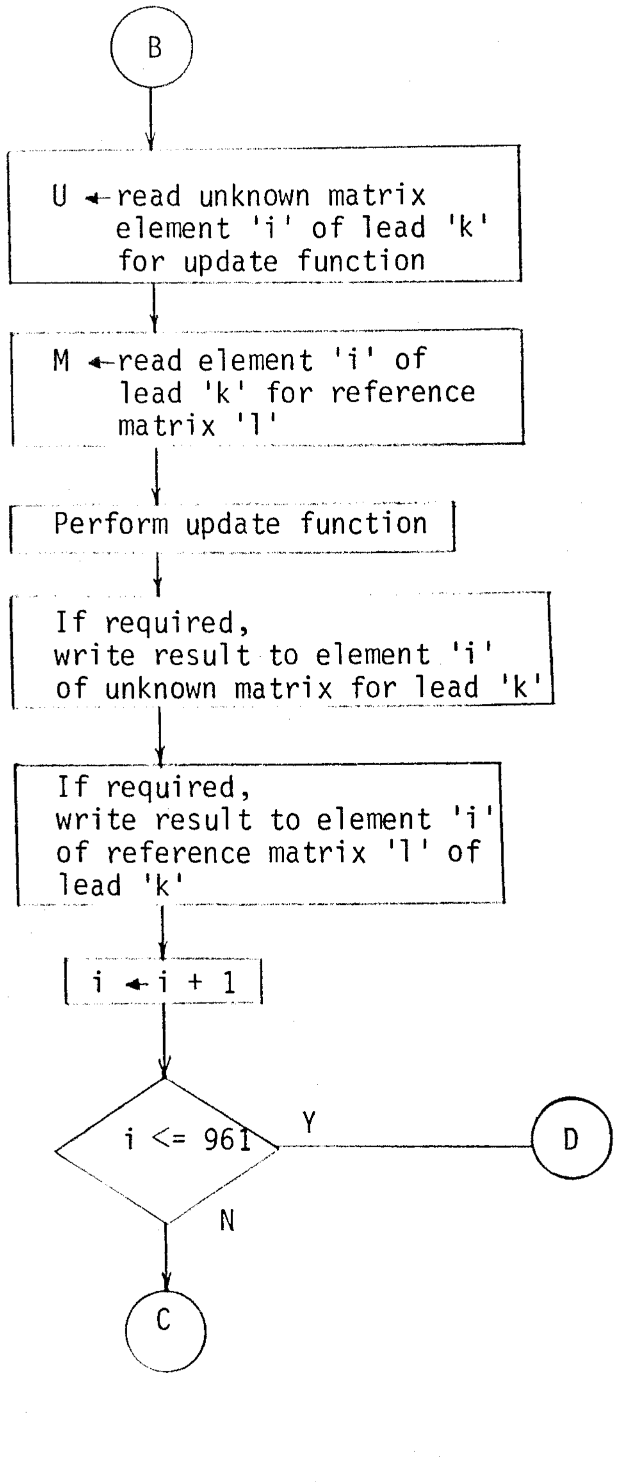

B
U ← read unknown matrix element 'i' of lead 'k' for update function
M ← read element 'i' of lead 'k' for reference matrix 'l'
Perform update function
If required, write result to element 'i' of unknown matrix for lead 'k'
If required, write result to element 'i' of reference matrix 'l' of lead 'k'
i ← i + 1
i <= 961
Y
D
N
C

Figure 4-3 continued

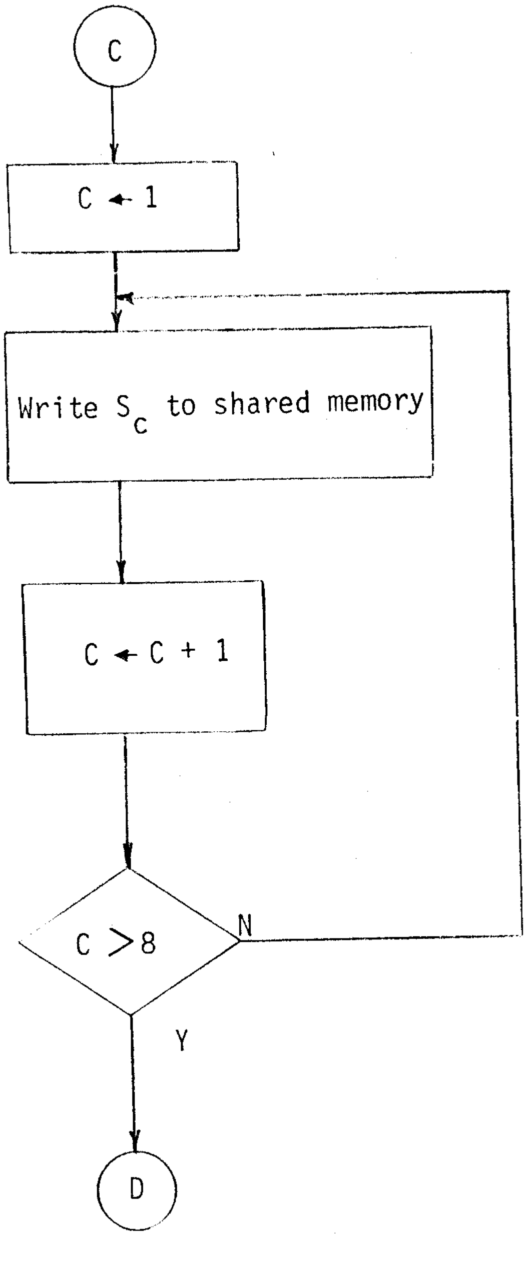


Figure 4-3 continued

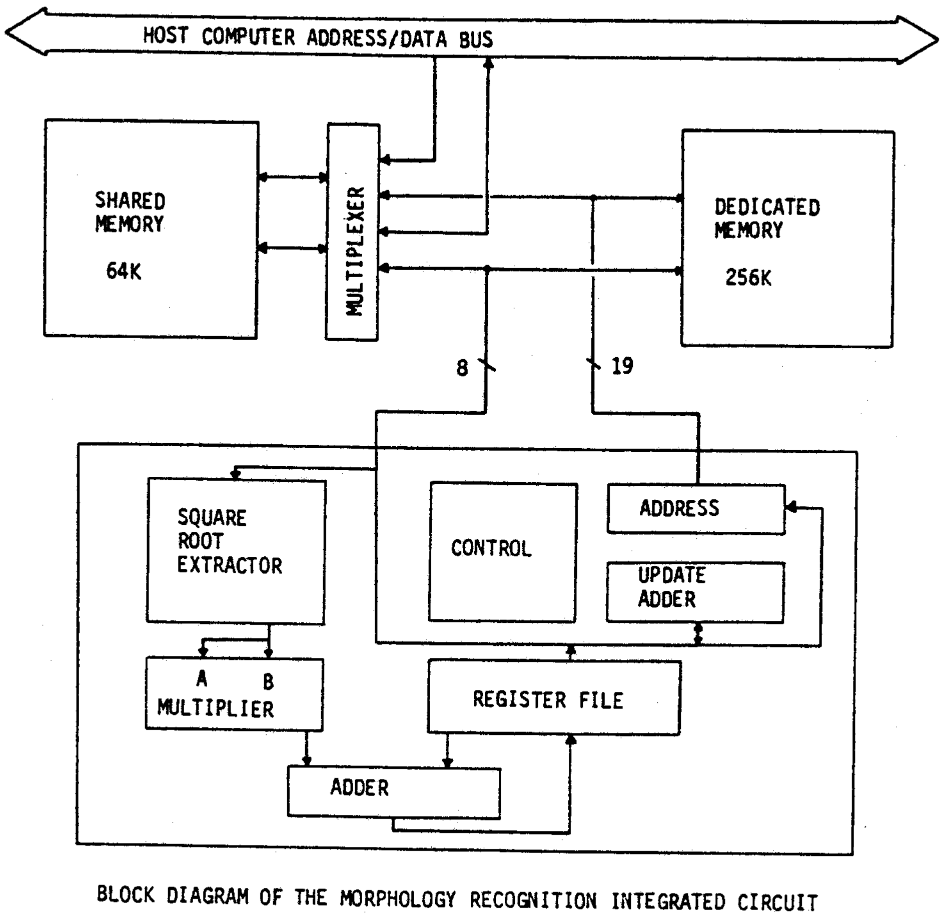


Figure 4-4: The block diagram of the ECG Processor.

as the square-root extractor, the multiplier, the adder and register file for the accumulation operations, the update adder for the update functions, a memory address unit, and a control unit. In the following sections, a register level diagram is presented for each of these function blocks along with a description on how the unit operates and a discussion of the decisions that were made in the unit's design. The final section presents the complete ECG processor including the circuits necessary to tie the function blocks together.

In the discussions of the square-root extractor and the multiplier, references are made to two versions of each. This is because in the initial development of the ECG processor, the square-root extractor was to generate an 8-bit result that provided an input to an 8x8-bit multiplier, but the precision of this root extractor was reduced to 4-bits at a later time. This change was made because the reduction in circuit complexity was seen as a way to speed up the circuit simulations which were becoming excessively time consuming. This reduction in precision has an effect on the values generated by the ECG processor, but further tests on the ECG analysis algorithm would have to be made to determine the effect of this precision on the accuracy of the ECG analysis. This design still demonstrates the technique for calculating the similarity between two matrices, and if it is found that more precision is required for the similarity calculations, future work can continue with the 8-bit root extractor and the 8x8 bit multiplier presented here.

4.2.1. The Square-Root Extractor

Three approaches for the design of a square root extractor were considered. One was an iterative process that used a multiplier and an adder, components which were already part of the IC design, to converge on the square-root much like the Newton-Raphson method for finding the root of a function. The second root extraction technique under consideration was the use of a look-up table in ROM, and the third option was a circuit designed exclusively for square-root extraction. The look-up table would require 256 words for an 8-bit input and so was rejected since it would be too large for this project. If the input word size were reduced, this could become a reasonable choice. The decision between the iterative process and a custom circuit for square-root extraction was made after an initial survey of literature. One paper, reference [31], described several iterative methods for square-root extraction and gave the computation time and the hardware requirements for each method. For a short word length like 8-bits, the fastest algorithm with the least hardware requirements was:

$$B_{K+1} = B_K + f(B_K)\,(N - B_K^2) \tag{4.3}$$

where B_k = square-root approximation after K^{th} iteration

N = value from which root is taken

$f(X)$ = a look-up table value for $\frac{2}{X}$ accurate to 'p' bits

The size of the look-up table, that is, the precision of 'p', determined how fast this algorithm converged to the correct value of the square-root. A small value of 'p' would require more multiplications for a given precision of the result than a large value of 'p', but the look-up table would be smaller.

References [32, 33, 34] are three papers which presented circuits for the direct extraction of square-roots. [32] is an early paper on the use of cellular logic arrays for square-root extraction while [33] and [34] present improved versions of the earlier circuit. Both [33] and [34] are based upon the same controlled-add-subtract (CAS) cell with the main difference being that [34] uses an almost completely regular layout which would be easy to generate for an IC while [33] uses a more compact, but more irregular layout pattern.

The circuit in [33] was the one selected for the ECG processor. It was selected over the iterative method because for the small input word size being used, the control circuitry and look-up table associated with this method were thought to be similar in size to a cellular logic array. As well, the iterative method would be considerably slower because two or three multiplications would have to be performed for each square root while with the logic array approach, not only is the root extraction faster than a multiplication, but square-root extraction could take place concurrently with any required multiplications. Of the two cellular logic array circuits, [33] was selected over [34] because for a small circuit, the layout of an irregular array would not be much more difficult than for a regular array and the savings in chip area would make the effort worthwhile.

The non-restoring square-root extraction method of [33] might be compared to the long division method of square-root extraction where the bits of the radicand are paired off and several trial subtractions are successively performed. The square-root algorithm is difficult to describe with mathematical equations, and so it is demonstrated in Figure 4.5 as a pencil and paper calculation. Figure 4.6 substitutes a numerical value for the variables in Figure 4.5 to provide a numerical example of this method of square-root extraction.

As presented in [33], the circuit is completely combinational and once the input value is stable, the extracted root is available after τ seconds where

$$\tau = \tau_s + \frac{(N^2 - N)\tau_c}{2} + (N-2)\tau_e \qquad (4.4)$$

and τ_s and τ_c are the full adder cell sum and carry delays, τ_e is the exclusive-or gate delay, and N is the number of bits in the root. However, when the initial design of the square-root extractor called for an 8-bit root, it was decided to modify the design into a pipelined architecture because for N=8, the total delay is about 28 carry delays which could become a limiting factor in the system clock. By changing to a pipelined architecture, the minimum clock period for the square-root extractor would be reduced from a 28 carry propagation delay to a 9 carry propagation delay. Another reason for developing a pipelined root extractor was that it was desired to have an 8 word first-in first-out (FIFO) buffer before the multiplier. Because the number of cycles required for multiplication depended on the values being multiplied, having an 8 word FIFO buffer would allow 8 values to be read from memory with no delay between reads, and then while the multiplier was busy emptying the buffer, other functions, like the update function, could access memory.

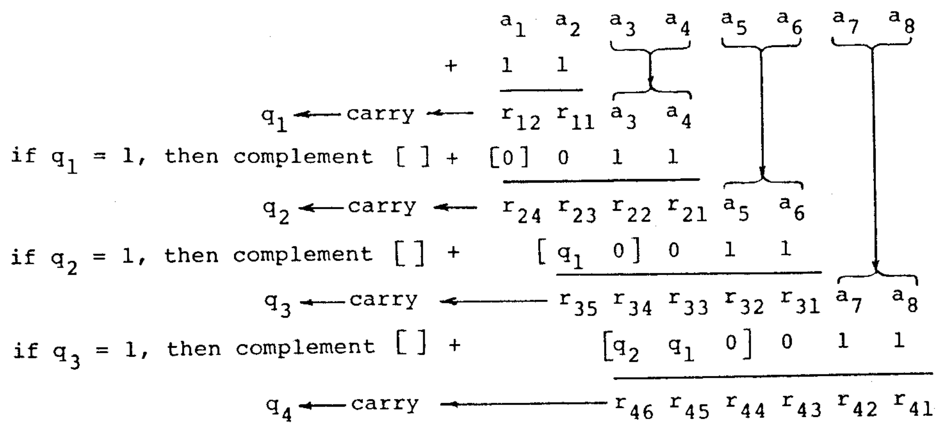
The circuit in [33], based upon the CAS cell in Figure 4.8, was

$A = 0.a_1a_2a_3a_4a_5a_6a_7a_8$

$Q = \sqrt{A} = 0.q_1q_2q_3q_4$

R = remainder after i^{th} cycle

r_{ij} = j^{th} bit of remainder after i^{th} cycle



a1 a2 a3 a4 a5 a6 a7 a8
+ 1 1
q1 ← carry ← r12 r11 a3 a4
if q1 = 1, then complement [ ] + [0] 0 1 1
q2 ← carry ← r24 r23 r22 r21 a5 a6
if q2 = 1, then complement [ ] + [q1 0] 0 1 1
q3 ← carry ← r35 r34 r33 r32 r31 a7 a8
if q3 = 1, then complement [ ] + [q2 q1 0] 0 1 1
q4 ← carry ← r46 r45 r44 r43 r42 r41

Figure 4-5: The square-root extraction algorithm.

```
A  =       0   1   1   1   1   0   0   1  =   121₁₀
           1   1
          --------
q₁ = 1 ←── 0   0   1   1
           1   0   1   1
          ----------------
q₂ = 0 ←── 1   1   1   0   1   0
               1   0   0   1   1
              --------------------
q₃ = 1 ←────── 0   1   1   0   1   0   1
                   1   0   1   0   1   1
                  ------------------------
q₄ = 1 ←────────── 0   0   0   0   0   0
```

$Q = q_1q_2q_3q_4 = 1011_2 = 11_{10}$

Figure 4-6: A numerical example of square-root extraction.

converted to a pipelined root-extractor by placing a register between each row of CAS cells as shown in Figure 4.7. In this figure, A_7-A_0 was the input value, and S_7-S_0 was the extracted root. The register latched the intermediate quotient and remainder values of the previous stage of the pipeline when ϕ_1 was high, and then the outputs of the registers would change when ϕ_2 was high. A square-root could be initiated every cycle and 8 cycles were required to extract a square-root

When the design of the ECG processor was changed so that only a 4-bit quotient was needed, the circuit in Figure 4.7 was modified by eliminating the last four rows of the array. The extractor still remained pipelined because its ability to act as a FIFO buffer, now a 4-word buffer, was still utilised.

A FORTRAN program was written to simulate the logic of the circuit presented in [33]. The quotients obtained from this method were compared to the floating-point quotients computed by the FORTRAN SQRT() function, and, as expected for a 4-bit word size, Figure 4.9 shows the error of the square-root extractor to be quite large for certain input values. The percent error is largest when the magnitude of the input value is small. As well, the figure shows that the extracted root is always less than the root obtained from the floating-point function. This means that the similarity value computed between two matrices will always be less than expected.

Because the error increases when the input values are small, it is possible that classification errors will be made when few non-zero elements exist in a matrix, or when the matrix is composed mainly of elements with small magnitude. Whether this is a problem in the ECG processor should be determined with actual data. Two possible solutions to this problem are to increase the precision of the square-root, or to maintain values of $\sum^i A_i$ and $\sum^i B_i$ on the IC rather than having the host computer compute A_T and B_T. The problem with incorrect classification arises from the consistent rounding down of $\sqrt{A_i}$ and $\sqrt{B_i}$ which makes $\sum^i (\sqrt{A_i})^2$ and $\sum^i (\sqrt{B_i})^2$ less

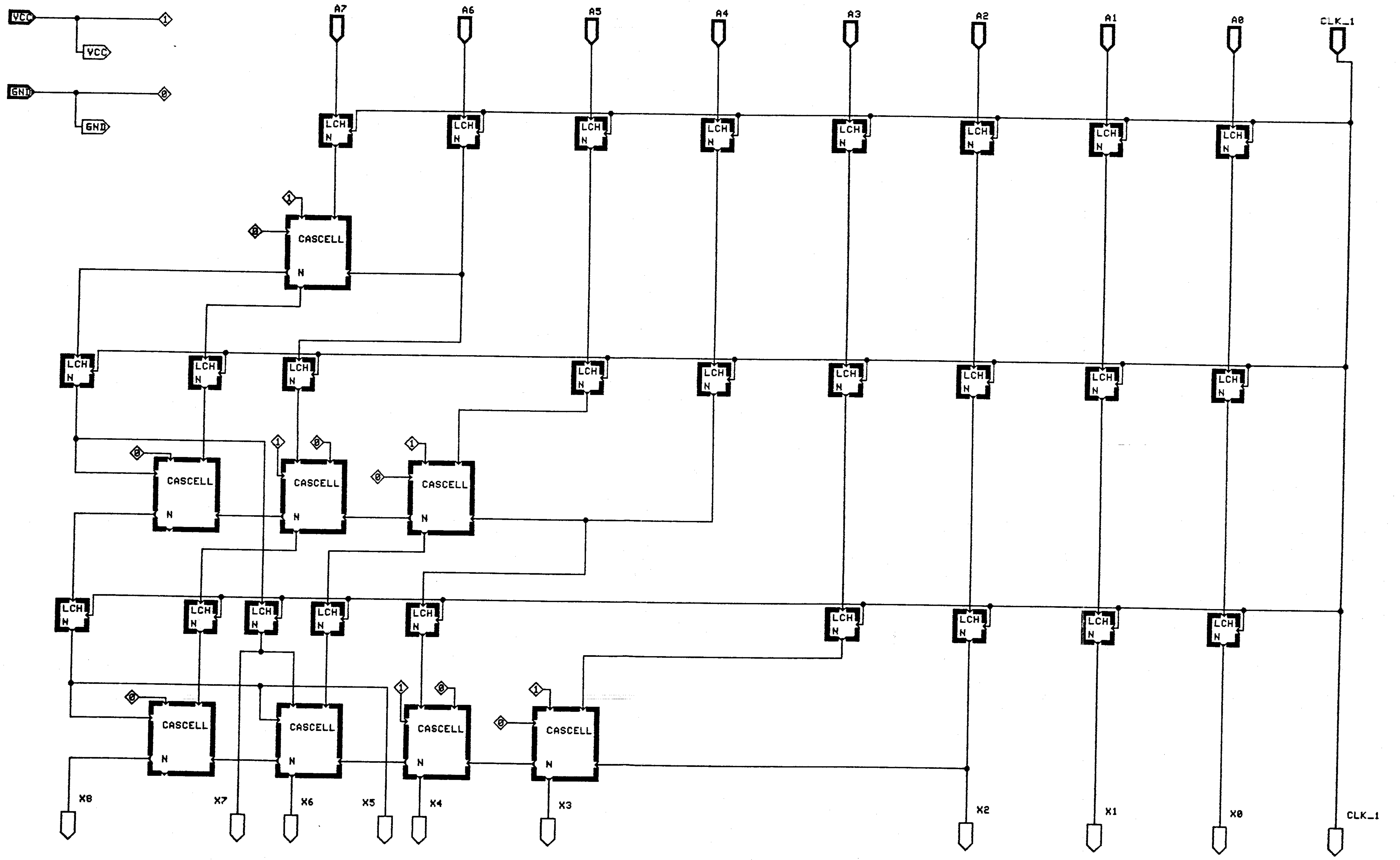


Figure 4-7: A pipelined square-root extractor.

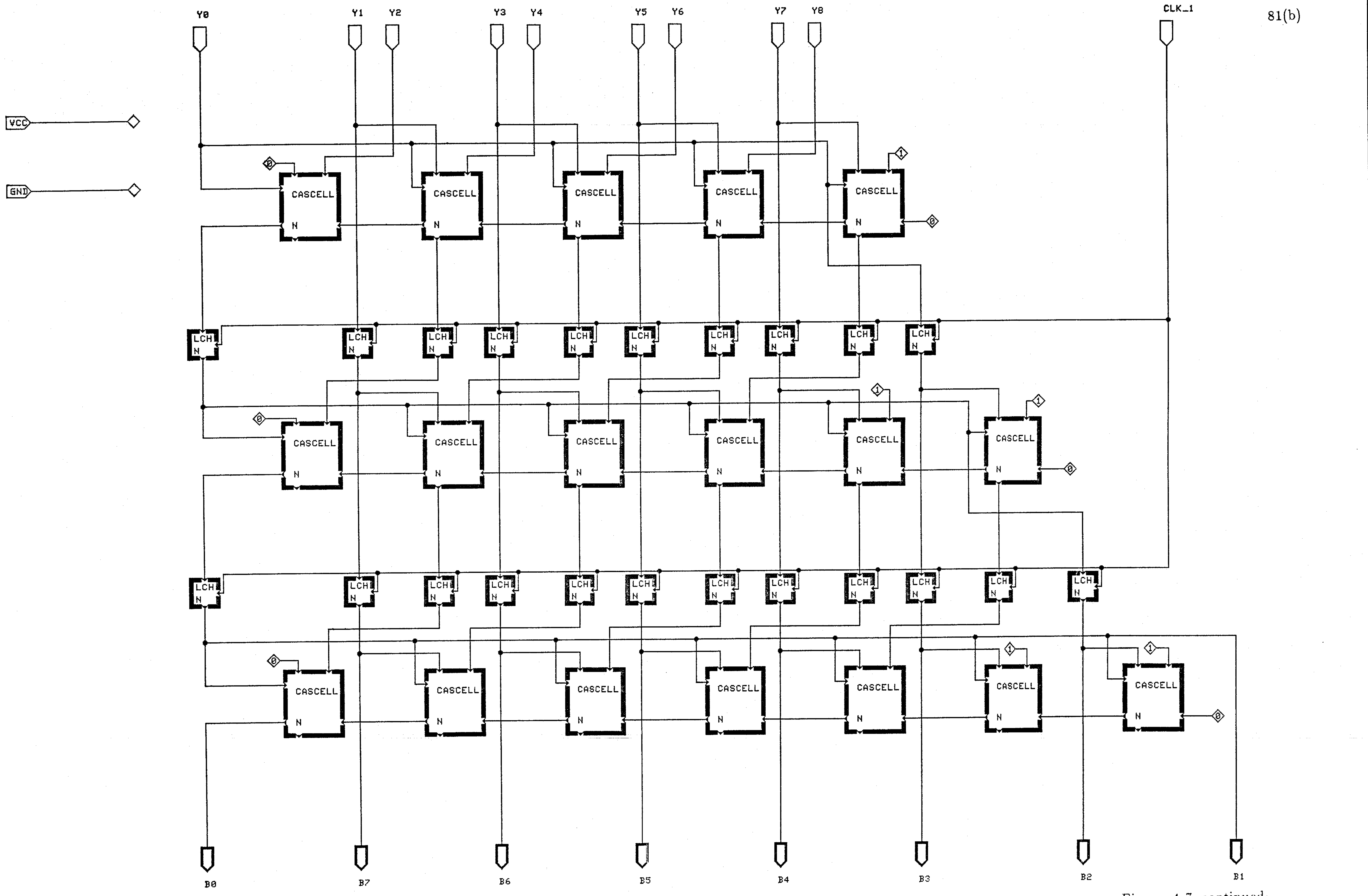


Figure 4-7 continued

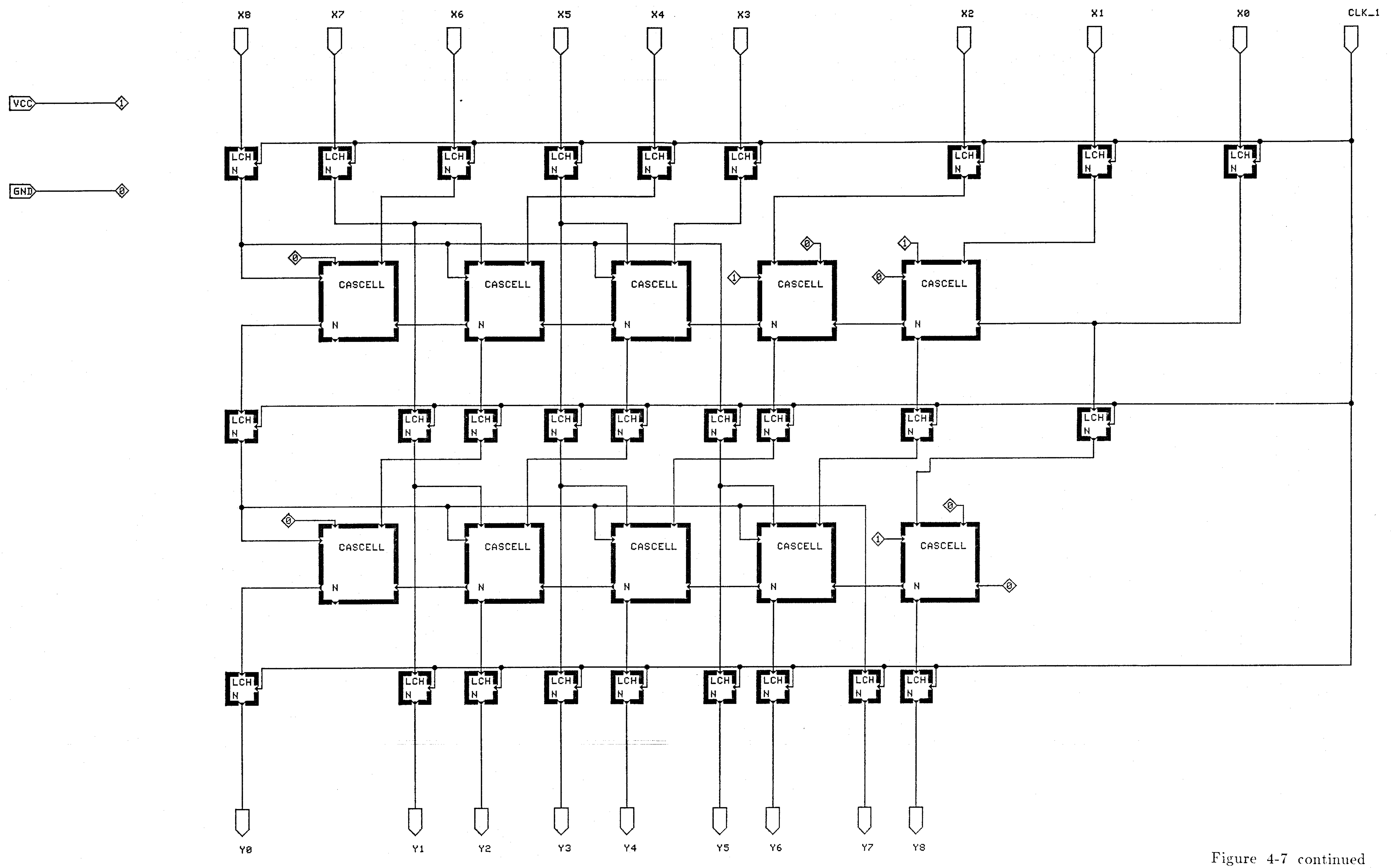


Figure 4-7 continued

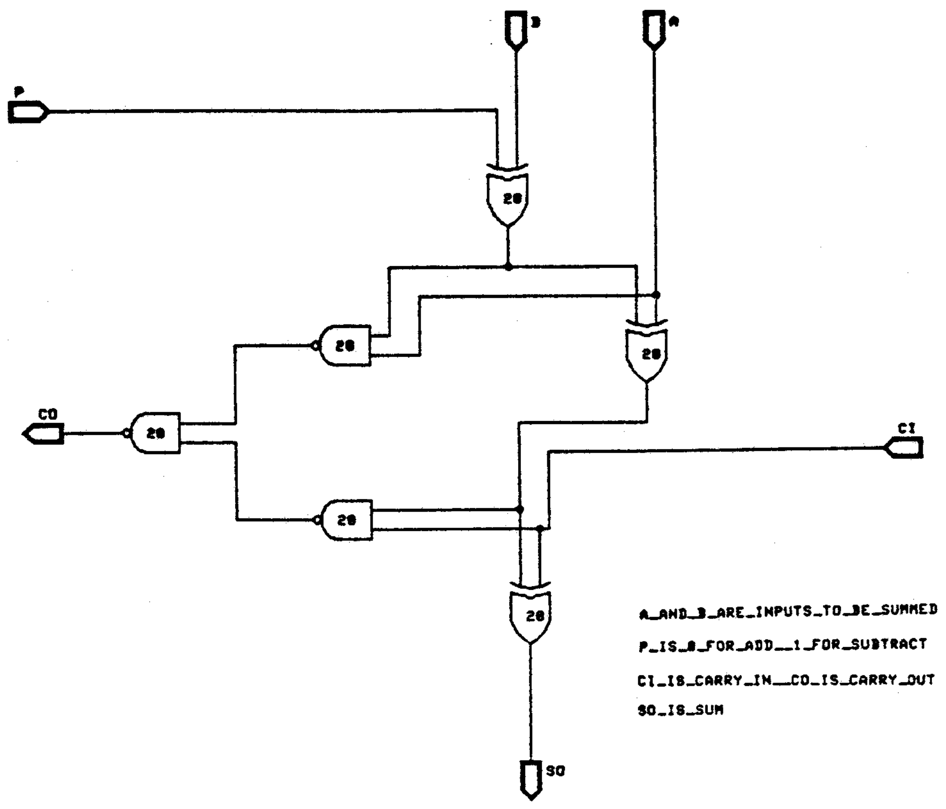

B
A
P
28
CO
CI
A_AND_B_ARE_INPUTS_TO_BE_SUMMED
P_IS_0_FOR_ADD__1_FOR_SUBTRACT
CI_IS_CARRY_IN__CO_IS_CARRY_OUT
SO_IS_SUM
SO

Figure 4-8: The controlled add/subtract cell schematic.

than A_T and B_T even though, if the precision were sufficient, they would be equal. The accumulated loss can make the similarity value between two identical matrices less than that between two slightly dissimilar matrices. For example, consider the simple case of comparing {2,2} to {2,2} and {125,100}. Using the 4-bit precision of the proposed circuit, the similarity calculation results between {2,2} and {2,2} is 0.5 while the results between {2,2} and {125,100} are 0.671. Using $\sum^i A_i$ and $\sum^i B_i$ in place of A_T and B_T eliminates the inaccuracies due to the consistent rounding down of $\sqrt{A_i}$ and $\sqrt{B_i}$. Both proposals for increased accuracy, however, increase the area of the chip.

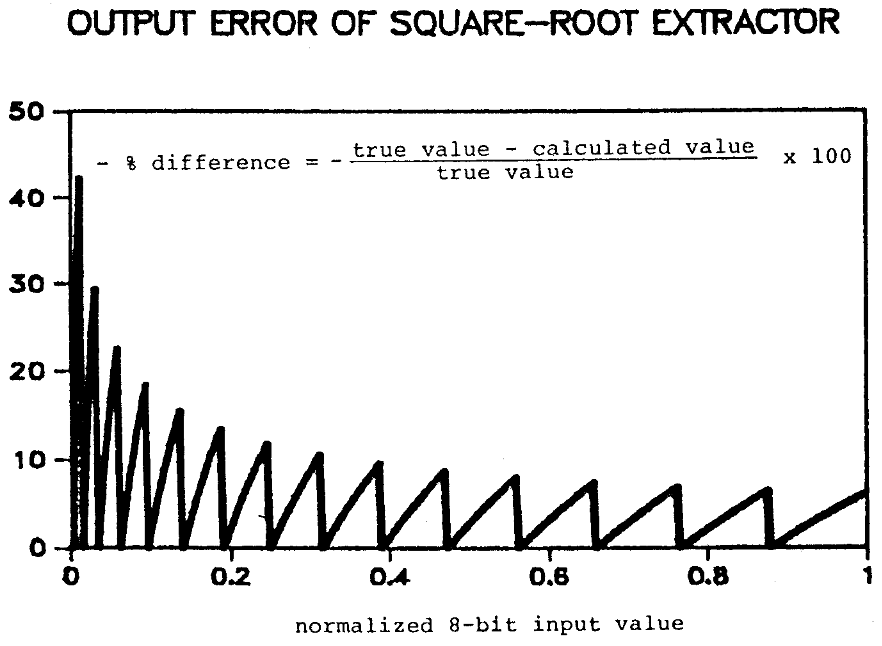


Figure 4-9: The error in a 4-bit square-root extractor.

4.2.2. The Multiplier

Two types of multipliers were developed for this project. The first multiplier was based upon canonical signed digit (CSD) recoding. The CSD multiplier was developed for an early version of the ECG processor, but when the requirements for an 8x8 bit multiplier were changed to a 4x4 bit multiplier, the CSD multiplier was exchanged for a pipelined multiplier because for small arguments like 4-bits, the CSD multiplier becomes less efficient.

4.2.2.1. The CSD Multiplier

For the first version of the ECG processor, an 8x8 bit multiplier was needed. It was decided rather than using a non-recoded multiplier, it would be more interesting to investigate the usefulness of CSD coding in digital multiplication circuits. This multiplier would be a shift-and-add multiplier and CSD coding was expected to improve the performance of the multiplier because it would reduce the average number of add operations per multiply to a minimum. CSD coding had been investigated in earlier work [35] and the expected performance improvement was known, however, the cost of this improved performance as measured in terms of increased circuit complexity and area would have to be determined from an actual IC layout.

The speed improvement offered by CSD coding is based on the operation of a shift-and-add multiplier where an addition or subtraction must be performed for every non-zero digit in one of the unsigned binary multiplier arguments. On average, for a non-recoded multiplier, the number of addition/subtractions is half the number of bits of the multiplier input, that is, if it is an 8x8 bit multiplier, the average multiplication would require four add/subtract operations. By recoding the multiplier input with some recoding algorithm, the number of addition/subtractions can be reduced, and in the case of CSD coding, the number of add/subtract operations can be reduced to a minimum. The number of non-zero digits, N, in a B-bit CSD coded value is given in [35] as:

$$N = \frac{1}{3} + \frac{1}{9B}\left(1 - \left(-\frac{1}{2}\right)^{B}\right) \tag{4.5}$$

For large values of 'B', the average number of non-zero bits in a number is 1/3 which means an average of 33% fewer add/subtract operations are required for a CSD multiplication when compared to a non-recoded multiplication. As 'B' is made smaller, the average number of add/subtract operations becomes the same as for a non-recoded multiplier. For an 8x8-bit multiplier, 30% fewer add/subtracts are required, and for a 4x4-bit multiplier, 28% fewer add/subtracts are required. One disadvantage with CSD coding is that each digit is represented by one of three values from the set {-1,0,1} making the storage requirement for CSD coded numbers greater than that for non-recoded binary numbers. For example, in the multiplier presented here, one 8-bit two's complement binary number is converted to a

CSD representation which requires two 8-bit words; one word is the magnitude word which marks all non-zero digits with a '1', and the other word is a sign word which indicates whether the corresponding digit in the magnitude word is a '1' or a '-1'.

To demonstrate the CSD code representation, the following numerical example is presented. The conversion from two's complement notation to CSD code is based upon the logic diagram shown in Figure 4.10.

Binary unsigned

$$0\ 1\ 1\ 0\ 0\ 1\ 1\ 1_2 = 103_{10}$$

CSD code

$$1\ 0\ {-1}\ 0\ 1\ 0\ 0\ {-1} = 128_{10} - 32_{10} + 8_{10} - 1$$

Binary Representation of CSD code

```
    1   0   1   0   1   0   0   1      =  magnitude word
    0   0   1   0   0   0   0   1      =  sign word
```

One option in the design of this multiplier is whether to convert one or both multiplier inputs to CSD code. By converting both inputs, the one with the least number of non-zero inputs could be selected as the multiplier while the other became the multiplicand. According to [37], when both operands are converted to CSD code, an average of approximately 16% fewer add/subtract operations would be required for an 8x8 multiplier when compared to the case where only one input was converted. However, additional hardware would be necessary to make the selection between two CSD coded operands. For this multiplier, it was decided to convert only one value to CSD code because for the application in the ECG processor; the same converted value could be used to multiply eight other values and so only one CSD conversion would be necessary for all eight multiplications.

Another option in the design of the multiplier was the number of shift operations which could be performed per clock cycle. Since the total

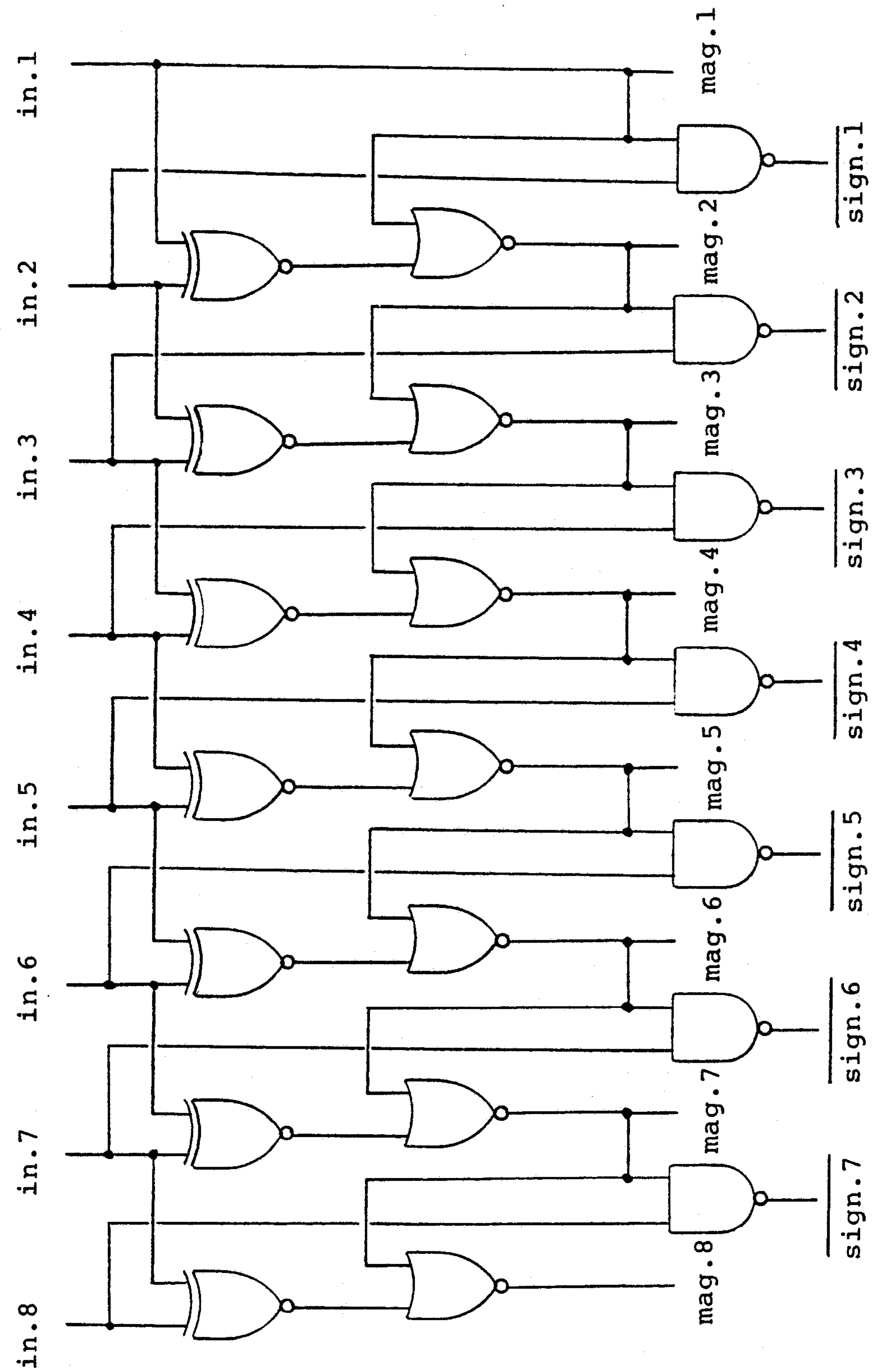

in.1
in.2
in.3
in.4
in.5
in.6
in.7
in.8
mag.1
mag.2
mag.3
mag.4
mag.5
mag.6
mag.7
mag.8
$\overline{\text{sign.1}}$
$\overline{\text{sign.2}}$
$\overline{\text{sign.3}}$
$\overline{\text{sign.4}}$
$\overline{\text{sign.5}}$
$\overline{\text{sign.6}}$
$\overline{\text{sign.7}}$

Figure 4-10: The schematic of the two's complement to CSD code converter [36].

multiply time equals the number of add/subtract cycles plus the number of shift operations which occur when the adder/subtractor is not busy, the ability to shift more than one bit at a time can speed up the multiplication. The fastest multiplication would take place if a barrel shifter were used; a barrel shifter can shift its input value any number of bit positions in one clock cycle and it could eliminate all multiplication time taken up by shift operations. However, the barrel shifter can grow to be large for big word sizes because the number of transistors in the shifter equals n^2 where 'n' is the bit size of the input word.

In order to determine the benefits of the ability to shift more than one bit position per clock cycle, a computer program was written to compute the average multiply time assuming that an add/subtract operation took one cycle, a shift that occurred when the adder was not busy also took one cycle, and that all input values had an equal chance of appearing. Figure 4.11 compares the average multiply time of an 8x8-bit multiplier for various shift lengths of both the CSD multiplier and a non-recoded multiplier. The greatest improvement occurs when a change is made from a one bit shift per cycle to a two bit shift per cycle while the improvement beyond a four bit shift per cycle is negligible.

This multiplier was designed to shift two bits per cycle not only because the greatest performance improvement was to be had at this point, but also because CSD code has the property of having no two consecutive digits as non-zero. This meant that the shifter and the control circuitry were simplified to 17 gates as shown in Figure 4.12.

A block diagram of the multiplier is shown in Figure 4.13, and the timing diagram for a sample multiplication is given in Figure 4.14. The number of cycles taken in this example represents the maximum for an 8-bit signed multiplication.

To compare the area and speed characteristics of the CSD multiplier to

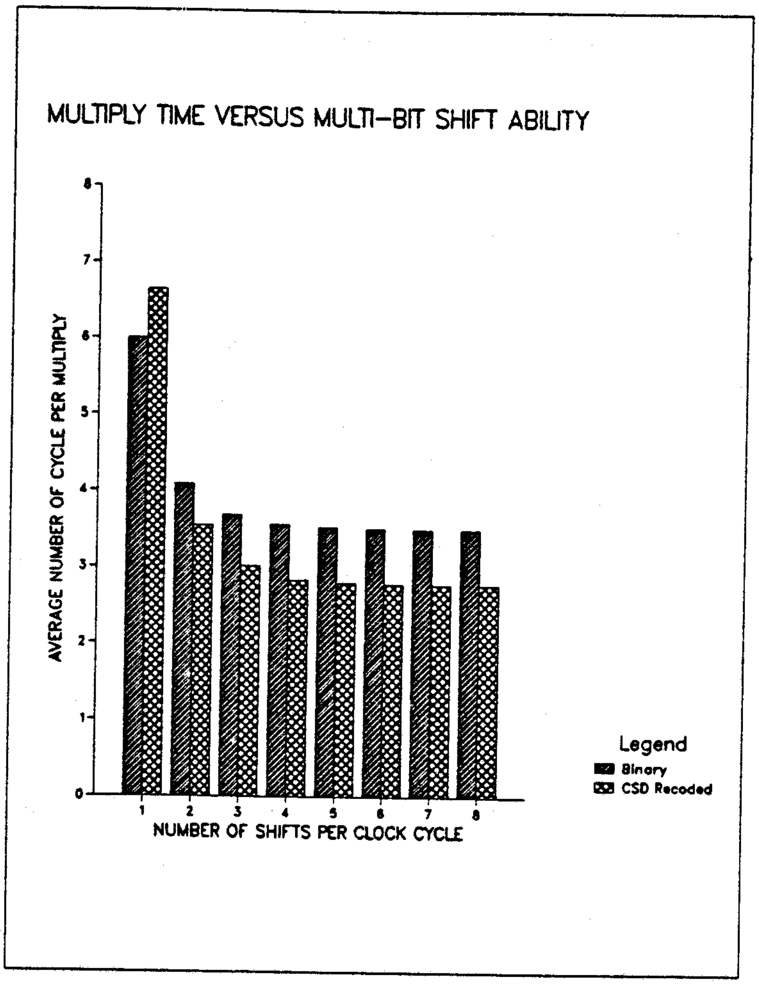


Figure 4-11: Multiplication time versus multi-bit shift ability.

a non-recoded multiplier, the areas of the various multiplier components were determined from the final layout of the CSD multiplier. Then by eliminating the components which would not be necessary for the non-recoded multiplier, the area of the non-recoded multiplier was determined. This method of comparison should be accurate because both multipliers would have similar maximum clock rates and would be based upon the same technology. Even if changes like using a dynamic shift register instead of a static shift register, were made to one multiplier, this improvement would also apply to the other multiplier.

The areas of the CSD multiplier components are listed in Table 4.1.

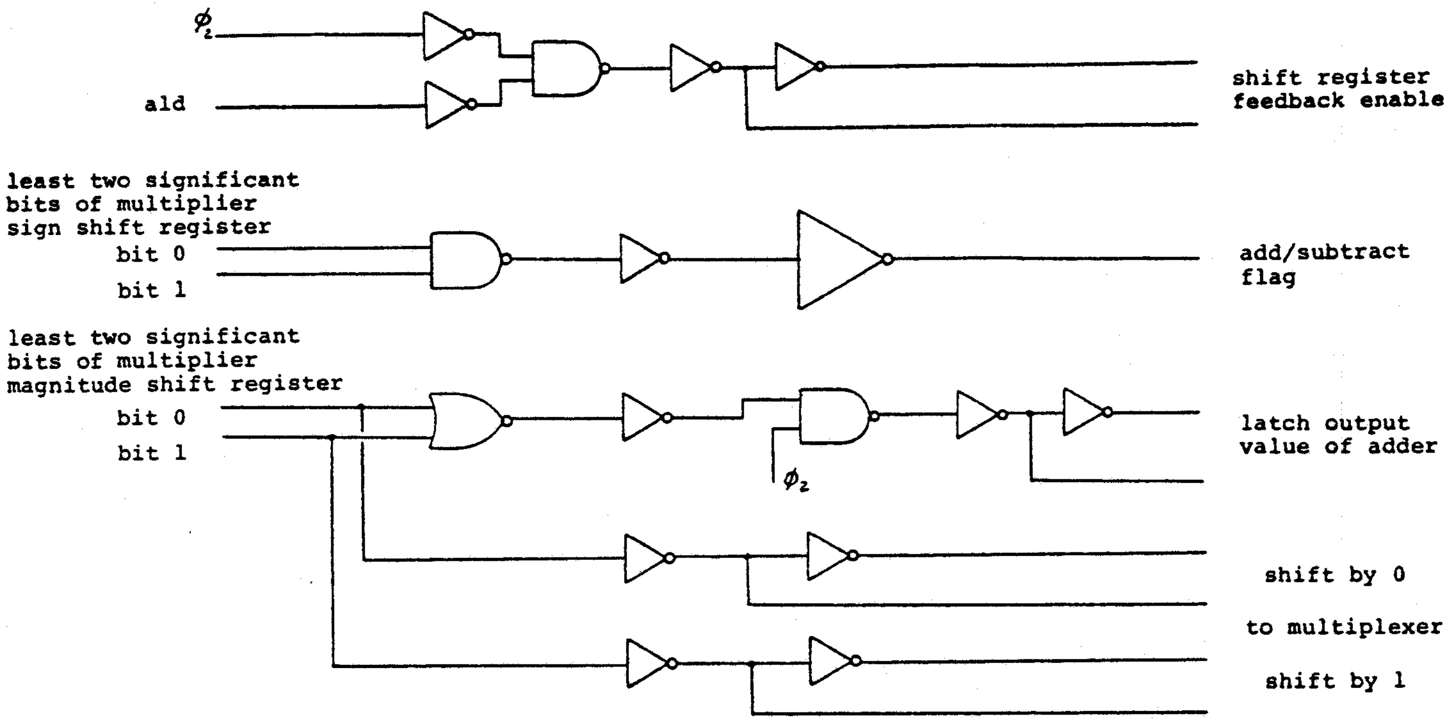

ϕ_2
ald
shift register feedback enable
least two significant bits of multiplier sign shift register
bit 0
bit 1
add/subtract flag
least two significant bits of multiplier magnitude shift register
bit 0
bit 1
ϕ_2
latch output value of adder
shift by 0
to multiplexer
shift by 1

Figure 4-12: The schematic diagram of the multiplier controller.

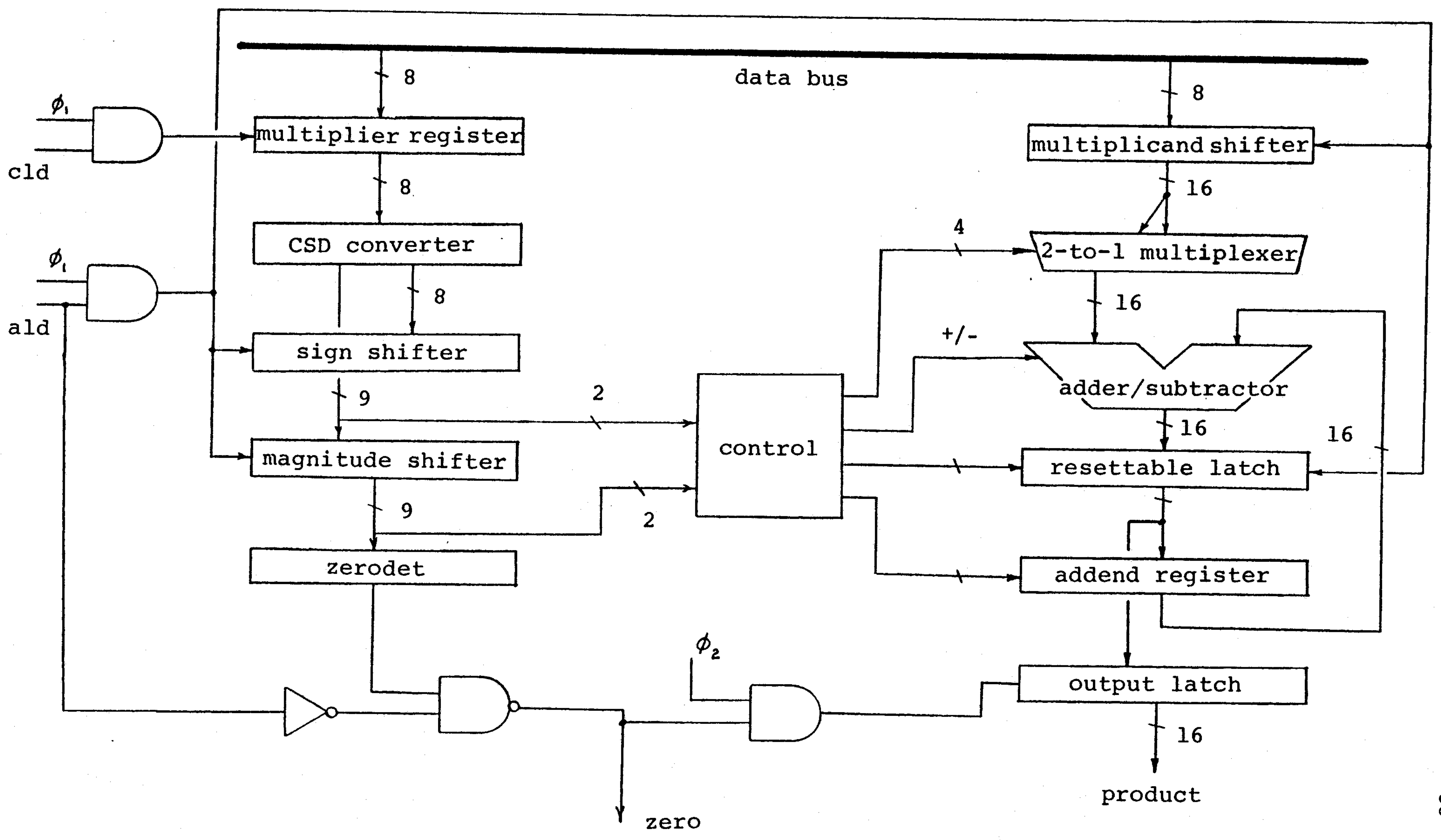
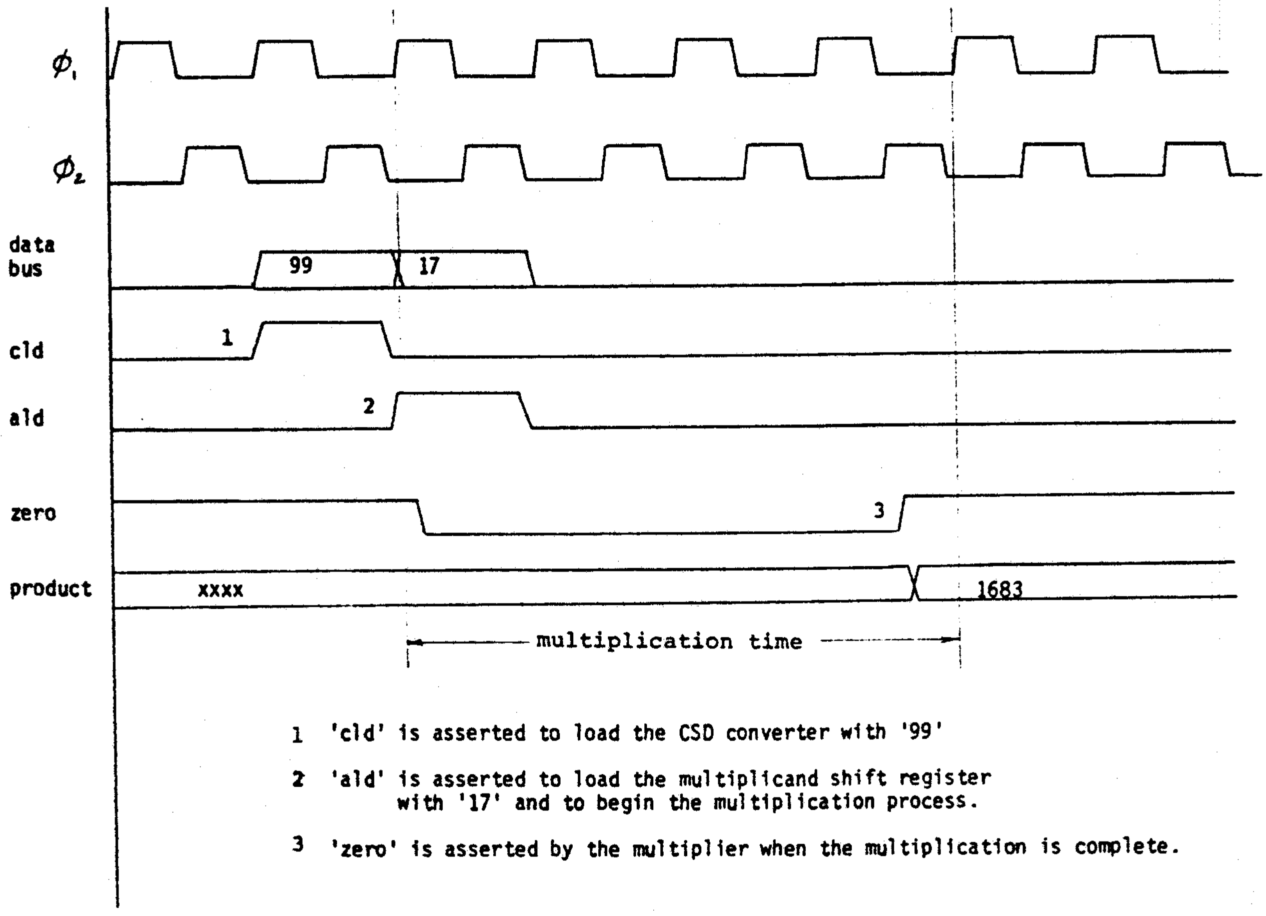


Figure 4-13: The register diagram for the CSD multiplier.



ϕ_1
ϕ_2
data bus
99
17
cld
1
ald
2
zero
3
product
xxxx
1683
multiplication time
1 'cld' is asserted to load the CSD converter with '99'
2 'ald' is asserted to load the multiplicand shift register with '17' and to begin the multiplication process.
3 'zero' is asserted by the multiplier when the multiplication is complete.

Figure 4-14: The timing diagram for the CSD multiplier.

The areas given are based upon the 5μ design scale and not the 3μ fabrication scale.

CSD Multiplier Component Areas

Component	Area (μ^2)
adder/subtractor	1.206×10^6
2-to-1 multiplexer	0.352×10^6
zerodet	0.083×10^6
control	1.086×10^6
areas	
CSD converter	0.313×10^6
multiplier register	0.194×10^6
reset-able latch	0.480×10^6
addend register	0.416×10^6
output latch	0.416×10^6
sign/magnitude shifters	$2 \times 0.480\times10^6$
multiplicand shifter	0.960×10^6
TOTAL	6.466×10^6

Table 4-1: Areas of the CSD multiplier components.

To find the area of the non-recoded multiplier, the areas of the multiplexer unit, one 8-bit shift register, and the CSD code converter were eliminated from the total CSD multiplier area. The control areas for these multipliers were assumed to be constant as the control requirements were similar in complexity.

The comparison of the speed and area characteristics between the CSD multiplier and the non-recoded multiplier is summarized in Table 4.2. The results show that by the area x time measure, the CSD multiplier is superior to non-recoded multipliers. For both of these multipliers, the time for shift operations could be nearly eliminated by increasing the clock rate so that while an add operation may take several clock cycles, a shift could be performed every clock cycle. In this case, the CSD coded multiplier would be 30% faster, consume about 21.5% more area, and have about a 15% smaller area-time product than a non-recoded multiplier.

CSD Multiplier versus One-shift Non-recoded Multiplier

	Ave. mult time (cycles)	Area ($x10^6\mu^2$)	Time x Area
Non-recoded	7.5	1.975	11.9
CSD coded	3.6	2.328	8.1
% difference	-41%	21.5%	-32%

Table 4-2: Comparison of multiplier speed area characteristics.

4.2.2.2. The Pipelined Array Multiplier

When the multiplier requirements changed from an 8x8 bit multiplier to a 4x4 bit multiplier, it was decided to change from the CSD coded multiplier to an array multiplier presented in [38]. The reason for the change was because for the smaller input word size, the CSD coded multiplier becomes only slightly smaller yet significantly slower than the array multiplier. As a demonstration of this, the area of the CSD multiplier was estimated from the CSD layout of the previous section by assuming that all areas, except the CONTROL unit, were halved, and the area of the array multiplier was found directly from its layout. The resulting area for the CSD coded multiplier was $1.36x10^6\mu^2$ while the area for the array multiplier was $1.72x10^6\mu^2$ (on the 3μ fabrication scale). Besides being faster, the fixed multiplication time made the design of the complete ECG processor easier. However, the array multiplier's advantage quickly disappears as the input word size is increased since the area of the CSD multiplier increases in the order of 4N for an NxN bit multiplier while the area increase for the array multiplier increases in the order of N^2.

The principle of operation for the pipelined multiplier is straight forward. A register diagram of the multiplier is given in Figure 4.15. There are three adders in series each of which add shifted values of the multiplicand to compute the final product. Whether a shifted value of the multiplicand is to become part of the product or not is determined by the

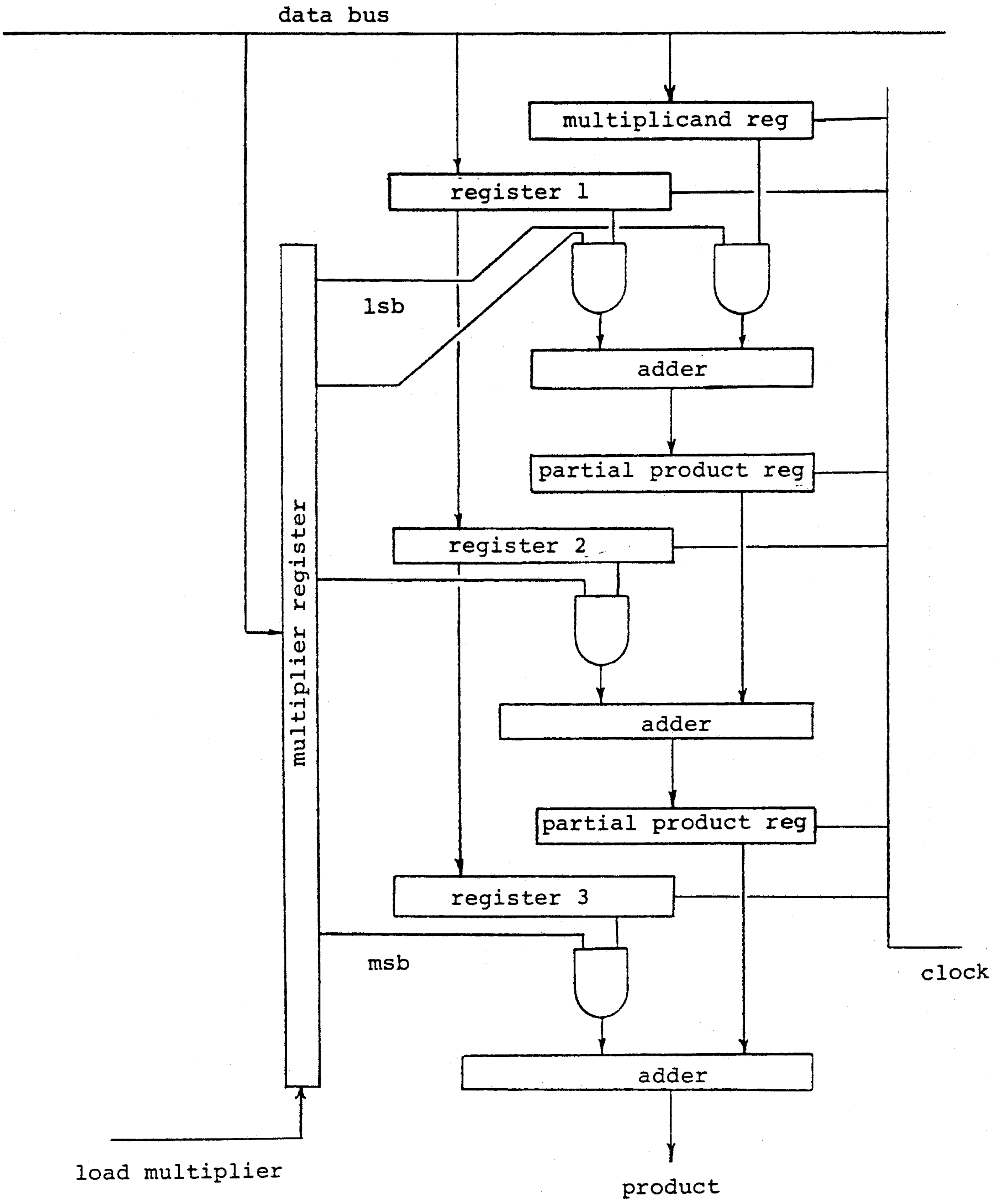


Figure 4-15: The register diagram for the array multiplier.

bits in the multiplier register. For example, if the least significant bit of the multiplier operand is set, then the unshifted value of the multiplicand is added towards the product. If the second least significant bit is set,then the multiplicand value shifted left by two is added towards the product. The third multiplier bit adds the multiplicand value shifted left by two bit positions, and the fourth, or most significant, bit adds the multiplicand value shifted left by 3 bit positions.

Figure 4.16 shows the timing diagram for the pipelined multiplier.

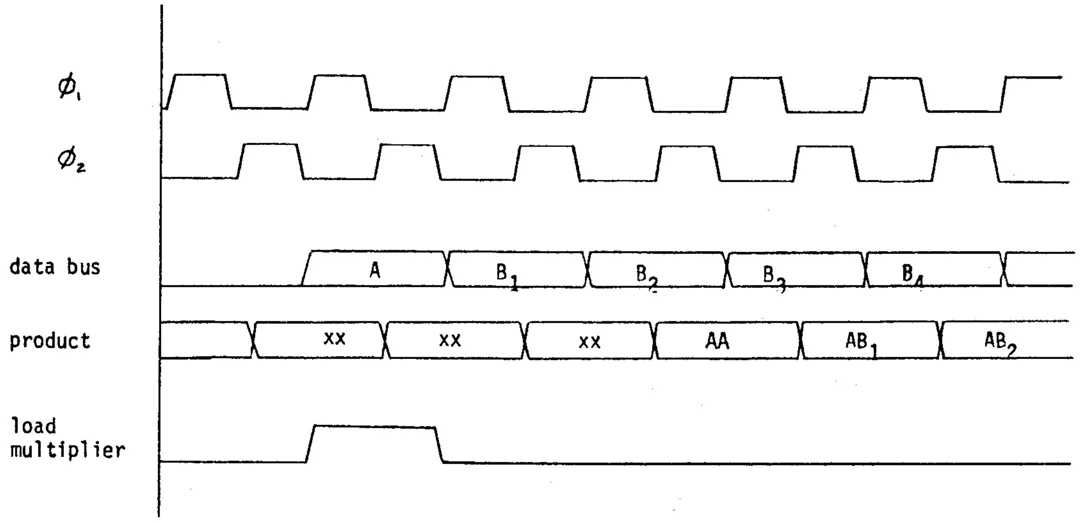


Figure 4-16: The timing diagram for the array multiplier.

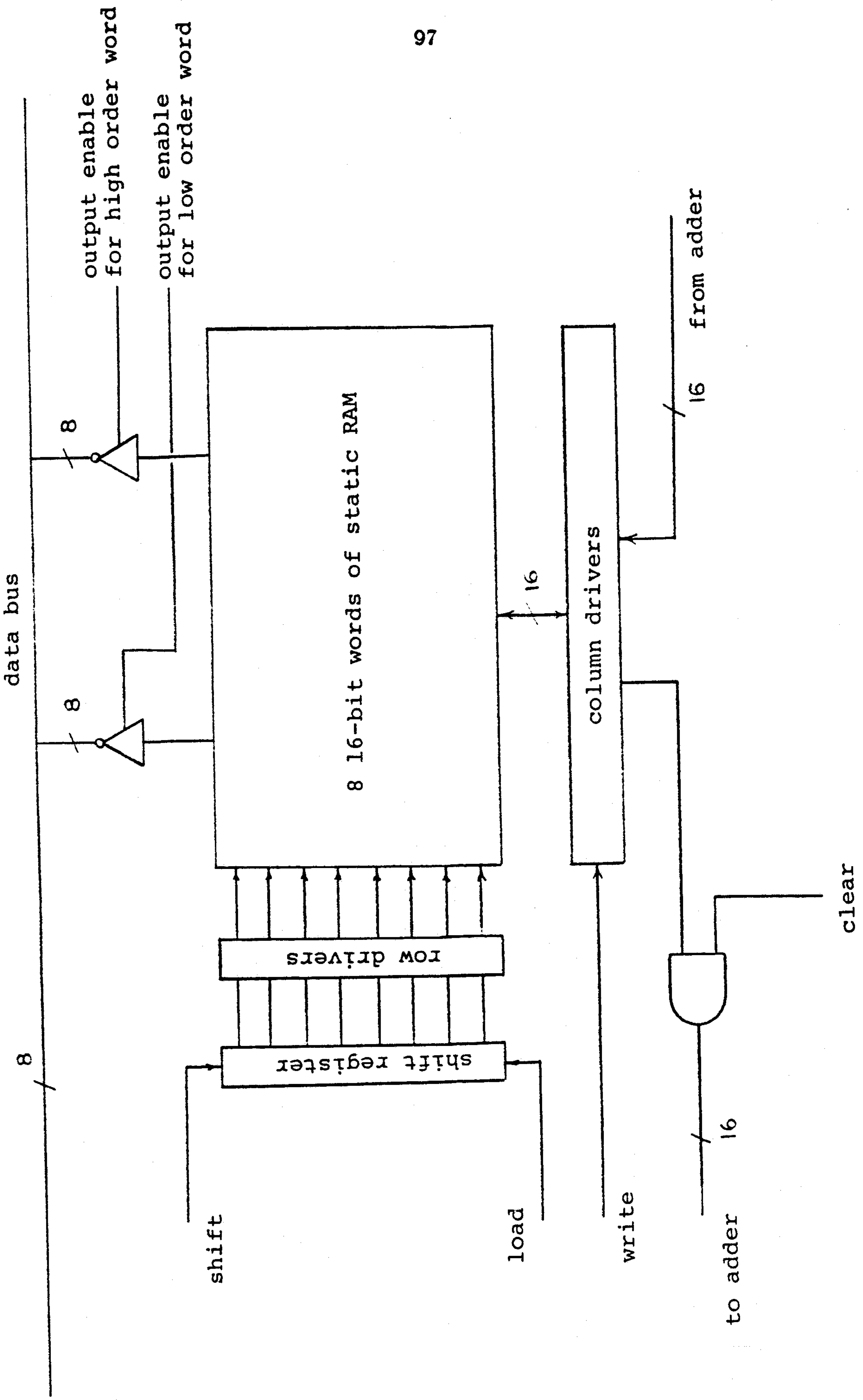
The multiplier operand is the first value loaded; it is loaded into both the multiplier and multiplicand registers although the resulting value of this multiplication, the square of the multiplier operand, is not used. Following the multiplier register load, the multiplicands are loaded at a rate one every clock cycle, and four cycles later, the corresponding products are available at a rate of one every clock cycle.

4.2.3. The Register File

To perform the summation function of Equation 4.2, an accumulating register is needed, and since the unknown matrix is being compared to 8 reference matrices, 8 accumulating registers are required for the ECG processor. These registers could have been built up from d-latches, but considerable area can be saved if a memory similar to random access memory (RAM) is used.

The two main types of RAM are dynamic and static; dynamic RAM uses capacitance to store information while static RAM uses a flip-flop. Dynamic RAM is more compact than static RAM, but it is more complicated to use because charge leaks from the capacitors which store the information and control circuitry must be developed to periodically refresh the memory cells. For this design, the static RAM was selected for two reasons: one reason was that static RAM was easier to use, and the other reason was that initially it was decided to make the ECG processor completely static. A complete static design would allow the clock to be stopped at any time and would give the circuit designer more flexibility when designing the shared memory interface.

The register level diagram for the register file is shown in Figure 4.17. It stores 8 16-bit words. The 16-bit word size was selected because it allowed a comfortable margin from overflow in the accumulation operations. The expected largest accumulation for a similarity calculation would occur when all 500 sample points of an ECG wave form were spread evenly over the recognition matrix and the reference matrix were the sum of, say, ten such matrices. The total then would be about $961 \times \sqrt{20}\sqrt{2} \approx 6000$ which would require a 13 bit representation. As can be seen, fewer bits could have been used for the accumulation process, but as 2 8-bit words must be written to shared memory, the extra three bits were seen as a way to guard against overflow.



data bus
8
8
8
output enable for high order word
output enable for low order word
8 16-bit words of static RAM
16
column drivers
16
from adder
row drivers
shift register
shift
load
write
16
to adder
clear

Figure 4-17: The register diagram for the register file.

The memory address is controlled through a shift register rather than an address decoder because each location in memory is always accessed sequentially and so there is no need for an address decoder. RAM, then, is not a completely accurate term for the register file, but aside from the shift register which could be replaced by an address decoder, it was felt that the memory could be considered as a RAM.

The memory outputs are connected to the ECG processor data bus through tri-state buffers. As the data bus is 8-bits wide, the tri-state buffers are divided into two groups: one for writing the lower 8-bits of the register file contents, and the other for the higher 8-bits. The outputs are also connected to an adder through an AND gate. The purpose of the AND gate is to provide a zero input to the adder for the initial summation iteration of the similarity calculation. This was necessary because otherwise the contents of the memory from some previous calculation would be accumulated into the new calculations. Another solution might have been to write a zero value to all memory locations before beginning the accumulation process, but the selected approach eliminated the controller states necessary to clear all memory locations.

To change the address of the memory, two signals are used. An asserted LOAD signal loads the shift register so that the first row of the memory is always selected. The SHIFT signal is asserted to access the next higher row of memory; the change in the row select occurs when either clock signal goes high, and the memory data lines should be valid before the same clock signal goes low. The WRITE signal enables the column drivers to write the value of the memory input lines into the memory.

4.2.4. The Update Adder

As mentioned in the previous section, the ECG processor was to have the ability to move data between shared and dedicated RAM, to reset matrix elements to zero, and to add unknown matrix values to reference matrices. These are the functions carried out by the update adder shown in Figure 4.18. A complete description of the update adder functions is given by Equations 4.6 and 4.7:

if $C_6 = 1$ then

$$\mathbf{A} \leftarrow \frac{C_5 \times \mathbf{A}}{1+C_3} + \frac{C_4 \times \mathbf{B}}{1+C_2} \tag{4.6}$$

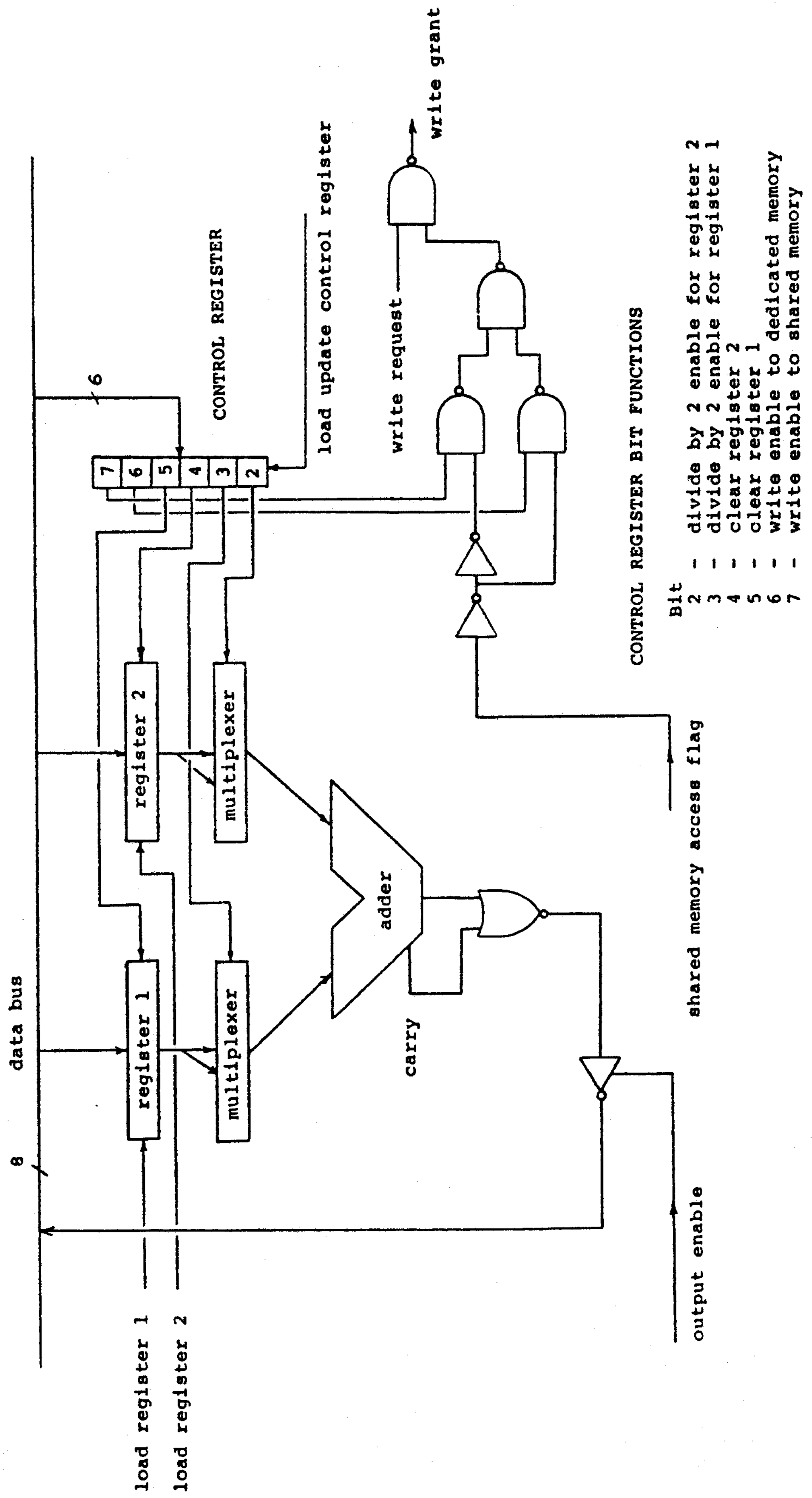
if $C_7 = 1$ then

$$\mathbf{B} \leftarrow \frac{C_5 \times \mathbf{A}}{1+C_3} + \frac{C_4 \times \mathbf{B}}{1+C_2} \tag{4.7}$$

where **B** is the unknown matrix, **A** is the reference matrix, and bits C_2-C_7 are control bits which may be either 1 or 0. These bits define the function of the update adder. Suppose, for example, that the unknown matrix **B** was to be reset to zero. Then C_6, C_5, and C_4 would have a value of zero, and C_7 would be set to one. Another example would be dividing the elements of the unknown matrix by two and adding them to a reference matrix to give the function

$$\mathbf{A} \leftarrow \mathbf{A} + \frac{\mathbf{B}}{2}$$

C_3 and C_6 would be set to 0 and C_5, C_4, C_2, and C_7 would be set to 1. The option of dividing a matrix element by two was provided because the maximum element magnitude of 255 would be quickly exceeded if several matrices were added together as in the case of the reference matrices; dividing the data by 2 before the addition would help prevent overflow.



CONTROL REGISTER BIT FUNCTIONS

Bit		
2	-	divide by 2 enable for register 2
3	-	divide by 2 enable for register 1
4	-	clear register 2
5	-	clear register 1
6	-	write enable to dedicated memory
7	-	write enable to shared memory

Figure 4-18: The register diagram for the update adder unit.

These control bits are set by loading the control register from shared memory; the load occurs during the loading of the forward address pointers. Of the 16 bits available for the forward pointer to the matrices for the update function, only nine bits are needed. This leaves seven extra bits of which six are selected for the control register.

Figure 4.18 shows how the update adder performs its functions. Register 1 stores the unknown matrix element value while register 2 stores the reference matrix value. The multiplexers following these registers perform the optional divide by two function by selecting either the register value or the register value shifted right by one position. The output of the adder is NOR'd with the carry out signal to ensure that the output value is 255 should an addition overflow occur. This effectively 'clips' the matrix data and distorts the recognition matrix, but it was seen as the best approximation in the event of an overflow. The output enable line enables the tri-state buffer to write the update adder results to external memory.

The logic for the 'write request' and 'write grant' signals is connected to the control bits C_6 and C_7. If C_6 is reset then no write can be made to dedicated memory, and if C_7 is reset then the update adder results are not written to the unknown matrix. If both of these bits are reset, then the update function is effectively disabled.

4.2.5. The Memory Address Unit

As the ECG processor operates upon data from an external memory, it must have a memory address unit that provides the appropriate addresses for the requested data. In the 64 unknown matrices and the 256 reference matrices, four types of data are accessed through a 19-bit address bus. These four types of data are:

- the unknown matrix element for the similarity calculations,
- the reference matrix element for the similarity calculations,

- the unknown matrix element for the update adder,
- the reference matrix element for the update adder.

The circuit for computing these addresses is shown in Figure 4.19. The 10-bit counter is used to sequentially access the matrix elements. Because all matrix data are aligned along 1024 byte borders, the least ten significant bits are common to all four types of data. From the circuit diagram, it can be seen that this counter may be loaded with one of two values. This counter is normally reset to zero at the start of each similarity matrix calculation, but when the ECG processor is in the 'reset' state, it is initially reset to 978 to allow the immediate polling of the 'stop-byte'. There is also a 3-bit counter which provides the address of each reference matrix. For each unknown matrix element access, an element from each of the 8 reference matrices is accessed by incrementing this counter eight times.

The lower 13 bits of the address bus discussed so far are used to access individual elements of the matrices for the similarity calculations. The remaining 6 most significant bits do not change during the calculation process and are kept in a 6-bit register. They are used to provide the address of the matrix being classified. This register only changes value when, at the end of the similarity calculations for one matrix, a forward pointer to the next matrix is loaded into this register.

There is also a 9-bit register which stores the addresses of the unknown and reference matrices being accessed for the update function. The 9 most significant address bits of the address bus are multiplexed between the 6-bit register plus 3-bit counter combination and the 9-bit register in order to select between the address for the similarity calculations and the address for the data required by the update adder unit.

The connection of the address bus to shared memory is different from the connection to dedicated memory because not all bits of the address bus are required for each memory segment. The three bits which select the

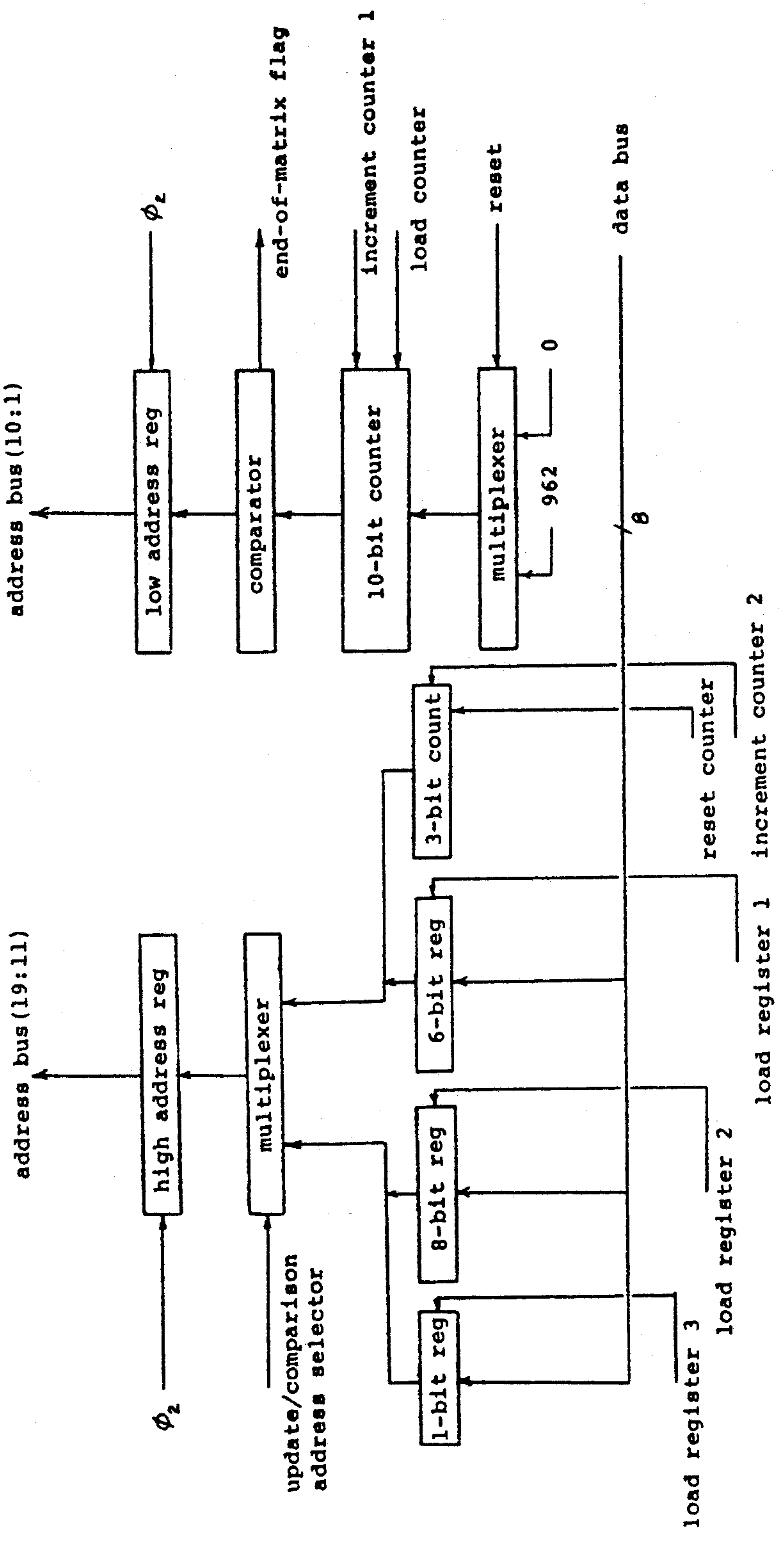


Figure 4-19: The register diagram for the memory address unit.

reference matrix in the dedicated memory are not needed for the shared memory, and the bit which selects the one of the two unknown matrices for each of the ECG leads is not required for dedicated memory. The address for the shared memory uses the bits {A(19:14) A(10:1)} and the address for the dedicated memory uses the bits {A(19:15) A(13:1)}. There is a separate line which selects between shared memory and dedicated memory.

4.2.6. The Control Unit

The function blocks presented so far provide the ability to operate upon data, but a circuit is necessary for co-ordinating the movement of data between the function blocks as well as in and out of the IC. This is the role of the controller. The controller issues a sequence of signals which enable each major component of the ECG processor to read and write data to and from the data bus and to initiate their intended functions.

The most general and common type of controller used in IC design is the finite state machine (FSM). As shown in Figure 4.20, the FSM uses a feedback path along with combinational logic to generate the desired sequence of digital signals.

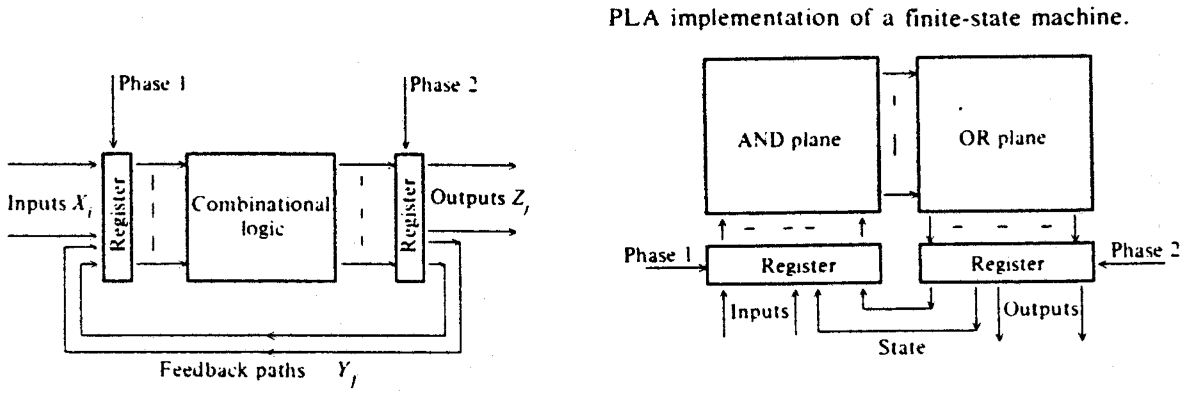


Figure 4-20: The block diagram of a finite state machine [14].

Each combination of outputs is associated with some FSM state which can be identified by the binary number formed by the feedback path. The FSM can change states every cycle with the state for the next cycle being determined by the current state and the input signals. When the input signals to a FSM are clocked, as they are in this design, its operation is totally synchronous and the outputs can change only during a clock signal transition. The asynchronous FSM has unclocked inputs and therefore, its outputs may change at any time during a clock cycle. A synchronous FSM is easier to design than an asynchronous FSM, but the disadvantage of the synchronous machine is that the output signals are always delayed by one clock cycle in response to the input signals. It is for this reason that, in the ECG processor, when a zero value is read from an unknown matrix element, one read from a reference matrix must be made even though the data is not used; the controller cannot react to the information that the unknown matrix element is zero in time to change the operation of the next cycle.

The combinational logic for an FSM could possibly be constructed from logic gates where the gates would decode every combination of inputs into the desired combination of outputs, but such an approach is complex and can be difficult to change and debug. For the controller being designed here, such complex logic functions are mapped onto a regular structure called the programmable logic array (PLA). A PLA is much like a read only memory (ROM) where every possible combination of outputs is stored and the input bits provide the address of the appropriate ROM location, but the PLA is more compact because it only stores the defined output combinations, and generally, the number of defined output combinations is only a fraction of the total number of possible output combinations. The advantage of the PLA structure for large combinational circuits is that because of the high degree of regularity, the layout generation for a PLA is highly automated and so changes in the control sequence are relatively easy to make.

For the ECG processor, a controller is necessary for:

- loading the address and control registers,
- incrementing the address counters,

- loading the recognition matrix data into the similarity calculation unit,
- writing the contents of the register file to memory,
- loading the update adder and writing its results to memory.

A total of 22 output signals from the controller are required to co-ordinate these functions, and to determine the sequence of change for these output signals, 5 input signals are used. These signals include information such as whether the data on the data bus is equal to zero, whether the similarity calculation unit has completed its function, and whether the last element of the recognition matrix has been read.

A state diagram for the ECG processor controller is given in Figure 4.21. The diagram identifies each of the 21 states with a brief statement. A complete state diagram would include the output state for each output, but listing all 22 output signals would make the diagram appear too complex.

The following listing is the controller description in the PEG finite state machine compiler language. This listing corresponds to the state diagram in Figure 4.21.

<u>The Controller Finite State Machine Description</u>

```
-- PEG decription of ECG processor PLA controller

INPUTS: zero eom end accum row8;

OUTPUTS: addl addh dbl lr1 lr2 lr3 load1 load2 uale incrow
         newmat share wr raml ramh uwr lvs lvd start update inc1 inc2;

-- power on reset begins at this point
--    (once reset is release a zero is written to shared memory)

reset1  : ASSERT dbl addh addl lr1 lr2 lr3 load1 load2;
        : ASSERT dbl share wr addh addl;

-- loop to wait for 'go-ahead' from host computer
```

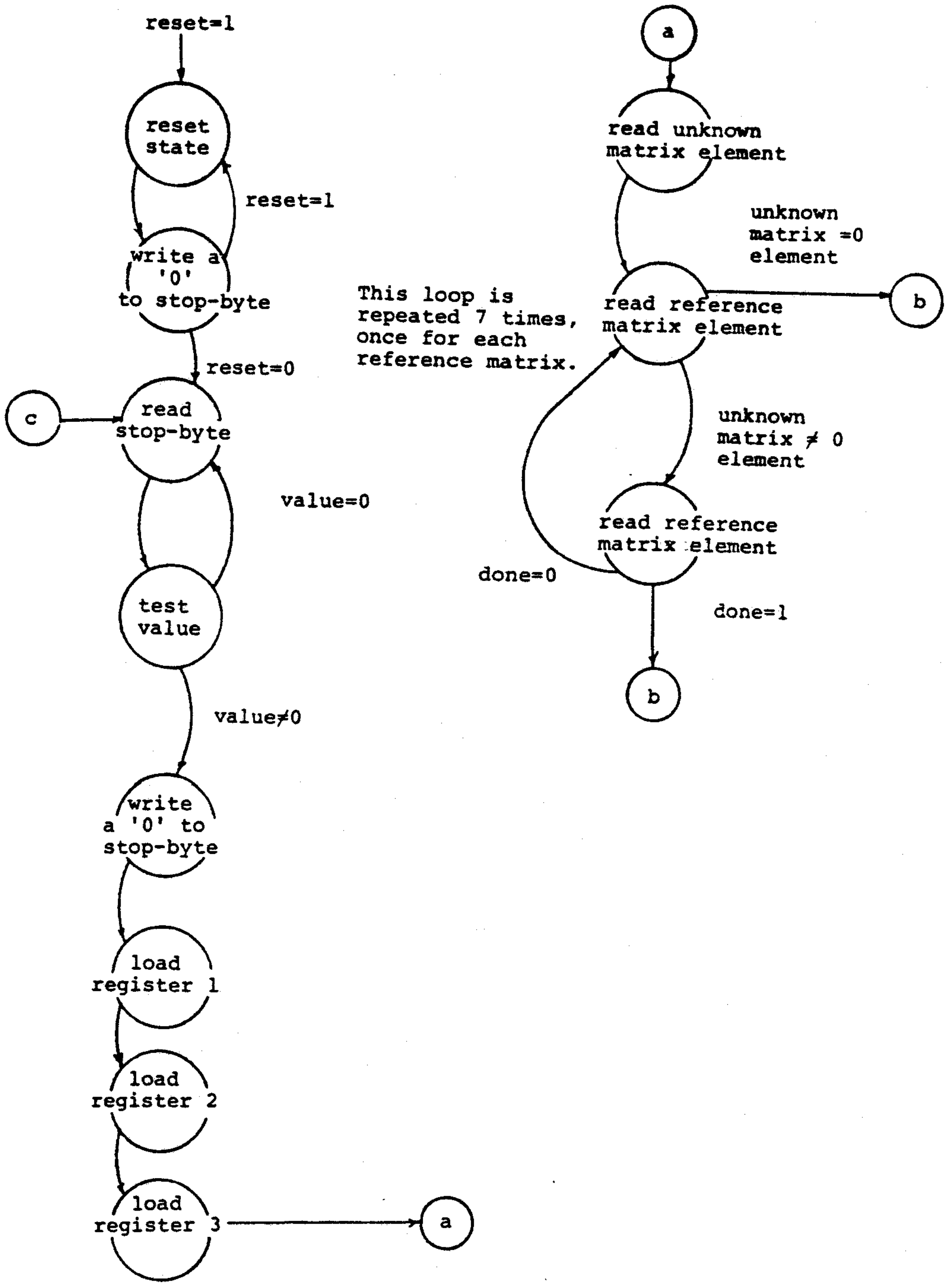

reset=1
reset state
reset=1
write a '0' to stop-byte
reset=0
c
read stop-byte
value=0
test value
value≠0
write a '0' to stop-byte
load register 1
load register 2
load register 3
a
a
read unknown matrix element
unknown matrix =0 element
b
This loop is repeated 7 times, once for each reference matrix.
read reference matrix element
unknown matrix ≠ 0 element
read reference matrix element
done=0
done=1
b

Figure 4-21: The controller state diagram.

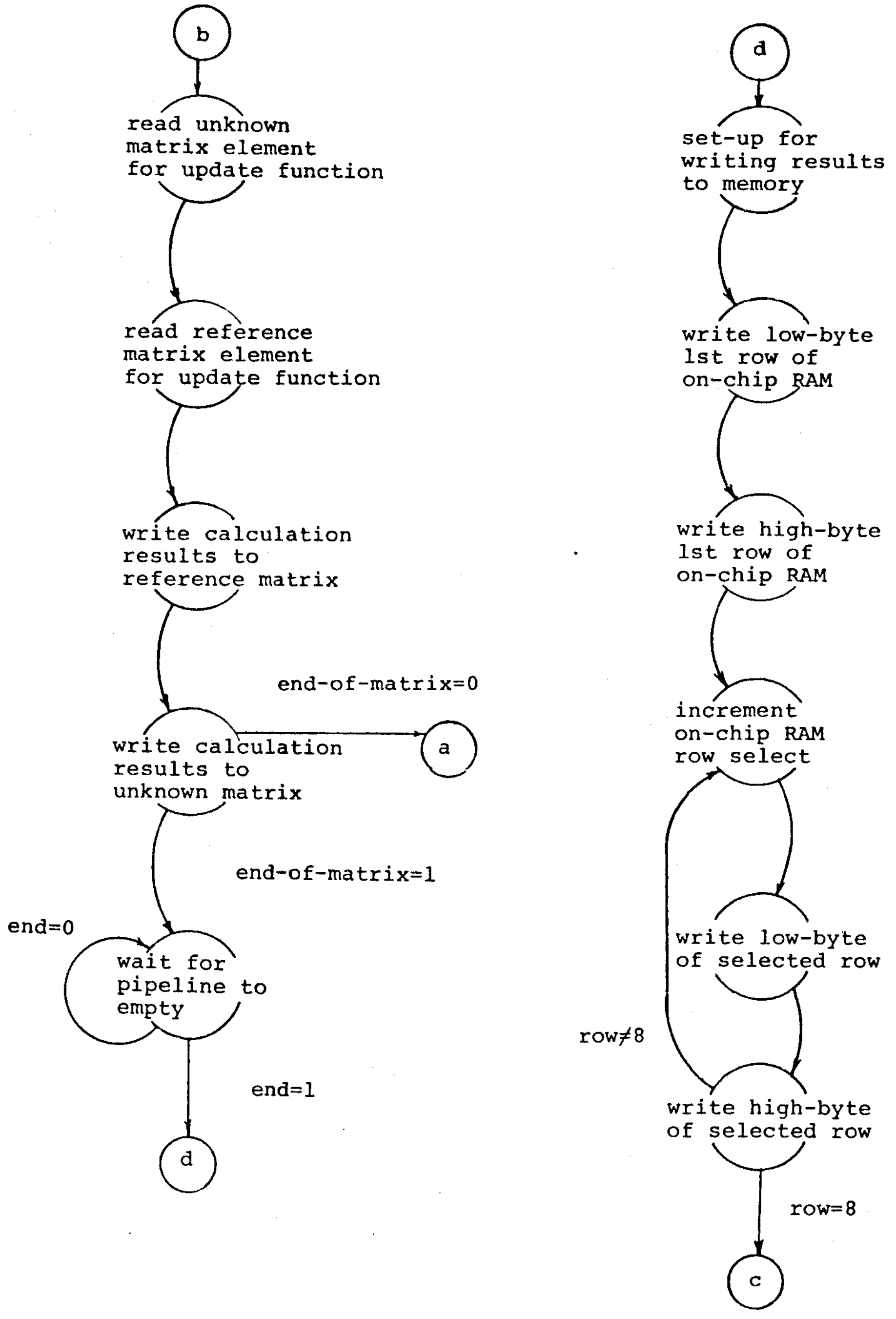

b
read unknown matrix element for update function
read reference matrix element for update function
write calculation results to reference matrix
end-of-matrix=0
write calculation results to unknown matrix
a
end-of-matrix=1
end=0
wait for pipeline to empty
end=1
d
d
set-up for writing results to memory
write low-byte 1st row of on-chip RAM
write high-byte 1st row of on-chip RAM
increment on-chip RAM row select
write low-byte of selected row
row≠8
write high-byte of selected row
row=8
c

Figure 4-21 continued

```
--     (a shared memory location is read every other cycle
--      until a non-zero value is detected.  Then zero is written
--      back to that location and continues with next part)
waiting : ASSERT share addl addh newmat;
        :  IF zero THEN waiting;
        : ASSERT dbl share addh addl wr incl;

-- the next 3 locations are pointers to data for the
--       next set of cacluations; they are read into 3 registers

        : ASSERT share addl lr1 incl;
        : ASSERT share addl lr2 incl;
        : ASSERT share addl lr3 load1 load2;

-- read a value from the unknown matrix.  If it is zero
--      then do not read values from reference matrices

matcal  : ASSERT share addl addh start;
        : ASSERT inc2 addh addl;
          IF zero THEN skipcal;

-- read 7 more values from reference matrices into calculator

calc    : ASSERT inc2 addh addl;
          IF NOT accum THEN calc;

-- this section performs the update matrix functions
--     A value is read from an unknown matrix and from a reference
--     matrix and some value is optionally written to the unknown or
--     reference matrix
--     matrix calculations continue at the next element unless
--     eom (end of matrix) signal is detected

skipcal : ASSERT update lvs share addh;
        : ASSERT update lvd addh;
        : ASSERT update uwr addh uale;
        : ASSERT update uwr addh uale share load2 incl;
          IF NOT eom THEN matcal;

-- wait for pipeline to empty before reading RAM

wait4end: IF NOT end THEN wait4end;

-- the 8 16- bit registers of the ECG processor are written
--      to shared memory one byte at a time

        : ASSERT newmat ;
        : ASSERT raml wr share addl addh incl;
        : ASSERT ramh wr share addl addh incl;
loopy   : ASSERT incrow ;
        : ASSERT raml wr share addl addh incl;
        : ASSERT ramh wr share addl addh incl;
          IF NOT row8 THEN loopy ELSE waiting;
```

The PEG program will generate a set of logic equations which are converted to a truth table by the program EQNTOTT. Besides producing a truth table, EQNTOTT will also attempt to reduce the number of minterms so that the size of the resulting PLA is minimized. TPLA then will generate a PLA layout from the truth table.

4.3. Support Hardware

To make the ECG processor part of the ECG analysis system, an oscillator for the ECG processor clock, a memory, and an interface circuit to the host computer bus are required. However, this section will concentrate on the memory circuits and leave out a discussion of the clock circuits and the bus interface circuit because the bus interface circuit depends on the type of host computer and the clock circuit for the ECG processor is generated externally in this version of the ECG processor. An external clock is used in the development of the ECG processor because the timing and voltage characteristics of the clock signal can be controlled better than if an on-chip oscillator circuit were used. Once the IC is tested with various clock signals, an on-chip oscillator can be added.

The most important aspect of the support hardware is the design of the shared memory. As far as the designer is concerned, the use of dual-port RAM would be the simplest method as the dual-port RAM has two sets of address and data lines. One set could be used for the ECG processor, and the other for the host computer. The problems associated with simultaneous access to common memory locations are resolved with circuitry built into the dual-port RAM. One example of dual-port RAM is the Hitachi HD63310 CMOS static RAM which has a storage capacity of 1k-by-8 bits [39]. A circuit using the dual-port RAM, as suggested in [39],

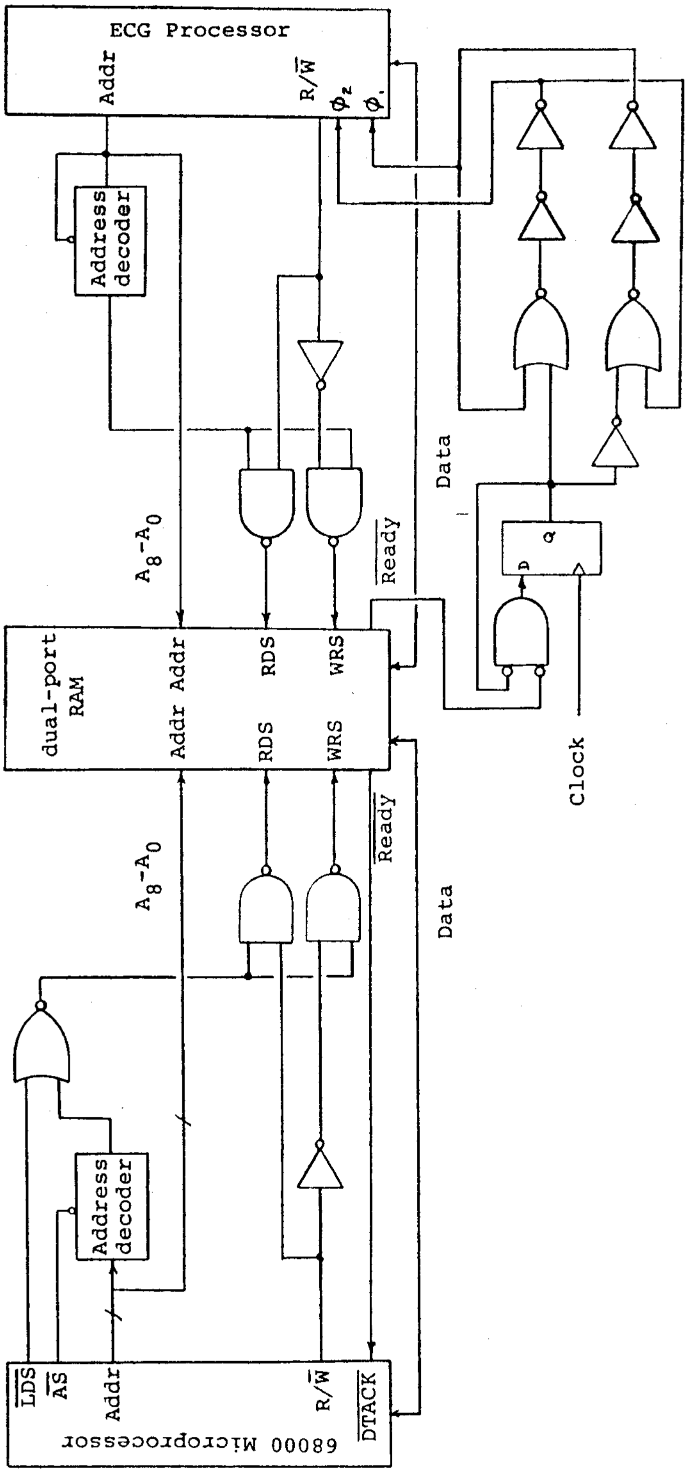


Figure 4-22: A dual-port RAM circuit.

would look like Figure 4.22. Either system makes a memory access by negating RDS for a read operation or WRS for a write operation and waits for $\overline{\text{Ready}}$ to go low when the memory operation is complete. A microprocessor like the 68000 has a $\overline{\text{DTACK}}$ pin which the $\overline{\text{Ready}}$ line can drive directly, but the ECG processor would have to have its clock halted. This can be done by preventing the clock signal ϕ_1 from going low until $\overline{\text{Ready}}$ went low. This is the purpose of the D-flip-flop in the ECG processor's clock circuit in Figure 4.22; the circuitry after the flip-flop is used to generate a non-overlapping two-phase clock from the flip-flop output.

The problem with the dual-port RAM approach is that the 64 48-pin RAM chips used in this design would consume considerable circuit board area. Another approach to the shared memory problem would be to multiplex the address and data lines to 64k-bytes of RAM. The key circuit for this approach would be the arbiter circuit which would allocate the use of the shared memory between the two independent processors. In this design, it would allocate the memory to the first processor which demanded it. This should be a straight forward design problem except for the synchronisation failure problem mentioned in Chapter 3. This problem exists in the arbiter because it is possible for two asynchronous circuits to generate a request simultaneously, and while the chance of this occurring is small, both the host computer and the ECG processor would have to wait until the arbiter made a decision between the two requests. The arbiter may take an indefinite amount of time to make this decision. The circuit in Figure 4.23, based on the nMOS interlock element from [14], shows one way of implementing the arbiter function. To use this circuit, the request lines, $\overline{\text{REQ}}$, are negated when the host computer or the ECG processor attempt to access the shared RAM, and the mutually exclusive outputs, ACK1 and ACK2, signal the multiplexers to connect the RAM to either the host computer bus or the ECG processor.

The results of SPICE simulations of the arbiter circuit using CMOS3 simulation parameters are shown in Figure 4.24. The top part of the figure

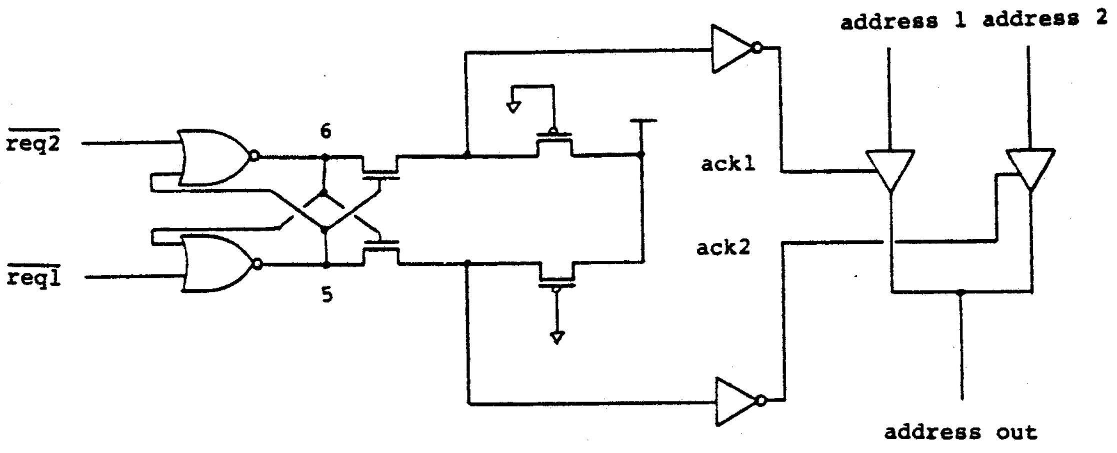
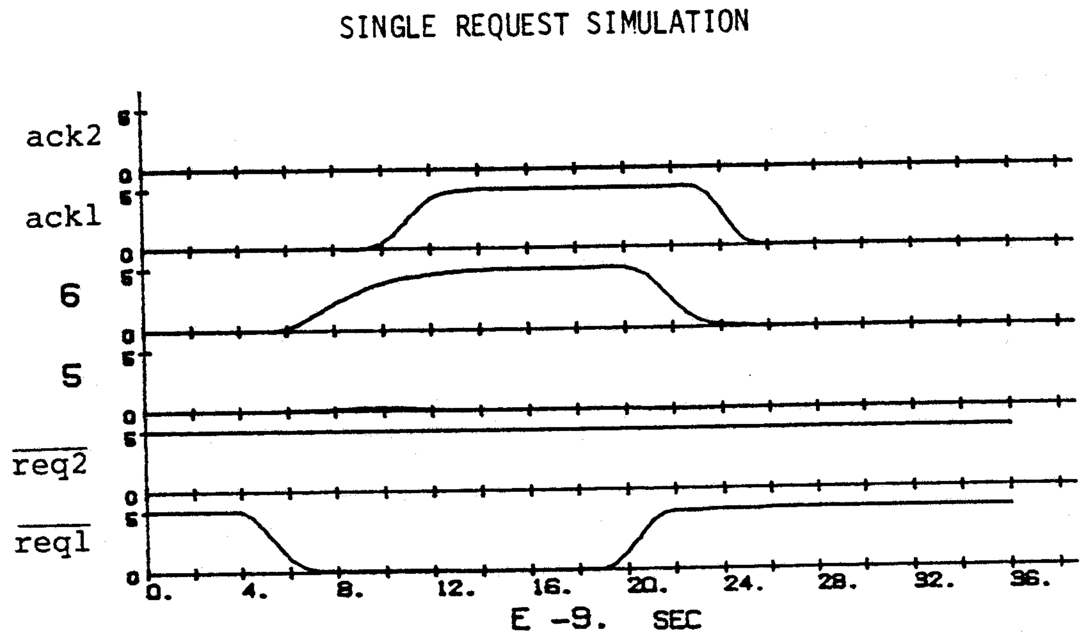


Figure 4-23: A CMOS arbiter circuit [14].

shows that 4 ns after $\overline{\text{REQ1}}$ requests a memory access, ACK1 signals the multiplexers to connect the memory to the device which made the request, and the bottom part of the figure shows the results of an attempt at a simultaneous access. According to the simulation, the arbiter finally selects ACK2 after about 25 ns. This simulation is just a demonstration of the arbitration process and may not be accurate because in the metastable state, the arbiter is extremely sensitive to circuit capacitances and transistor gains and both could have different values than an actual circuit. That the SPICE simulation did leave the metastable state means the simulation was somehow biased towards ACK2 even though the circuit as described in the SPICE input was perfectly symmetrical. The simulation does demonstrate, though, that ACK1 and ACK2 are mutually exclusive and that they could be used to drive the multiplexers in the shared RAM circuit. It should be noted that ACK1 and ACK2 are not the same as the Ready line of the dual-port RAM because ACK1 and ACK2 change state at the beginning of the memory access cycle while Ready changes state when the memory access was complete. This means that when using ACK1 to generate a signal like the $\overline{\text{DTACK}}$ signal for the 68000 microprocessor, memory access time would have to be accounted for.



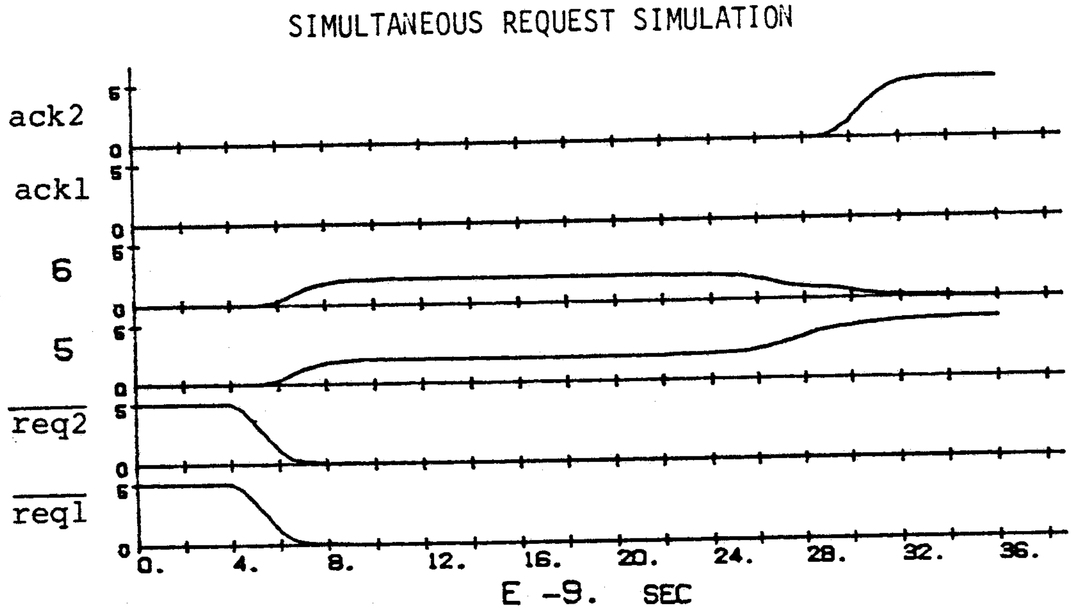


Figure 4-24: The arbiter circuit simulation results.

As long as dual-port RAM is limited to a small size like 8k-bits, the multiplexer address and data line approach to shared memory would consume less circuit board area. Further development of the ECG processor could include the arbiter circuit, the multiplexers, and possibly even a dynamic memory refresh controller on the ECG processor chip itself.

4.4. Support Software

This section discusses some of the aspects of the host computer software necessary for the use of the ECG processor and is based upon material presented in [40]. This software is required because by itself, the ECG processor cannot perform any useful functions; it needs the host computer to supply the address of the recognition matrices to be compared and the addresses of the matrices to be modified by the update functions. The discussion on software is broken down into several process descriptions where each process may be considered as a single program, and these programs have been given different levels of priority so that the host computer can keep up with the continuous stream of ECG data. For instance, the highest level process, the mapping of ECG data into a recognition matrix, cannot be delayed because the ECG data arrives continuously at 500 Hz and any delays would cause a loss of ECG data.

1. The high level process

This is an interrupt driven process which is called for every sample point on the ECG signal. To minimize the interrupt processing overhead, data should be taken for all ECG leads for every interrupt. This gives a 500Hz interrupt rate due to ECG signal sampling. For each ECG lead, this process should:

- read the ECG sample value and its corresponding Hilbert transformed value,
- map this data into the recognition matrix using a method described in Chapter 2,

- at the end of a heartbeat for a particular lead, enter the unknown matrix into the classification queue. Placing a matrix into the classification queue involves placing the address of the new matrix in the forward pointer at the end of the last matrix entered in the queue.

To carry out the mapping process, the following data structures are required:

- a table of pointers to the base addresses of the unknown recognition matrices; there is a pointer for each ECG lead,

- a table a scale factors for each lead. The scale factor normalizes the peak voltage of the ECG signal to fit inside a recognition matrix,

- a list of matrices entered into the classification queue.

The table of pointers is necessary because each lead could be mapped into one of two possible unknown matrices. When one unknown recognition matrix is completed at the end of a heartbeat, it is submitted to the classification queue and mapping for the next heartbeat's ECG data continues in the alternate unknown matrix. This change in the unknown matrix is accomplished by changing the pointer in the pointer table.

Depending upon the speed of the host computer and the complexity of the mapping function, this process is expected to be the most time consuming function on the host computer. An assembly language program for the mapping function was written for the 68000 microprocessor to determine an approximate host computer load for this process. Assuming that the beginning and end of each heartbeat and the peak signal amplitude were determined by hardware other than the host computer, the estimated time to map the ECG data for 32 leads with an 8MHz microprocessor clock would be 2.2ms per interrupt. As sample points are taken every 2ms, this means over 100% of the processor time would be used for this mapping process, and that the capability of the ECG processor currently exceeds the ability of a host computer based upon the 68000 microprocessor.

2. The medium level process

This process polls the 'stop-byte' locations in shared memory in order to determine which similarity calculations the ECG processor has been completed. After a valid forward pointer has been written at the end of a recognition matrix, the 'stop-byte' locations are changed by the host computer to a non-zero value. As the ECG processor encounters these 'stop-bytes', they are changed back to zero. Thus, if the host computer keeps track of which matrices have been entered into the classification queue, then by checking these matrices for a 'stop-byte' value of zero, the computer can determine which unknown matrices have been through the similarity calculations. Once this process has determined which similarity calculations are complete, it can read the values computed by the ECG processor and complete the similarity calculations by multiplying them by $(A_T B_T)^{-1/2}$. The estimated load of this process for an 8MHz 68000 microprocessor is 0.13% of available processor time per ECG lead assuming a 60 beats per minute heart rate. This shows that implementing the complete similarity calculation on the chip would not substantially improve the ECG analysis performance. After the similarity values have been calculated, they are passed to lower level processes for use in the ECG analysis program.

3. Low level processes

These are the background processes which are expected to be able to keep up with the data produced by the higher level processes. The low level processes include:

- using the results of the similarity calculations along with other information extracted from the ECG signal, to classify the heartbeat. The host computer may also take some action like operator notification as a result of the classification,

- schedule matrices to have their elements reset to zero in preparation for data produced by the next heartbeat. As well, unknown matrix data may be added to a reference matrix. These are functions of the ECG processor's update function and are organized in an update queue in a similar manner as the classification queue.

- perform input and output to peripherals such as the disk drive and the operator's terminal.

The inter-action among the processes is complex and the actual performance of such a system is difficult to predict because the computational requirements for all the processes have not been determined. However, the mapping function is expected to be the most time consuming because of the large volume of data produced by 500Hz sampling of the ECG leads which must also be scaled through a multiplication. Other functions like the classification of the heartbeat, while complex, are done relatively infrequently with a rate of about once per second per ECG lead. This assumes, however, that most of the other ECG analysis functions such as identifying the P, Q, R, S, and T waves are not performed by the host computer but by external hardware.

4.5. Test Patterns for the ECG Processor

After the ECG processor has been fabricated, it is necessary to test the IC as thoroughly as possible to ensure that a faulty chip is not made part of a completed ECG analysis system. The most thorough test of the chip would be to apply all possible input combinations under worst case conditions and compare the output to the output of a known good circuit. However, with 961 8-bit elements per recognition matrix, testing all possible ways of comparing two recognition matrices would require 256^{961}x256^{961} input combinations, a number too large to be realistically considered. A more realistic test procedure takes advantage of the property that trying all input combinations is generally not necessary because many input combinations are redundant.

One method for generating the input combinations, or test vectors, to test the nodes of a purely combinational digital circuit is based upon the D-algorithm. To do this, for each node in the circuit, the D-algorithm assumes that a node is faulty and is stuck at either a logic '0' or a logic '1', and then it attempts to find the input combination which meets two requirements: one requirement is that the input forces the node to assume the value opposite to the state at which it would be stuck; and the second requirement is that input combination makes the circuit output dependent upon the state of the node being tested, that is, if the node is faulty, the output must be different from the case where the node is not faulty. The D-algorithm is not a perfect means of generating test vectors as it models only two kinds of faults, the stuck-at-1 fault and the stuck-at-0 fault, which are not always good assumptions for CMOS circuits [41]. As well, the D-algorithm considers only single faults.

The D-algorithm could not be directly applied to a circuit like the ECG processor because it is a sequential and not a combinational circuit. However, one technique has been developed, called the Level Sensitive Scan Design method [24], which attempts to make a sequential circuit testable using combinational circuit test techniques by making all memory elements in the circuit part of a single shift register. These memory elements would be designed to perform two functions. In the test phase of the circuit they act as a shift register; an input combination is shifted into this shift register to provide inputs to all combinational logic circuit elements, and then the outputs of these logic circuits are read into the shift register and shifted out to external test equipment for comparison to a value generated by a good chip. In the normal operating mode, these memory elements function as registers, flip-flops, and latches.

In the case of the ECG processor, designing a circuit which was easy to test was not found to be difficult and neither of the above design for testability techniques were required to improve the circuit testability. This was for three reasons.

1. One reason was that most of the ECG processor's registers, counters, and adders were located logically close to output pins. The address registers and counters were easily set-up from the data bus, and observed through the

address bus, and the internal RAM and update adder were accessible through the data bus.

2. The second reason for the circuit being easy to test was because of the small input word size for the functions in the ECG processor. For example, only 256 different inputs were required to fully test the square-root extractor and multiplier, and the largest address counter was only 10-bits meaning it was completely tested in 1024 clock cycles.

Trying this approach for the 16-bit adder does not work because of the input word size; 2^{32} input combinations would be necessary. But trying all input combinations is not necessary for a complete test. The ability of all circuit nodes to change states can be checked by observing the sum of four input combinations. If A and B were the inputs to the adder, the input combinations would be:

(A)	(B)	(Expected Sum)
(0000 0000 0000 0000)	(0000 0000 0000 0000)	(0000 0000 0000 0000)
(0111 1111 1111 1111)	(0000 0000 0000 0001)	(1000 0000 0000 0000)
(1010 1010 1010 1011)	(1010 1010 1010 1010)	(0101 0101 0101 0101)
(0101 0101 0101 0101)	(0101 0101 0101 0101)	(1010 1010 1010 1010)

3. The third reason for the ECG processor being easy to test was that the accumulating registers can be used as a check sum registers, that is, after a series of inputs which are known to give a certain result are entered into the ECG processor, the results of the computations are accumulated in RAM and can be recalled and compared to a set of 'correct' results. If there is an error at any point in the calculations, it should show up as an unexpected accumulated value stored in RAM. The check sum approach is not a guaranteed method of error detection because errors could accumulate in a way to produce the correct value. The probability of this occurring in

occurring in the ECG processor has not been calculated, but for a 16-bit register, the odds of such an event, assuming a random value produced by a faulty circuit, would be 1/65,536. This probability is reduced even further because eight separate accumulation operations are performed for each unknown matrix classification.

A set of test vectors has not been developed for the ECG processor because their development would require considerable computer processing time; the simulation of 40 clock cycles of the ECG processor required about 20 minutes of MicroVAX II CPU time, and at least 980 clock cycles would be used to try out one test vector. What is presented instead is a description on how the test vector generation problem could be approached.

To test the PLA controller and the address counters, it is only necessary to cycle through the unknown matrix comparison cycle once. This exercises all the controller states, puts the address counters through their full address range, and allows all the data necessary to completely test the square-root extractor to be read from external memory. It is also possible to provide the data to completely test the multiplier, but the data for the multiplier must account for the fact that the data first passes through the square-root extractor before reaching the multiplier.

To test the on-chip RAM, the final values in the accumulating registers should be made to equal alternating bit patterns like, for example, {0101 0101 0101 0101} for even memory locations and {1010 1010 1010 1010} for odd memory locations. Then the complete matrix calculations should be repeated to give results which toggle every memory bit location in RAM. This ensures that all memory locations have the ability to store a logic '1' and a logic '0'. The alternating bit pattern is selected so that shorts between memory locations are detected.

The address registers and the update adder functions can be tested through a repeated reset of the ECG processor. Since the first action taken by the ECG processor after a reset is to load forward pointers into the address registers, several different values can be loaded into these registers through the repeated application of the reset signal. This avoids having to cycle through a complete matrix calculation before changing these registers. The contents of these registers are readily observed through the address lines. The repeated reset approach also allows testing of several update adder functions without having to wait for a completed matrix calculation for each function test.

These tests can be performed relatively quickly. Two complete unknown matrix calculations take about 6ms with a 2.5MHz clock, and the testing of the address registers and the update adder add a small amount to this test time.

4.6. Summary

The first section of this chapter covered the definition of the algorithm to be implemented on the IC, the selection of an interface between the ECG processor and the host computer, and the development of the ECG processor memory map.

In the second section, a register level description was given for each of the major function blocks of the ECG processor. Along with these descriptions were provided alternate design approaches and reasons why the selected approach was taken.

The next two sections described the external hardware and the host computer software which would be necessary to make the ECG processor part of an ECG analysis system.

The final section told how the test vectors required for testing a fabricated ECG processor could be developed to provide a thorough test of the ECG processor's internal circuitry.

The chapter should have given the reader an understanding on how to use the ECG processor as well as an understanding of how the computations are executed on the IC.

A detailed block diagram of the ECG processor based upon the completed Netlist description is presented in Appendix A. This diagram includes the signal lines and circuitry needed to tie together all the functions presented in this chapter.

Chapter 5
Circuit Design, Simulation, and Testing

The purpose of this chapter is to present the circuits at the transistor level for the functions defined in Chapter 4. While Chapter 4 was concerned with the design of the ECG processor at the algorithmic level, this chapter concentrates on the logical and electrical circuit development. Together, these two chapters provide a complete description of the ECG processor.

This chapter is divided into five sections. The first section defines the simulation parameters with which the circuit simulations are performed. To show that we can have confidence in these simulation parameters, a comparison is made between the simulation of a circuit and its actual measured performance. The second section presents the basic building blocks for the construction of the ECG processor. These building blocks are collected into a cell library which includes common logic functions such as the inverter, the NAND, NOR, and XOR gates and larger circuits such as counter cells, shift-register cells, and add/subtract cells. Other more specialized circuits like input pad protection, output pad circuits, and memory cells are also discussed. The third section presents the test results for two working design submissions: the accumulating adder, and the random access memory. The fourth section presents a simulation of the complete ECG processor which shows, as far as the simulation can predict, that the design performs its intended function. The fifth and final section discusses the expected performance of the ECG processor and makes comparisons between it and other techniques for computing the similarity between two recognition matrices.

5.1. The Simulation Parameters

Important to the accurate simulation of an electrical circuit are good models for the circuit elements and accurate parameters for these models. For this design, the circuit models are already available as part of the simulation programs, but the model parameters, which are directly influenced by the manufacturing process and the layout of the circuit, must be determined especially for Northern Telecom's CMOS3 fabrication process. The process dependent parameters can be determined either by using a description of the fabrication process or by measurements taken directly from fabricated circuit elements. The layout dependent parameters like transistor gate dimensions are taken from an examination of the layout.

In this section, the process dependent parameters for SPICE simulation are examined and then used to develop the parameters for the RNL simulator. In the final part of this section, the results of simulations for a ring oscillator circuit are compared to its measured characteristics.

5.1.1. The SPICE Simulation Parameters

The parameters used for SPICE simulation as provided by the Canadian Microelectronics Corporation are listed in Table 5.1.

```
.MODEL NCHAN NMOS LEVEL=3,TOX=500E-10,NSUB=1.7E16,XJ=0.6E-6,VTO=0.7,
+                 UO=775,THETA=0.11,GAMMA=1.1,KAPPA=1,ETA=0.05,
+                 VMAX=1E5,NFS=0,PB=0.7,JS=1E-5,LD=0.35E-6,RSH=25,
+                 CJ=4.4E-4,MJ=0.5,CJSW=4E-10,MJSW=0.3,CGSO=3E-10,
+                 CGDO=3E-10,KP=50E-6,CGBO=5E-10
.MODEL PCHAN PMOS LEVEL=3,TOX=500E-10,NSUB=5E15,XJ=0.5E-6,VTO=-0.8,
+                 UO=250,THETA=0.13,GAMMA=0.6,KAPPA=1,ETA=0.3,
+                 VMAX=0.7E5,NFS=0,PB=0.6,JS=1E-5,LD=0.25E-6,RSH=80,
+                 CJ=1.5E-4,MJ=0.6,CJSW=4E-10,MJSW=0.6,CGSO=2.5E-10
+                 CGDO=2.5E-10,KP=16E-6,CGBO=5E-10
.OPTIONS DEFAD=35E-12 DEFAS=35E-12
```

Table 5-1: SPICE parameters for the CMOS3 process.

These parameters can be grouped into three classes: one class defining the DC characteristics of the transistor; the second class defining the transistor capacitances which are important for modelling the switching characteristics; and a third class which includes the manufacturing process parameters like substrate doping levels and gate oxide thickness. A brief description of the significant parameters in each of these classes is provided here beginning with parameters affecting the DC characteristics:

- VT0 (volts) = the zero-bias threshold voltage. The threshold voltage is defined as the voltage where a channel is created between the drain and source and current begins to flow through the channel. Actually, there is some drain to source current flow when the gate is below the threshold voltage and the current flow increases exponentially as the gate voltage increases past VTO. Another definition of threshold voltage for an nMOS transistor is the gate voltage which makes the concentration of electrons on the silicon surface under the gate equal to the concentration of holes deep in the substrate.

- KP (ampere/volts^2) = the transconductance parameter which when multiplied by the transistor gate width/length ratio gives the gain factor of the transistor.

- GAMMA (volts$^{1/2}$) = the bulk threshold parameter. This is used to determine the effect of source-to-substrate voltage on the threshold voltage.

- PB (volts) = the p-n junction potential between the source and substrate, and the drain and substrate.

For the transistor's switching characteristics, the following parameters are used to provide data for inter-terminal capacitances.

- CJ (Farads/meter^2) = the zero-bias capacitance between the bottom of the source and drain diffusions and the substrate.

- CJSW (Farads/meter) = the zero-bias sidewall capacitance between the sides of the source and drain diffusions and the substrate.

- CGSO (Farads/meter) = the gate-to-source overlap capacitance

- CGDO (Farads/meter) = the gate-to-drain overlap capacitance

- CGBO (Farads/meter) = the gate-to-substrate overlap capacitance

Besides these parameters, there are process dependent parameters like the gate oxide thickness (TOX), the depth of source and drain diffusions (XJ), and the substrate doping level (NSUB) which are used by SPICE to compute the gate capacitance and any other parameters which are not provided by the user.

Layout dependent parameters like gate width (W), gate length (L), drain and source areas (AD and AS), and drain and source perimeters (PD and PS), can be specified for each individual transistor. SPICE then uses these values to compute the total drain and source capacitances and the transistor gain.

One note should be made about how AS and AD are used in this thesis because the diffusion areas are generally a significant part of circuit capacitance. When generating a SPICE circuit listing from a layout, AS and AD are set to zero because the capacitances due to drain and source diffusions are computed by the program generating the SPICE listing. They are included as ideal capacitors, and in order not to count this capacitance twice, AS and AD must be set to zero. However, there are minor disadvantages to this approach. This approach to computing drain and source capacitance has a small effect on the computed p-n junction leakage currents when the transistor is in the off state since AS and AD are used in these calculations, and it does not account for the voltage dependent characteristics of these capacitances. To remedy these problems, a program to convert a '.sim' file to a SPICE listing would have to be written which computed the drain and source areas and included AS and AD with the transistor specifications.

On the other hand, a SPICE listing generated from NETLIST or by hand has AS and AD set to a default value of $35.\text{x}10^{-12}\text{m}^2$ which equals the drain and source areas of a minimum size transistor. This default value is used because the NETLIST to SPICE conversion program does not compute

AS or AD and there are no layer areas given in a NETLIST description from which capacitances can be computed. The use of a default drain and source area is one way to estimate these capacitances.

An accurate model of the transistor is one requirement for a good simulation; accurate models for the interconnecting wiring is another. For the simulators in this project, the only effect of interconnecting wiring which is considered is the wiring capacitance to ground. Inter-node capacitance, resistance, and transmission line effects are ignored because the circuit extractor, MEXTRA, is unable to produce from the layout the required information for these effects. What MEXTRA does generate (the '-o' option is used with MEXTRA) is information about the total area and perimeter length for each layer attached to a node. Both area and perimeter information are included to account for both the parallel plate capacitance and the edge effect capacitance. The program SIM2SPICE then uses the user-defined information in Table 5.2 to convert the layer information generated by MEXTRA into the circuit capacitances for the SPICE circuit description. The values in Table 5.2 are provided by the Canadian Microelectronics Corporation [42].

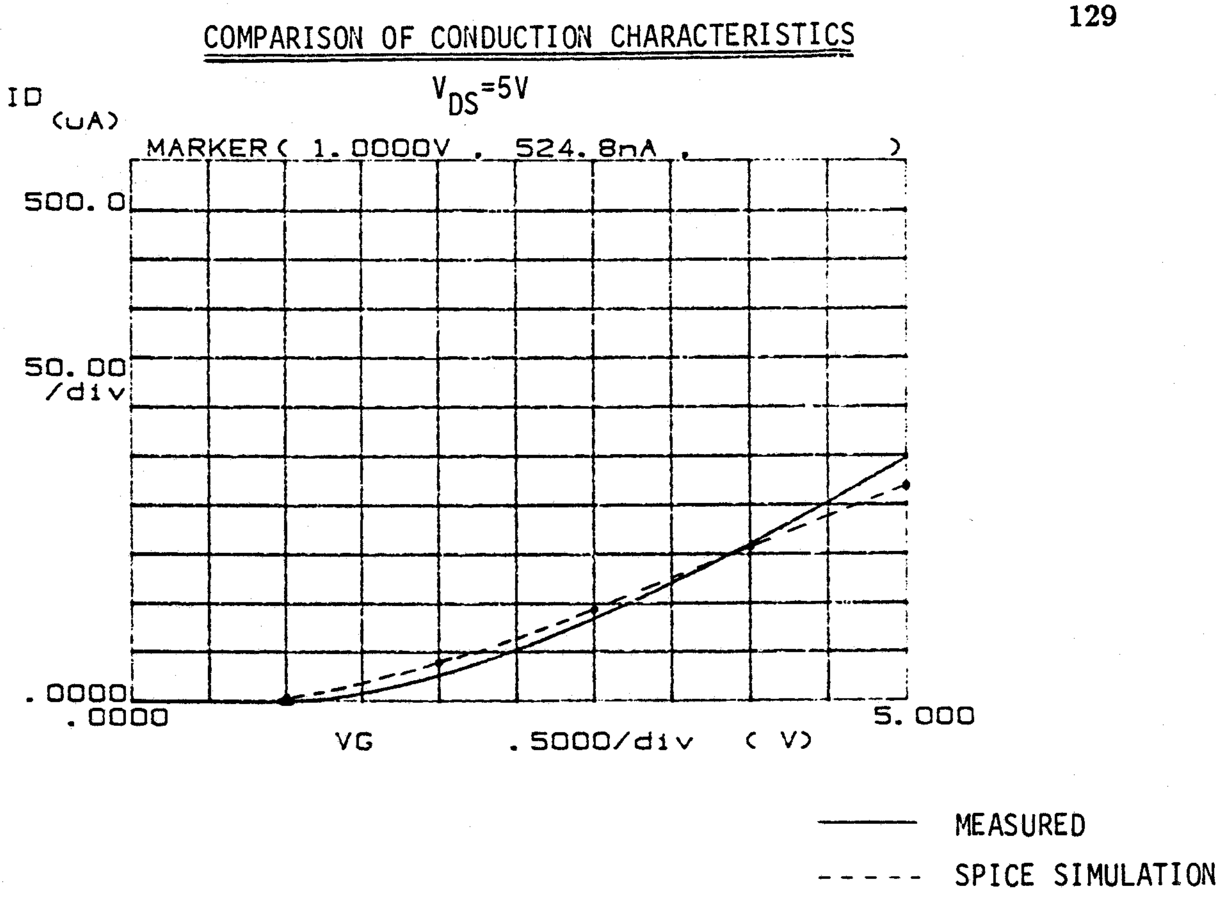
Sim2spice layer capacitances

Polysilicon area capacitance	60 fF/μm^2
Polysilicon perimeter cap.	20 fF/μm
Diffusion area capacitance	300 fF/μm^2
Diffusion perimeter cap.	400 fF/μm
Metal area capacitance	27 fF/μm^2
Metal perimeter capacitance	40 fF/μm

Table 5-2: Sim2spice layer capacitance constants

In Figure 5.1, a comparison of the measured characteristic curves for a transistor fabricated with the CMOS3 process is compared to the curves predicted by a SPICE simulation. The length and width of the transistor gate are 3μ. There is a general match between the two sets of data, but the error is largest for low and high gate voltages. For example, at V_G=5V and V_{DS}=5V, the measured channel current I_D is $248\mu A$ while the SPICE value is $220\mu A$. The simulation error in this case is -11%. At the low gate voltage of 1V, the measured current is 525nA while the SPICE predicted value is 4060nA: an error of 1500%. This last error, however, is not as serious as the percentage implies. These low currents only affect how long charge may be stored on a node capacitance and do not affect the time delay computations. As well, these low currents are difficult to measure; they are affected by temperature and ambient light levels. The previous error with V_G=5V, though, does affect the time delay estimations, and from this comparison, it is expected that SPICE simulations predict a slower than actual circuit operation. Besides these errors in the normal operating region, the SPICE simulation results begin to diverge from the measured characteristics when the drain-to-source voltage is greater than 10V.

Whether these errors are due to model limitations, the selection of parameter values, or a measurement based on only one sample is not known. The parameter values could be modified to give a better fit of the characteristic curves, but this has not been attempted because the average characteristics of a large sample of transistors should be used as a reference. However, if the transistors were properly characterized, one paper, reference [43], describes a technique where a computer program searches for an optimum fit between the measured and SPICE simulation curves by changing several model parameters.



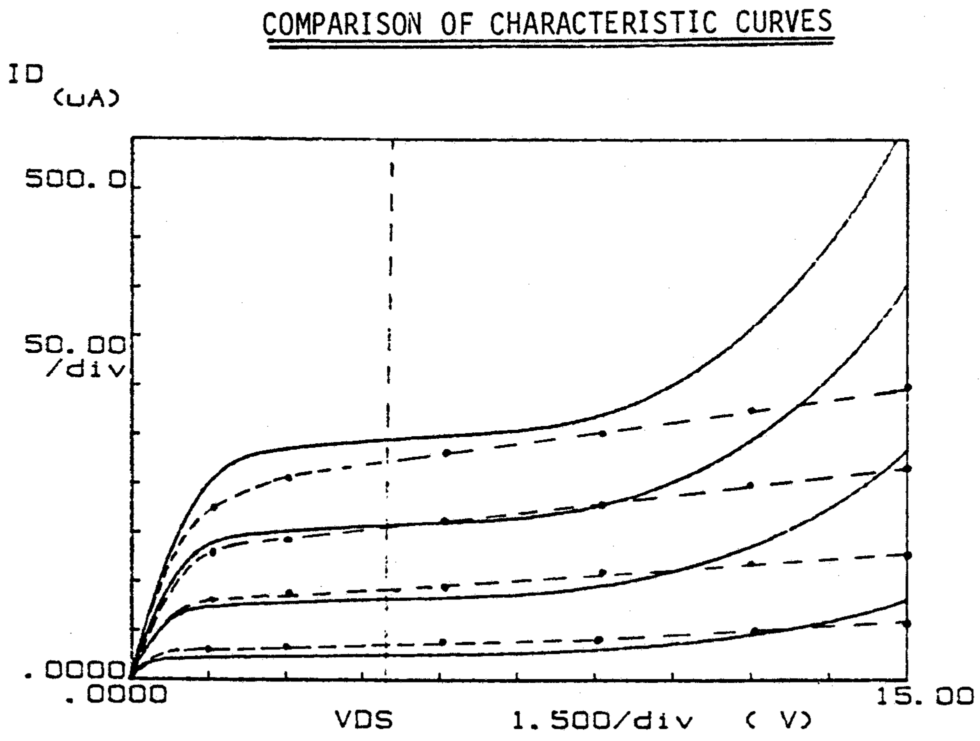


Figure 5-1: Comparison of simulated and measured DC characteristics.

5.1.2. The RNL Simulation Parameters

The RNL transistor model, though much simpler than the SPICE model, still needs to be calibrated for a specific process, and since the Canadian Microelectronics Corporation does not provide the model parameters for RNL as they do for SPICE, this section will describe the method by which the RNL model was calibrated.

Before describing the calibration method, a brief summary of RNL is presented here.

RNL models a circuit by using a resistor in series with a switch to model a transistor and by representing all circuit loads as a capacitance to ground. The state of each node is recorded as being either a '0', '1', or 'X' where '0' means the node has a voltage below a specified low-voltage threshold, '1' means the node voltage is above a specified high-voltage threshold, and 'X' means the node voltage is unknown. The capacitance for each node is found the same way as described in the previous section with the layer areas being multiplied by a constant to obtain a capacitance. The selection of a transistor resistance, however, is more complex and three different resistance values are used in finding its switching delay. The 'static' resistance value is used to predict the final state of a node; all the 'static' resistances connected to a node form a voltage divider network between the power supply and ground. The final state of the node is determined by computing the node voltage from this voltage divider network and assigning the state as specified by the threshold voltages. While the 'static' resistance is good for calculating the final steady-state voltage level of a node, it does not necessarily give a good prediction of the transistor switching times, and for this reason the 'dynamic-high' and 'dynamic-low' resistances must also be specified for each transistor. These values are used to predict when a node will change state after a transistor turns 'on' or 'off'. For example, if a node is at logic '1' and a transistor turns 'on' which, when using the static resistance analysis, RNL predicts will make the final

final state of the node equal to '0'. The time delay until the node actually changes from '1' to '0' is determined by the node capacitance multiplied by the 'dynamic-low' resistance to ground. The other switching resistance, 'dynamic-high', is the resistance of the paths to V_{DD} and is used for a '0' to '1' transition delay calculation. These dynamic resistances are divided into two classes because pMOS transistors are generally poor conductors of low logic levels and nMOS transistors are poor conductors of high logic levels, and as a result, low-to-high transition times do not necessarily equal high-to-low transition times.

To determine what these resistance values should be, tests based upon SPICE simulations were made. For the static resistances, a voltage divider network like that shown in Figure 5.2 was used; the effective static resistance equalled the drain to source voltage divided by the drain to source current.

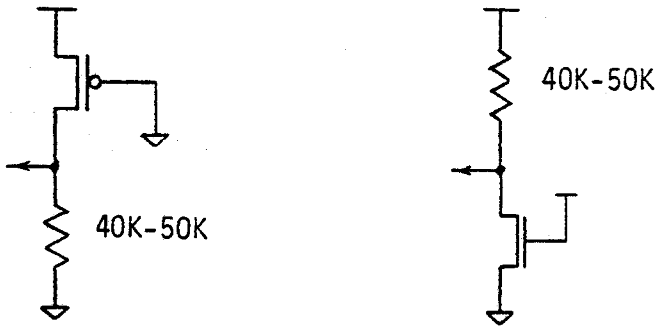


Figure 5-2: Voltage divider network for measuring static resistances.

This static resistance value was difficult to determine accurately because it was strongly dependent upon the biasing voltages. For these tests, the drain to source voltage was kept low, in the range of 0.2V to 2V, because this was the 'linear-resistance' operating region and also because they were the lowest effective resistances encountered in normal operation. Using these resistances provided the worst case analysis of pull-up and pull-down transistors.

For measuring the dynamic resistances, the circuits in Figure 5.3 were used.

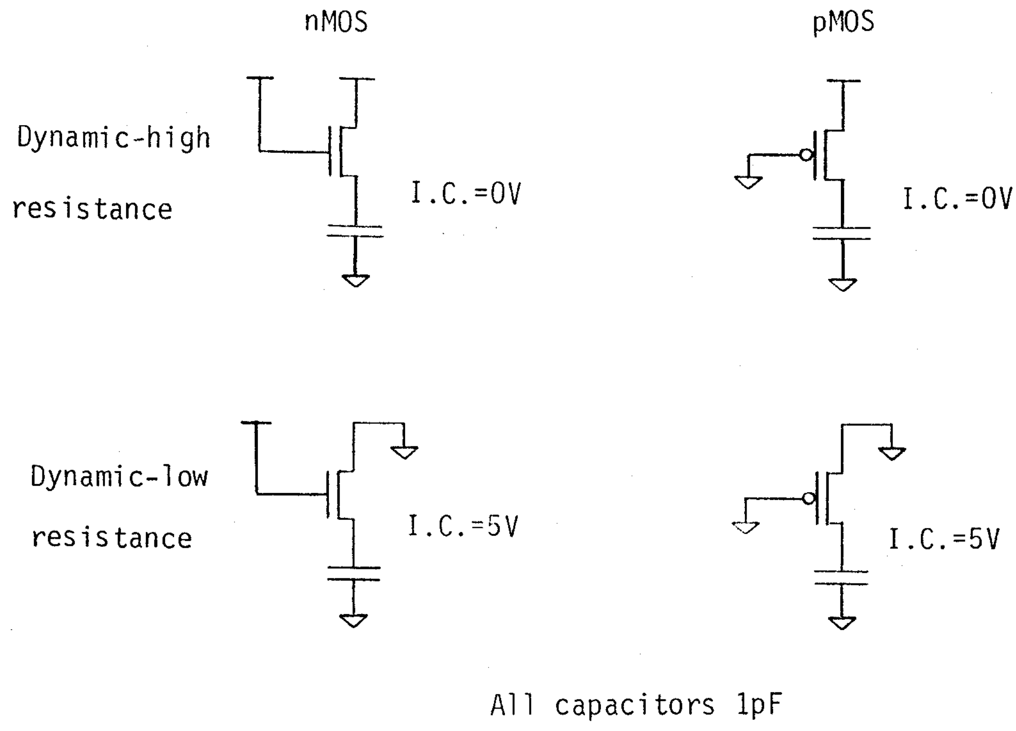


Figure 5-3: Circuits for measuring the dynamic resistances.

The charge of a large capacitor was allowed to build-up or discharge through a transistor. The resulting curves of the capacitor voltage versus time were not exponential, but for the purposes of determining the dynamic resistances, they were assumed to be exponential. The effective dynamic resistance could then be determined by measuring the RC time constant. In the case of a decaying node voltage, 't' and 'V(t)' were taken from the SPICE output and the effective dynamic-low resistance calculated by:

$$R_{dyn-low} = -\frac{t}{C}\log\left[\frac{V(t)}{5}\right]$$

The results of these tests are recorded in Figure 5.4. This figure is actually the file used by PRESIM to generate a circuit description for RNL.

```
cat net2_config
;
; layer capacitances fo NT CMOS3 process
;
; note: capacitances have been reduced to 70% of their original
;       value because RNL calculates delay time based on
;       R * C.  R * C is a time constant to which a voltage drops
;       to 30% (or rises to 70%) of its final value.  Delay times
;       are measured at the 50% point.  The delay to the 50%
;       point is .7RC.
capma .0000189  ; metal area (0.000027)
capmp .000028   ; metal perimeter (0.00004)
cappa .000042   ; poly area (0.00006)
cappp .000014   ; poly perimeter (0.00002)
capda .00025    ; n+ diff area (0.00044) note: less than .7 of original
;                 because mextra treats all diffusion as n+ so use ave.
capdp .00028    ; n+ diffusion perimeter note above:
cappda .000105  ; p+ diff area (0.00015)
cappdp .00028   ; p+ diff perimeter
capga  .000483   ; gate area (0.00069)!!!besides the .7 reduced by factor
; set some flags for more accurate circuit extraction
diffperim  0    ;do not include diff perimeter along gate
diffext 8       ;no default
lambda 1
;
; set threshold voltages (voltages are normalized to the range 0 -> 1)
lowthresh 0.3
highthresh 0.8
;
; resistance tables (from SPICE simulations using JAN/86 CMC manual data)
; format: RESISTANCE (ENH or P-CHAN) (STATIC or DYNAMIC-HIGH or DYNAMIC-LOW)
;                 WIDTH LENGTH RESISTANCE(in ohms)
; supposedly, linear interpolation is done by presim (how good?)
; note: table good for range width: 3 -> 100  length 3
resistance enh static 3 3 7500
resistance enh static 5 3 3900
resistance enh static 7 3 2600
resistance enh static 9 3 2000
resistance enh static 11 3 1600
resistance enh static 20 3 890
resistance enh static 100 3 160 ;extrapolated
resistance p-chan static 3 3 32700
resistance p-chan static 5 3 16000
resistance p-chan static 7 3 10500
resistance p-chan static 9 3 8300
resistance p-chan static 11 3 6200
resistance p-chan static 20 3 3300
resistance p-chan static 100 3 530 ;extrapolated
resistance enh dynamic-high 3 3 29000
resistance enh dynamic-high 5 3 18000
resistance enh dynamic-high 7 3 13000
resistance enh dynamic-high 9 3 10000
resistance enh dynamic-high 11 3 8600
resistance enh dynamic-high 20 3 4600
resistance enh dynamic-high 100 3 940 ;extrapolated
resistance p-chan dynamic-high 3 3 47000
resistance p-chan dynamic-high 5 3 29000
resistance p-chan dynamic-high 7 3 21000
resistance p-chan dynamic-high 9 3 16000
resistance p-chan dynamic-high 11 3 13000
resistance p-chan dynamic-high 20 3 7300
resistance p-chan dynamic-high 100 3 1470;extrapolated
resistance enh dynamic-low 3 3 16000
resistance enh dynamic-low 5 3 9500
resistance enh dynamic-low 7 3 6800
resistance enh dynamic-low 9 3 5300
resistance enh dynamic-low 11 3 4400
resistance enh dynamic-low 20 3 2400
resistance enh dynamic-low 100 3 490 ;extrapolated
resistance p-chan dynamic-low 3 3 79000
resistance p-chan dynamic-low 5 3 47000
resistance p-chan dynamic-low 7 3 34000
resistance p-chan dynamic-low 9 3 26000
resistance p-chan dynamic-low 11 3 21000
resistance p-chan dynamic-low 20 3 12000
resistance p-chan dynamic-low 100 3 2450 ;extrapolated
```

Figure 5-4: The Presim configuration file.

It includes the constants for the layer area-to-node capacitance conversion constants as well as the definition of the threshold voltages and some simulation flags. As can be seen, the resistances are specified for several sizes of transistors although it was not necessary because PRESIM will interpolate or extrapolate the resistances according to each transistor's gate width and length.

The capacitances in this figure are based upon the same values used by the SIM2SPICE program described in the previous section, but they are multiplied by 0.7. These capacitances are scaled this way because RNL calculates transition times by computing $R_{effective} x C_{effective}$. This equals the time to reach 70% of the node's final voltage, but time delays are usually specified by signal crossings at the 50% voltage level. To make the adjustment for the time to reach the 50% voltage level, 0.7RC is used for the time delay computations.

One comment should be made about the 'diffext' flag in the configuration file. This flag is set to some non-zero value for NETLIST based simulations because the capacitance due to interconnecting wiring is usually not part of a NETLIST description, and to help offset this missing capacitance, a diffusion layer is assumed to extend 'diffext' microns past the transistor gate. A diffusion area equal to the gate width times 'diffext' is then assumed to exist for each transistor source and drain and provides a simple estimation of node capacitance.

5.1.3. Simulation Results Versus Measured Values

In a simple test of the IC simulator's accuracy in predicting transistor switching times, a layout for a ring oscillator circuit was created and the oscillation frequency predicted by the RNL and SPICE simulations of this circuit was compared to the measured frequency of the fabricated circuit. To cover the various levels of simulation, three types of simulation were performed: an RNL simulation based upon a NETLIST circuit description; an

RNL simulation based upon a circuit description extracted from the layout; and a SPICE simulation also based upon an extracted circuit description. This test provides an easy-to-measure value for an inverter's switching time, but the test was limited as a test of a simulator's overall ability because only one type of circuit, the inverter, was tested.

The ring oscillator logic schematic is shown in Figure 5.5.

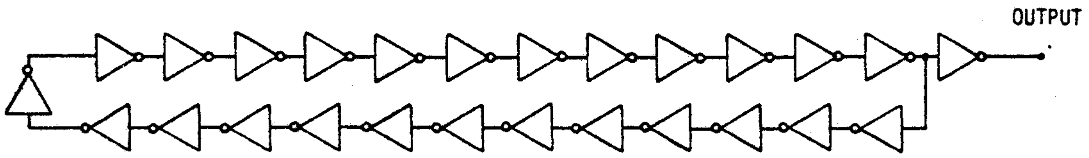


Figure 5-5: Ring oscillator schematic.

For the inverters in the loop, all pMOS transistors have a gate length of 3μ and a gate width of 5.4μ while the nMOS circuits have the same gate length but a gate width of 3μ. The inverter driving the oscilloscope has a pMOS transistor gate length of 3μ and a gate width of 35μ and an nMOS transistor gate width of 20μ. The circuit was analyzed on a Wentworth microanalytical probing system, and the power was supplied by a Hewlett-Packard 4145A semiconductor parameter analyzer. The resulting frequency measurements are shown in Figure 5.6.

The power supply current was measured with the oscilloscope disconnected. The power supply current was not constant; initially, it was measured at 1.061mA, and then after a test where the power supply voltage was raised to 10 volts, the 5 volt power supply current increased to 1.24mA. The cause of this increase was not determined, but it may have been caused by heating effects at the higher power supply voltage or the circuit characteristics may have been altered at the higher voltage.

Table 5.3 shows the predicted and the measured ring oscillator

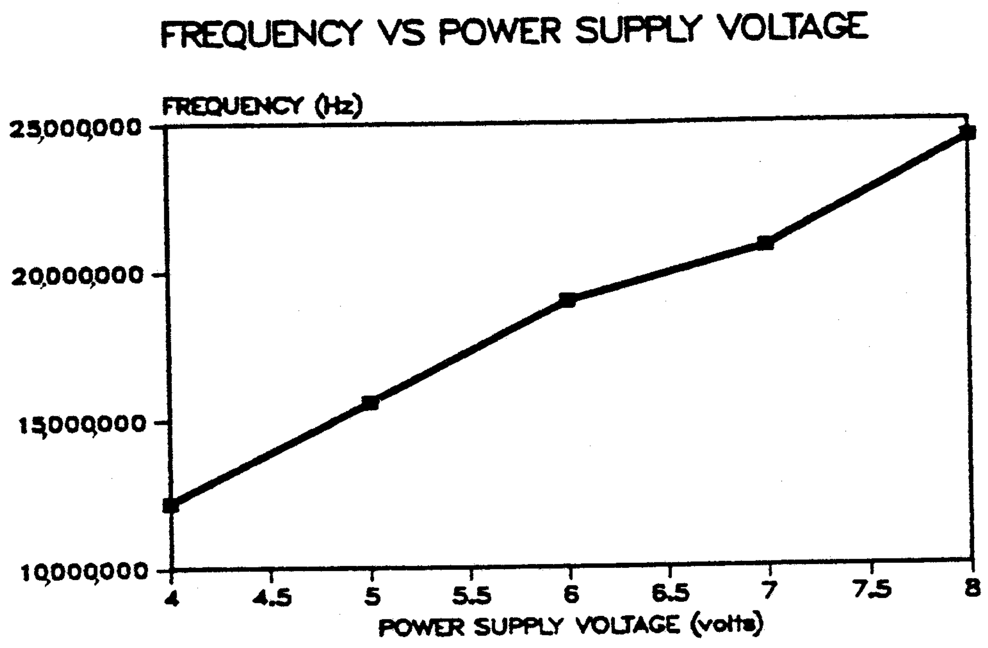


Figure 5-6: Ring oscillator frequency versus power supply voltage.

performance and the error of the simulations with respect to the measurement. As expected, the SPICE simulation predicted a lower than actual circuit speed performance while the NETLIST circuit description gave a too high circuit speed prediction.

Ring oscillator performance

	SPICE	RNL	RNL/NETLIST	Measured
Frequency	13MHz	16MHz	22MHz	15.6MHz
% error	-17%	2.6%	41%	-

Table 5-3: Predicted and measured oscillator frequencies.

One key component in the simulation accuracy is the node capacitance and it is useful to compare node capacitances for the various simulators. This comparison helps reveal whether the simulation inaccuracies are due to poor transistor parameters or poor inter-connect wiring capacitance parameters. The node capacitances are listed in Table 5.4.

Node capacitances

	SPICE	RNL	RNL/NETLIST
Output node	0.257pF	0.290pF	0.214pF
Internal node	0.0312pF*	0.0518pF	0.0363pF

*does not include gate capacitance which SPICE itself calculates.

Table 5-4: Node capacitances for various simulators.

From this it can be seen that the NETLIST based simulation predicts a circuit speed simulation that is too high because the node capacitances are too low. The cause of the discrepancy between the SPICE and RNL circuit descriptions, though, is not clear because the node capacitances for SPICE appear to be lower or equal to the RNL node capacitances.

For more accurate simulations, more attention should be given to developing good model parameters in future projects. What is required is a test structure and a test method to determine the SPICE model parameters and the inter-connect wiring capacitance parameters since both play an important role in simulation accuracy.

5.2. Integrated Circuit Development

In this section, the circuits for the basic building blocks of the ECG processor are discussed. The first circuits to be presented are the most common logic functions like inverters, tri-state buffers, latches, shift-register cells, flip-flops, and add/subtract cells which make up a basic cell library. Many of the cells in this library have their layout arranged so that they have a common cell height and may be placed next to each other to share power supply lines; this approach is much like the standard-cell IC design method. As well, the layout of add/subtract cells, shift-register cells, and latches are also arranged to allow any number of them to be placed together to create functions for any desired operand length. After the library cells are presented, the special purpose circuits like the static memory, the PLA, and the input and output pad circuitry are discussed.

5.2.1. The Library Cells

A schematic diagram for each of the cells in the library is presented along with a description of the circuit operation whenever necessary. A NETLIST description of these cells is given in Appendix B.

1. Inverters and tri-state inverter

The schematics for these circuits are given in Figure 5.7. The inverters are available with varying load driving capability with pull-up/pull-down gate widths of $5.4\mu/3\mu$, $12\mu/5.4\mu$, $25\mu/15\mu$, $35\mu/20\mu$, $50\mu/30\mu$, and $70\mu/40\mu$.

2. NAND, NOR, XOR, and XNOR logic functions

The schematics for these logic functions are given in Figure 5.8. The NAND gate is available with two load driving capabilities: the NAND6 transistor gates have twice the gate width/length ratioes of the NAND gate transistors. The exclusive-OR and exclusive-NOR gates use ten transistors rather than the six-transistor version presented in [38] because SPICE

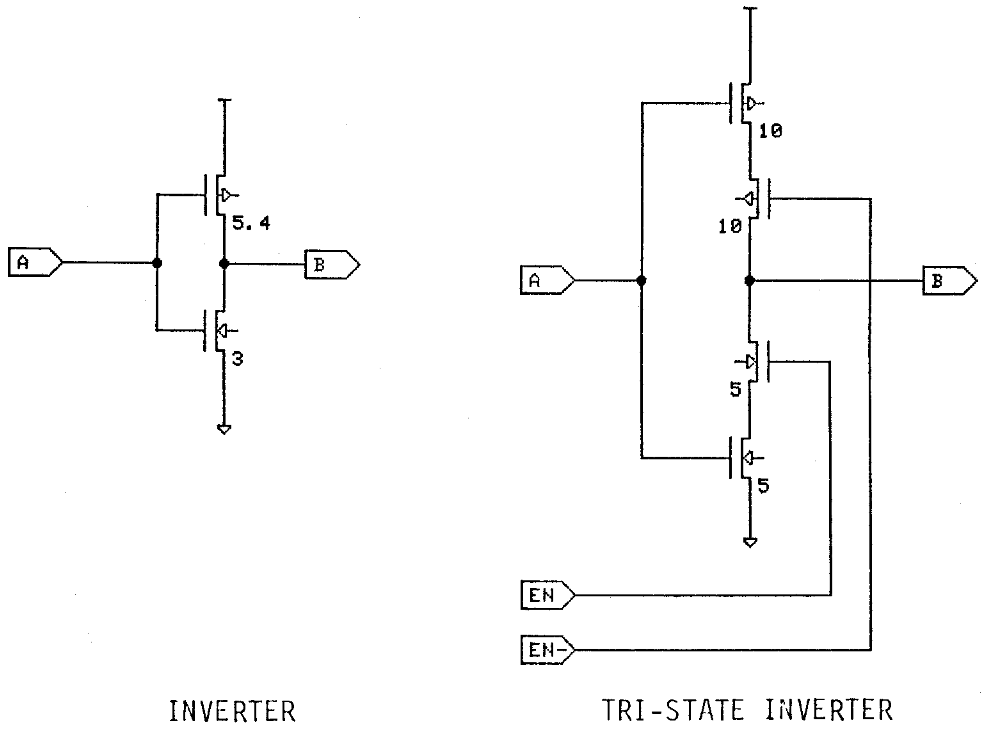


Figure 5-7: Schematic for inverter and tri-state inverter.

simulations showed that the six-transistor design was not fully-restoring and did not pull the output below $.3V_{DD}$ for certain loads.

RNL simulations were performed on circuits extracted from the layout. With a one inverter load (0.022pF), the maximum delay times predicted by RNL for the NAND, NOR, and XOR gates were 1.5ns, 3.2ns, and 4.0ns respectively.

3. D-latch

The D-latch schematic is given in Figure 5.9. When 'ck' is logic 1 and 'ck-' is logic 0, the output logic state equals the input logic state. When 'ck' is logic 0 and 'ck-' is logic 1, the output equals the value of the logic state when 'ck' and 'ck-' make the transition to these logic states. The latch is completely static.

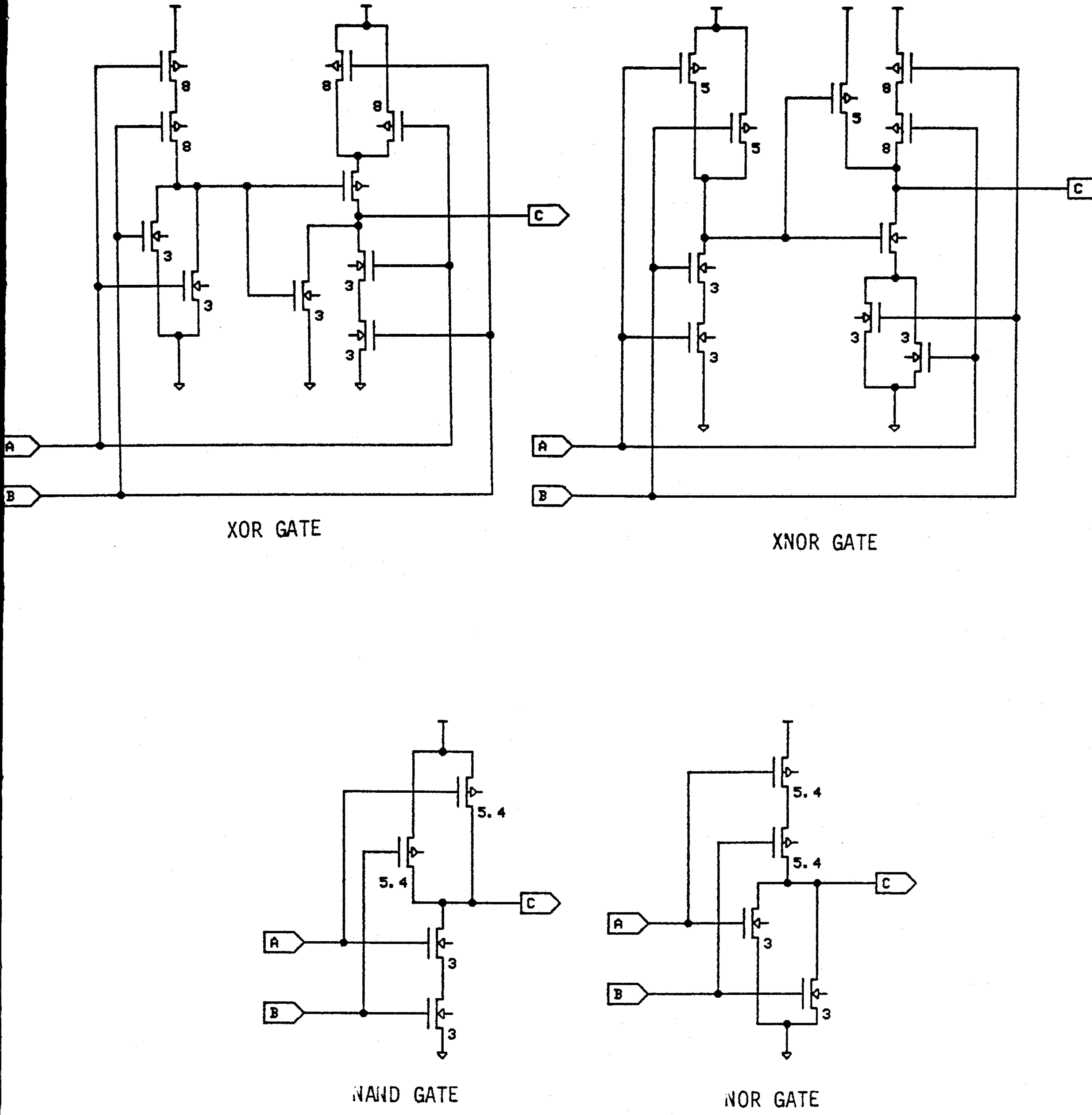
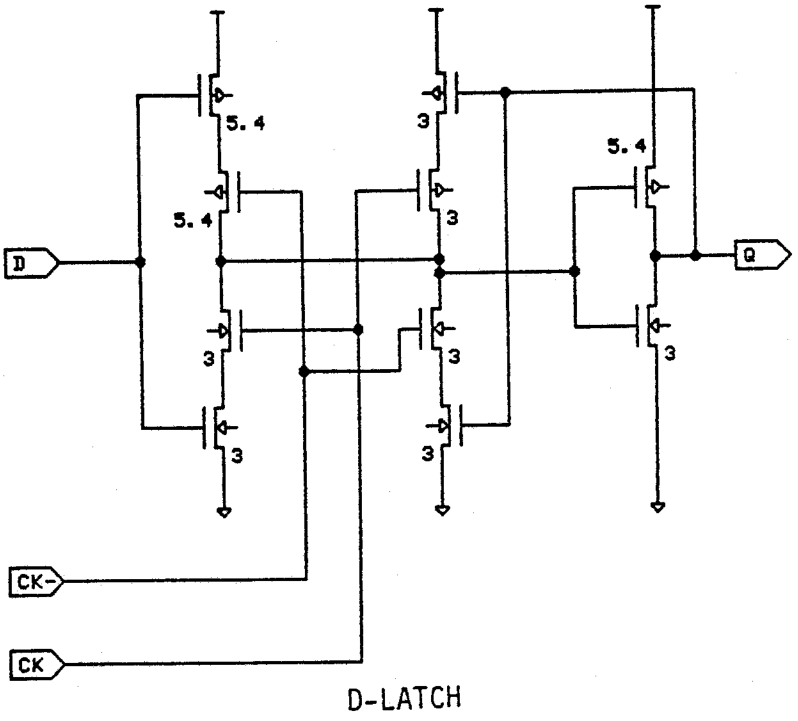


Figure 5-8: Schematic for NAND, NOR, XOR, and XNOR gates.



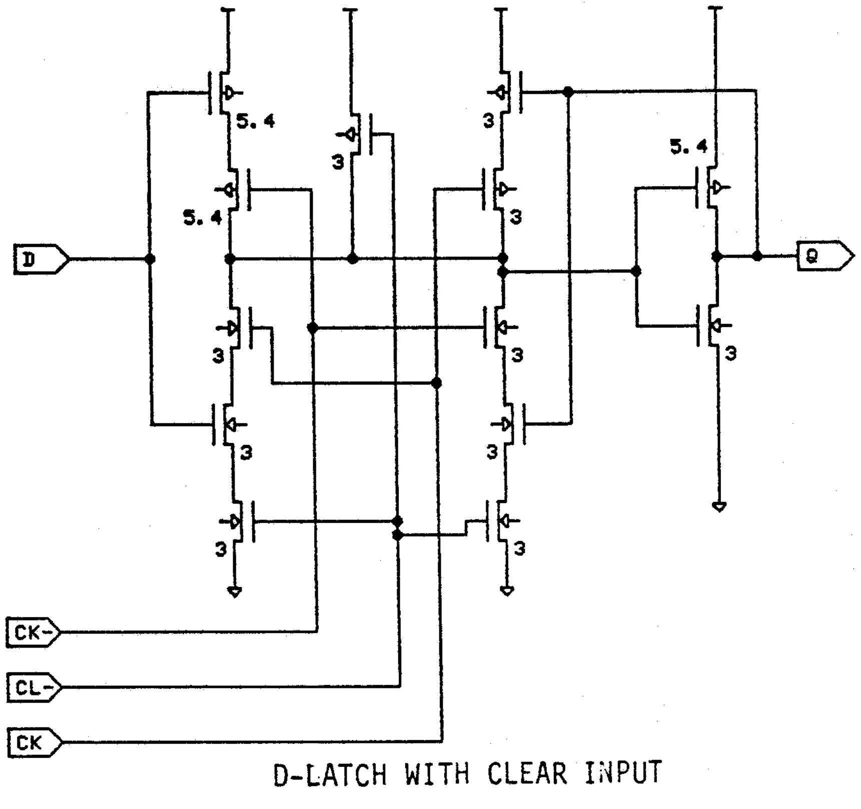


Figure 5-9: Schematic for D-latch and D-latch with Reset.

4. SR latch

The schematic for the SR latch is given in Figure 4.10. For the illegal state where R=1 and S=1, both Q and Q- are logic 0.

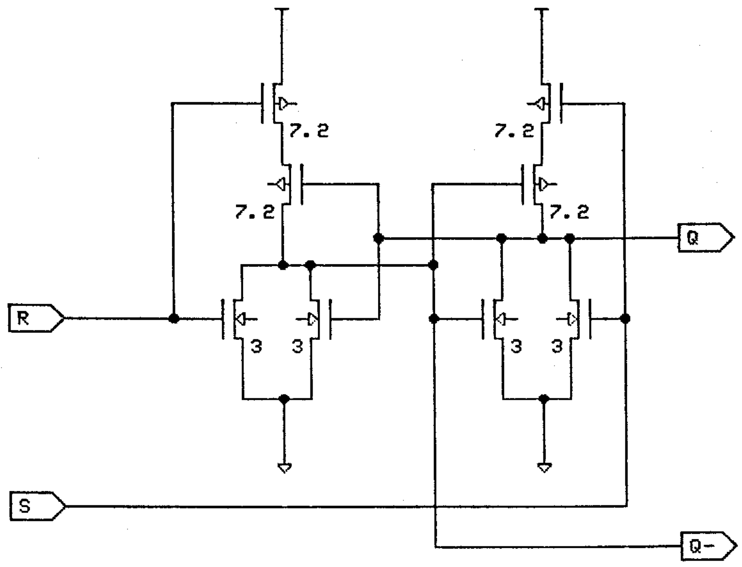


Figure 5-10: Schematic for SR latch.

5. Shift-register cell

The schematic for the shift-register cell which is used to make a parallel loading serial-out shift-register is shown in Figure 5.11. It shifts data according to a two-phase non-overlapping clock, and it is completely static as long as either phase of the clock is in the logic 1 state. Because of the static design, the layout area is much larger than for a dynamic design; a dynamic design would use five transistors as opposed to this twelve transistor circuit. The maximum clock frequency based upon a RNL simulation of the extracted circuit is 156 MHz, but this is with perfect clock signals having no time delays between phases.

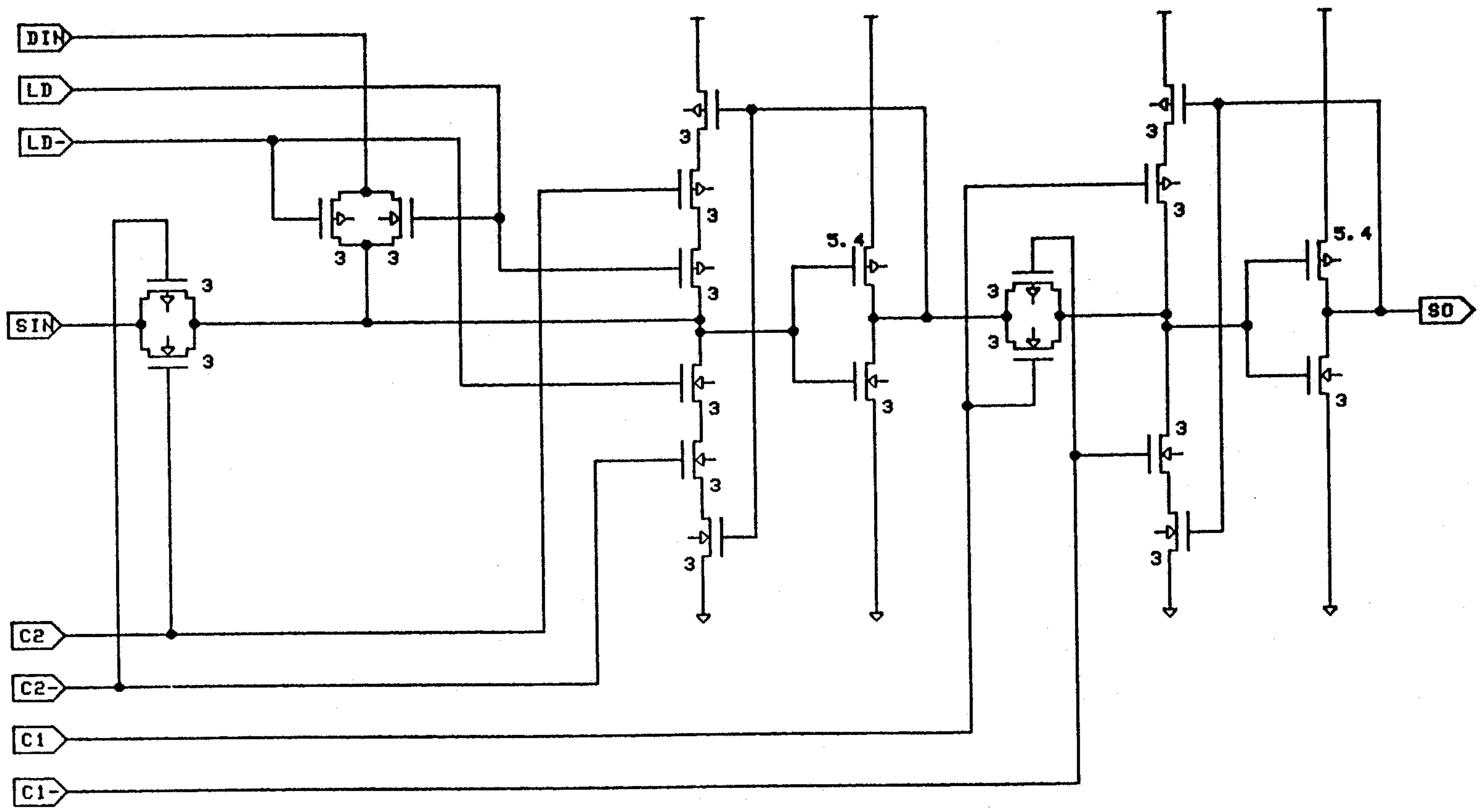


Figure 5-11: The shift register schematic.

6. Add/subtract cell

Also referred to as the controlled add/subtract cell, this cell is based upon the schematic diagram shown in Figure 4.8. The logic is completely static. The layout for this cell is shown in Figure 5.12.

The carry delay time is the limiting factor in the adder's computation time, and the overall adder delay is

$$\text{Addtime} = 3\tau_{\text{XOR}} + (N-1)\tau_{\text{CARRY}}$$

where N is the number of add cells, τ_{CARRY} is the worst case delay from a carry-in transition to a carry-out transition, and τ_{XOR} is the XOR gate delay. τ_{XOR} is assumed to be a small fraction of the total delay time, therefore, determination of the add time depends upon τ_{CARRY}. τ_{CARRY} from an RNL simulation of an extracted circuit was found to be 3.7ns, and so for a 16-bit adder, gives the expected adder computation time to be about 60ns.

The add/subtract cell can be modified to an add-only cell by eliminating the top-most XOR gate.

7. The counter cell

This cell is intended as a building block for the 10-bit and and 3-bit address counters. Two features of this counter cell are:

- it may be pre-loaded with any value,
- the outputs change simultaneously when the clock signal ϕ_2 makes a low-to-high transition to provide synchronous operation.

The schematic for the counter cell is shown in Figure 5.13.

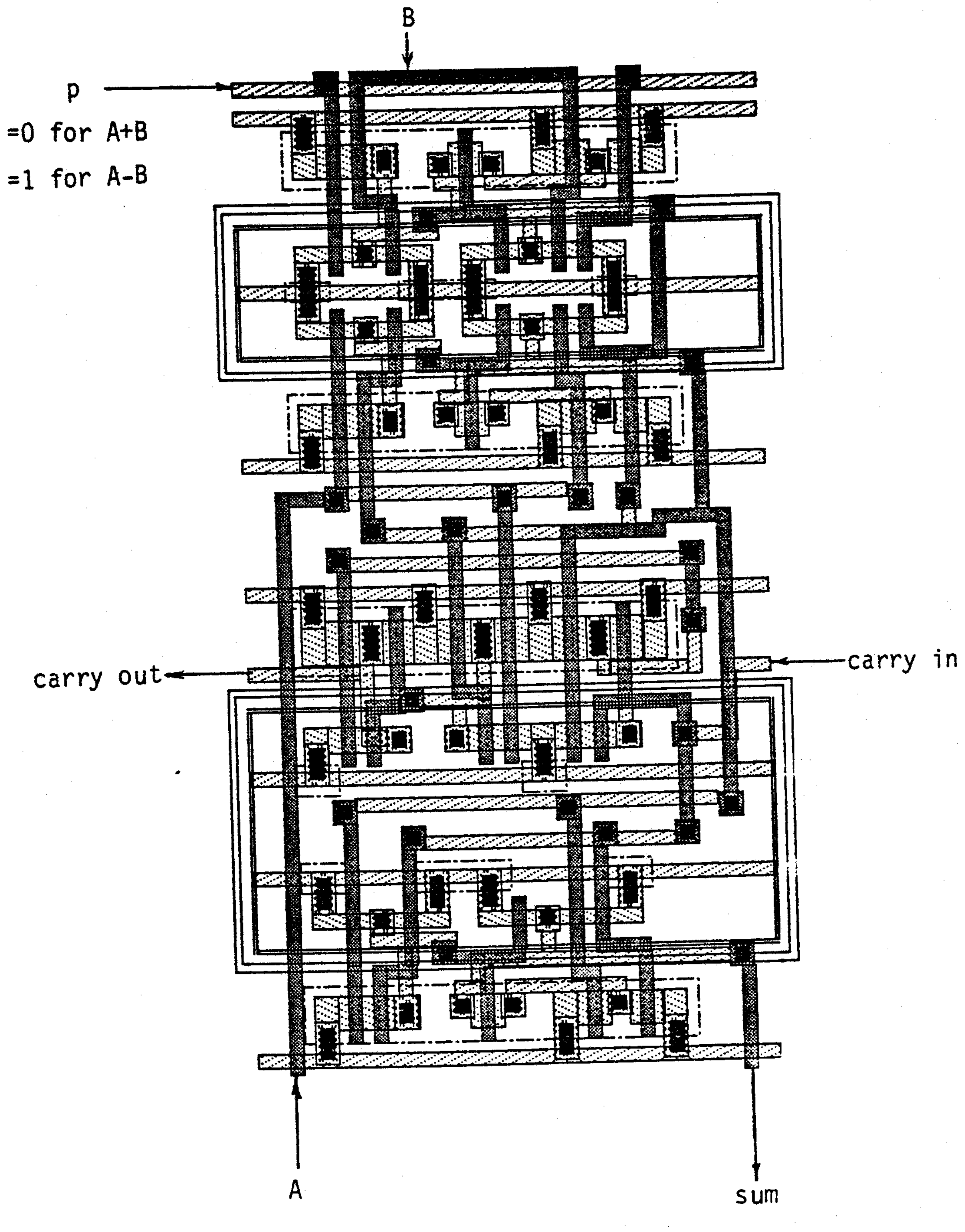


Figure 5-12: Layout for the add/subtract cell.

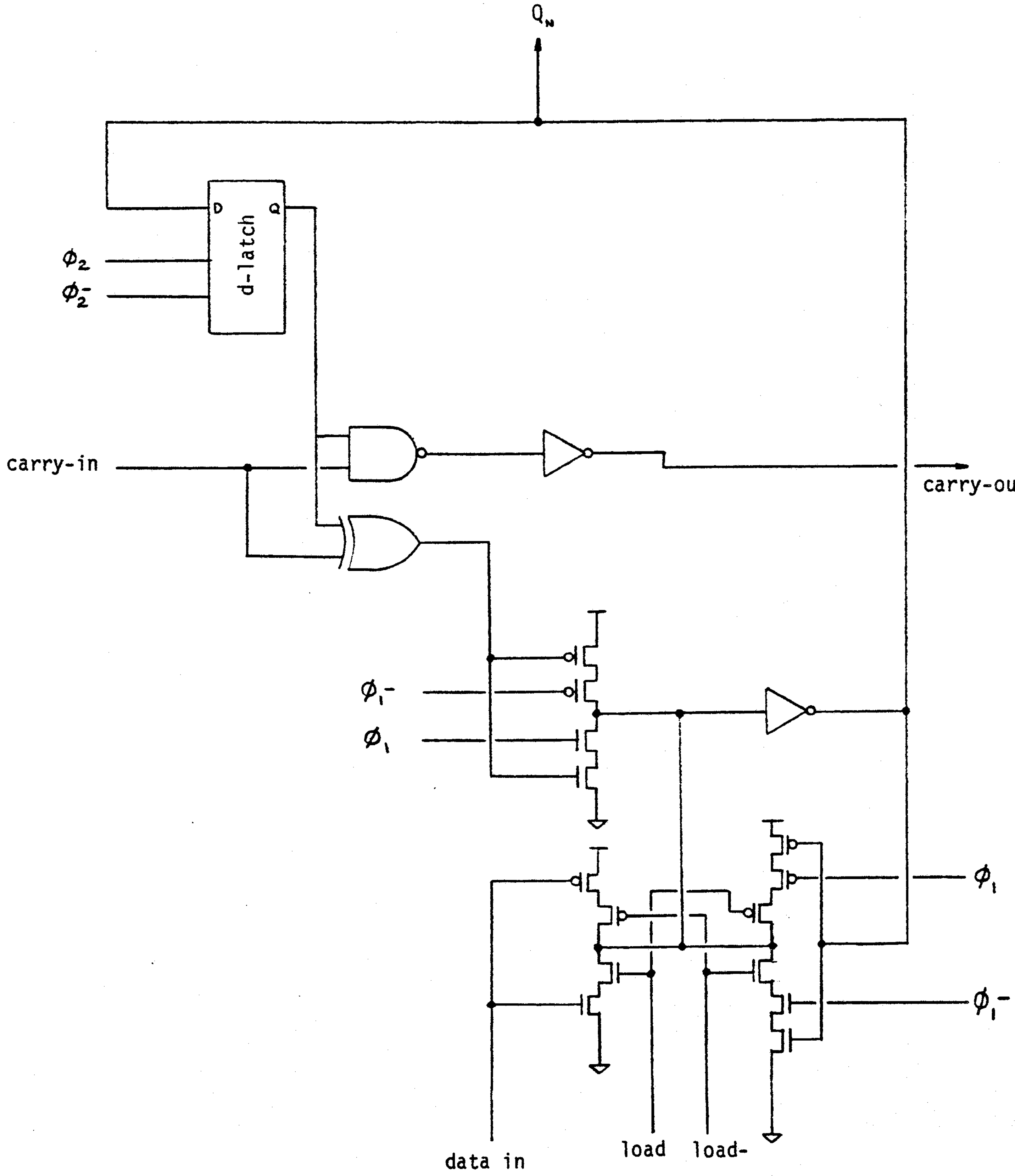


Figure 5-13: Schematic for the counter cell.

5.2.2. Special Purpose Cells

The IC cells studied in this section have been separated from the previous section because more care must be taken in their design. The previous section dealt with logic cells which perform their intended functions under a wide range of input rise and fall times and output loads; their layouts were not critical for correct operation. However, in the cells in this section, transistor sizes do affect the correct operation of the circuits and more attention must be given to the analog characteristics of the circuit design.

5.2.2.1. Static Memory Development

Static memory is based upon a cell which uses positive feedback to retain information for as long as power is applied. A common type of 6-transistor CMOS RAM cell is shown in Figure 5.14. This circuit consists of two cross-coupled inverters with each inverter output being connected to a level sensing line through an nMOS pass transistor.

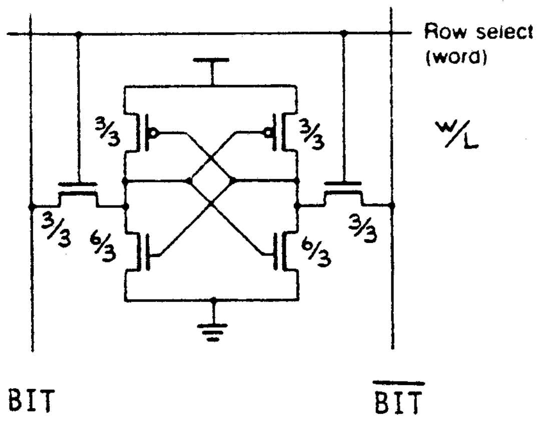


Figure 5-14: Schematic for the 6-transistor static RAM cell.

The ratio of the size of the pass transistors to the nMOS pull-down transistors is critical to the correct operation of the memory cell. A general rule for the sizing of these transistors is that the conductance of the pull-down transistors should be three times the conductance of the pass transistors, and the conductance of the pass-transistor should be several times that of the pMOS pull-up transistor [44]. The actual transistor sizes depend on the ratio of the sense line capacitance to memory cell capacitance.

Reading and writing to the memory cell depends upon the voltages of both the BIT and $\overline{\text{BIT}}$ sense lines. When writing to the cell, the BIT line is set to the logic level to be stored and the $\overline{\text{BIT}}$ is set to the complement logic level. Then, when the WORD line is asserted, the voltages applied to the cell unbalance the cross-coupled inverters enough so that positive feedback causes the cell to assume the desired state.

When reading the contents of the cell, both the BIT and $\overline{\text{BIT}}$ sense lines are pre-charged to equal voltages in the range $0.6V_{DD}$ to V_{DD}. The sense lines should have equal voltages which are greater than the V_{INV} of the memory cell inverters in order not to unbalance the cell when the WORD line is selected. Once the memory cell is selected, the memory cell should cause a detectable voltage difference between the BIT and $\overline{\text{BIT}}$ lines. If the conductance of the pass transistors is too high, the contents of the cell will be erased, and if the pass transistor conductance is too low, the time taken for a read operation will increase and the write operation may not work correctly. The transistor width-to-length ratioes used in this design are shown in Figure 5.14 and were verified using SPICE simulation.

Figure 5.15 shows the memory column driver which precharges the BIT and $\overline{\text{BIT}}$ lines. The PRECHARGE line is used to raise the voltages on the sense lines to the high voltage level for the read operation. For the write operation, the sense lines are driven through a tri-state inverter.

Usually, for the fastest operation in the read cycle, a differential amplifier is used to detect the memory cell state from the voltages on the BIT and $\overline{\text{BIT}}$ sense lines, but to keep the memory compact, this design uses

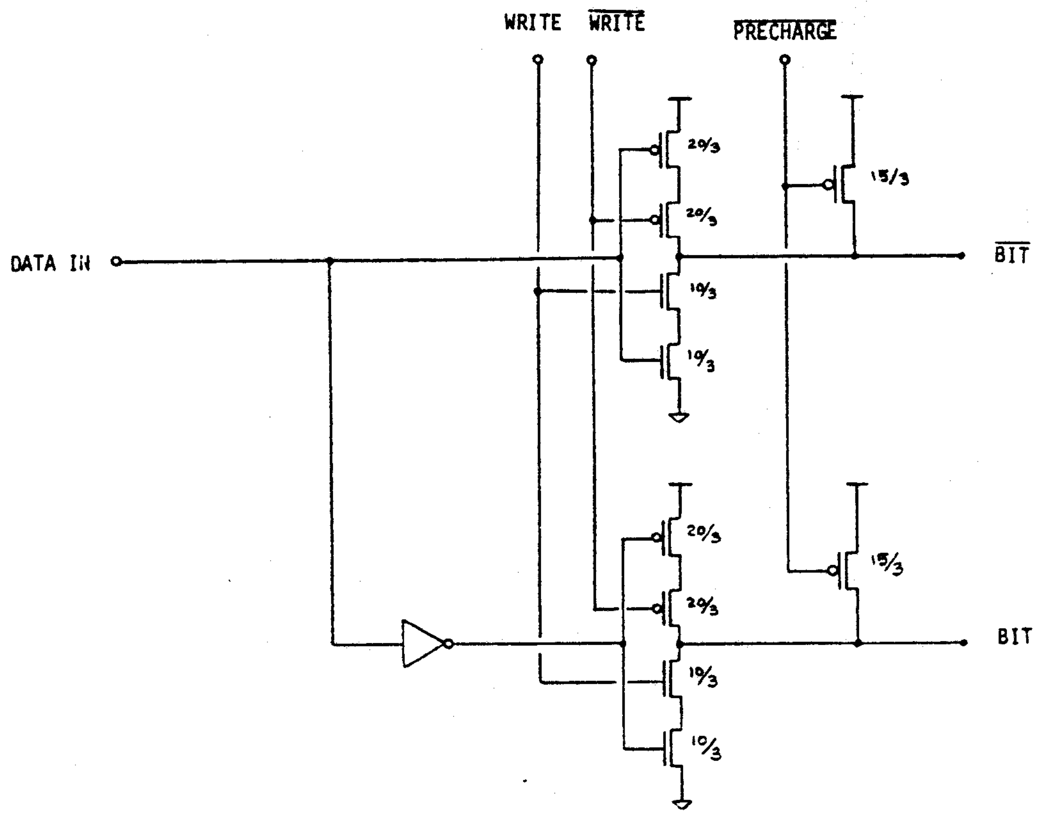


Figure 5-15: Memory column driver circuit.

only an inverter to restore the sense line voltages to normal logic levels. The loss in speed is not critical here because the RAM is already fast enough for this application. Memory speed depends upon the capacitance of the sense lines, and since a relatively small number of memory cells are connected to the sense lines, the sense lines are lightly loaded making the memory relatively fast. When there are many memory cells per sense line, the line voltages change slowly due to the increased capacitance, and the need for a differential amplifier to detect small voltage differences becomes stronger.

The row drivers assert the WORD select lines. The logic for the function, shown in Figure 5.16, is not complex. The enable line is included to provide the ability to de-select all WORD lines. This is important for the sense line precharge operations and for changing the memory address.

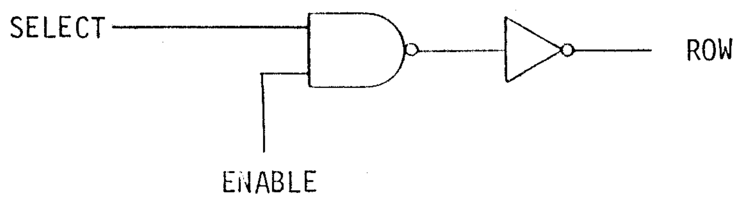


Figure 5-16: Memory row select circuit.

The static memory has a control unit which provides the signals necessary for the row driver enable function, the sense line precharge function, and the write enable lines. The schematic for this circuit is shown in Figure 5.17. The monostable multivibrators are used to enable the memory on the rising transitions of both clock signals ϕ_1 and ϕ_2. This means that this memory is designed for synchronous operation. Because transitions of both clock cycles are detected, two memory accesses may be made per clock cycle, and for this application, a memory read is performed during the ϕ_1 clock phase, and a memory write operation is made during a ϕ_2 clock phase.

The operation of the control circuit depicted in Figure 5.17 is as follows. When ϕ_1 or ϕ_2 makes a transition to a logic 1 level, the monostable multivibrator sets the SR latch to the logic 1 state. This, in turn, disables all row select lines, and in the case of a read operation, enables the precharging of the sense lines. When both sense lines, BIT and $\overline{\text{BIT}}$, are sufficiently precharged, the SR latch is reset, the precharge transistors are turned off, and the row select lines enabled.

5.2.2.2. PLA Circuit Development

As discussed earlier, the layout of the PLA is fixed by the PLA generator TPLA. However, a check of the PLA circuitry is done to verify that the pull-up and pull-down transistors are ratioed correctly and to determine the worst case delay time. The check of pull-up and pull-down

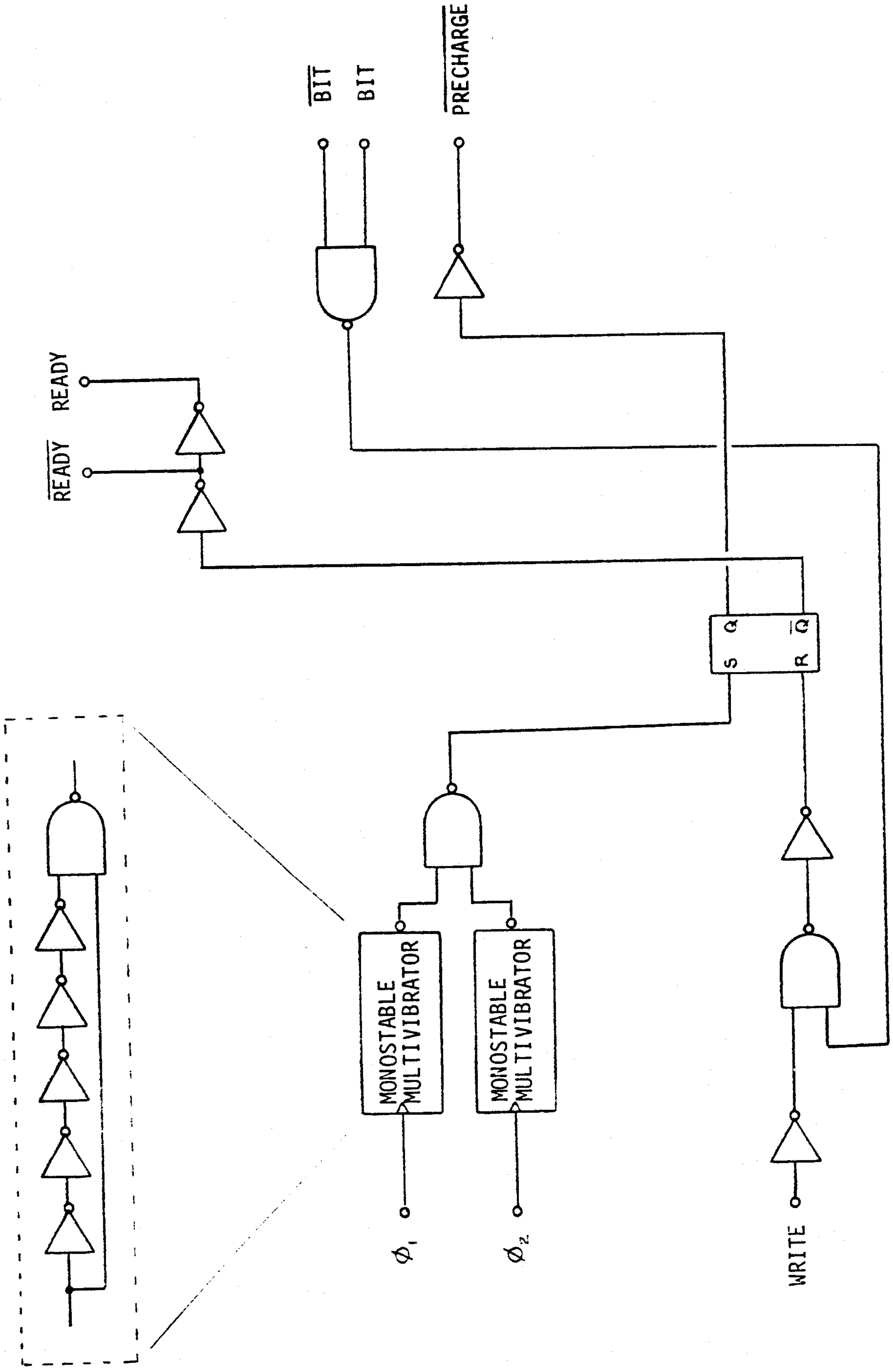


Figure 5-17: Memory control circuit.

transistor sizes is especially important because the layout pattern was developed for a different IC fabrication process.

SPICE modelling of a PLA circuit is straight forward and only one input and output of the PLA needs to be considered. The PLA circuit used for this simulation is shown in Figure 5.18. The resistances and capacitances have been added to account for the worst case values, which in this case, includes a 1000μ polysilicon wire connected to 10 transistors in the AND plane of the PLA, and 150μ diffusion wires to ground for both the AND and OR planes. From the SPICE simulation, the delay time for a low-to-high transition is 55ns and the high-to-low transition time is 60ns.

5.2.2.3. Input Protection

A test of the dielectric breakdown voltage of the gate oxide showed that a MOS transistor fabricated in CMOS3 technology can be permanently damaged when a voltage greater than 50 volts is applied between the gate and the substrate. Since it is possible that during handling, an IC can be exposed to static charges greater than 2000 volts [45], a means of limiting the on-chip voltages to less than 50 volts is necessary. One type of input protection which is used for this project is shown in Figure 5.19. The diodes clamp the input voltages relative to the substrate to the range $-V_{diode}$ to $V_{DD}+V_{diode}$. The resistance is used to help dissipate energy when current surges do occur.

The layout for this input protection circuit has been fabricated and has been found to limit the range of input voltages. However, the heating effects of dissipating static discharges has not been accounted for and the present design will not withstand large current surges. More work must be done to develop input protection which will survive the electrostatic discharges normally encountered in IC handling.

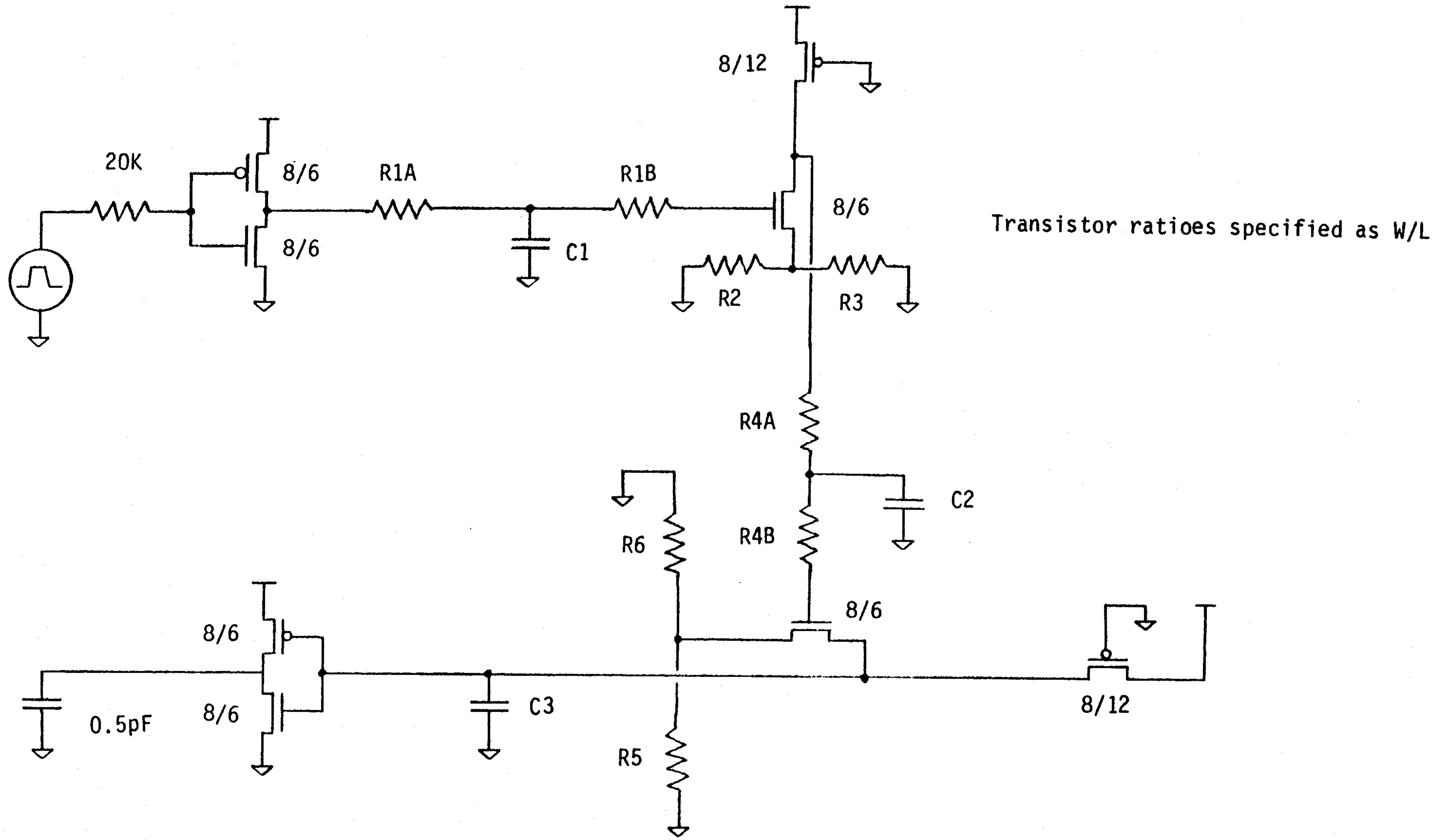


Figure 5-18: PLA circuit model for simulation.

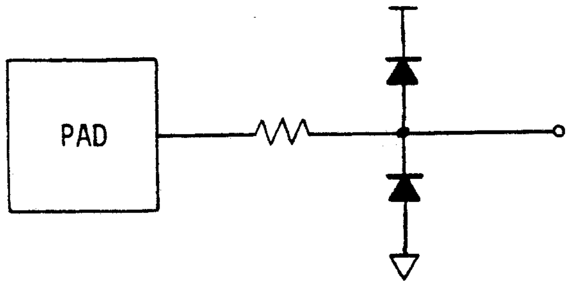


Figure 5-19: Input protection circuit.

5.2.2.4. Output Pad Drivers

To drive off-chip loads which typically have capacitances three orders of magnitude greater than the on-chip node capacitances, a buffer with high gain transistors is necessary. For this purpose, the circuit shown in Figure 5.20 is used.

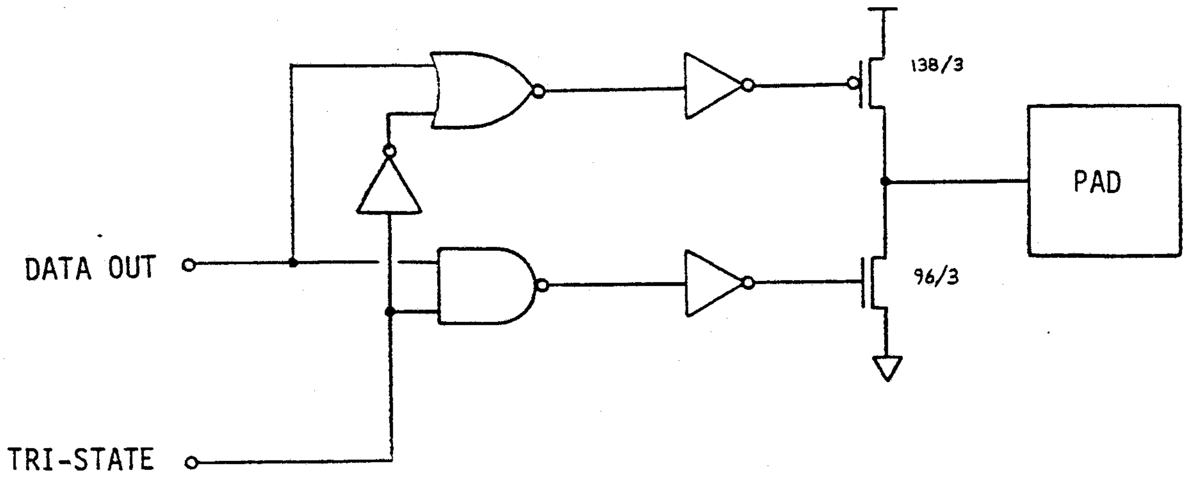


Figure 5-20: Output pad driver.

The output transistor gate widths for the pMOS and nMOS transistors are 138μ and 96μ respectively. These values were selected through SPICE simulation to drive a 10pF load (equivalent to one oscilloscope probe) with a

10ns delay time. To drive these transistors, a large inverter with pull-up and pull-down transistor gate widths of 30μ and 16μ is used. These gate widths are the geometrical means of the transistor widths of a standard transistor and the output transistor widths and provide the minimum delay time for driving a large capacitive load from a minimum size inverter output with a single intermediate buffer.

Another feature of this pad driver is that it has a tri-state capability; by asserting the TRI-STATE signal, both output transistors are in the off-state and the pad may be used as an input pad.

A demonstration of the pad driver's performance is shown in Figure 5.21. The delay time for a high-to-low transition is 10ns while the low-to-high transition time is 15ns for a 10pF load.

5.3. Test Results of Two Sub-circuits

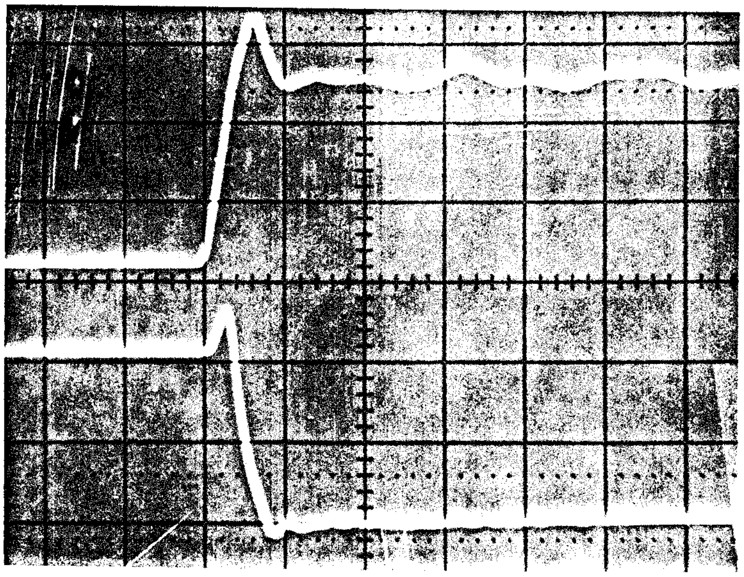
A total of five integrated circuit designs were submitted for fabrication. They included an 8-bit accumulating adder, a PLA - based finite state machine, an 8-bit by 8-bit canonical signed digit multiplier, and an 8-bit square-root extractor. The multiplier and square-root extractor did not work because of layout errors. The finite-state machine worked, but not as intended. The 8-bit accumulating add/subtract unit, which was based on the same layout for the multiplier's add/subtract unit, did work as did the static RAM.

5.3.1. The 8-bit Accumulating Add/Subtract Unit

This add/subtract unit is part of the add/subtract function of the shift-and-add multiplier. It was one of the first designs and was fabricated with the CMOS1B technology.

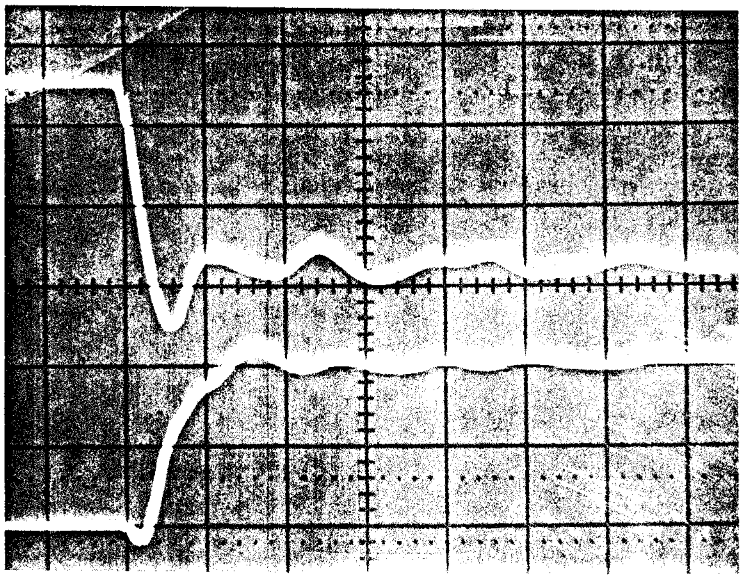
The circuit for testing this IC is shown in Figure 5.22.

Pad driver output fall time



Load = 10pF // 10MΩ
Sweep = 50ns / division

Pad driver output rise time



Load = 10pF // 10MΩ
Sweep = 50ns / division

Figure 5-21: Pad driver output characteristics.

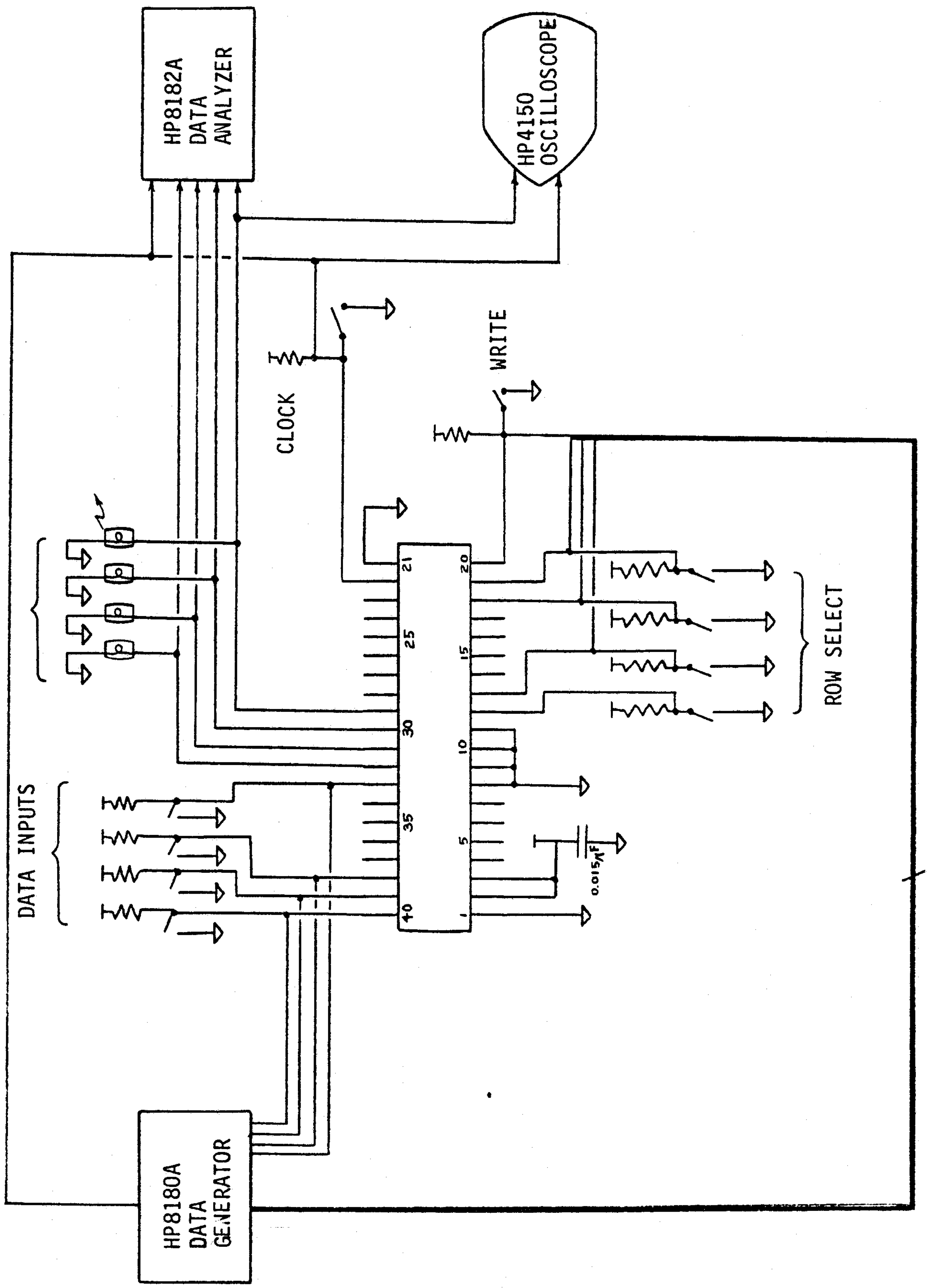


Figure 5-22: Test circuit for the add/subtract IC.

The test was conducted in two steps. In the first step, a push button was used to generate a two-phase clock signal; in this way, the addition and subtraction of numbers could be verified using a row of logic level indicators provided on the breadboard. In the second step, the data generator was used to generate the two-phase clock, and the data analyzer was connected to the outputs of the device under test. The purpose of this test was to find the maximum operating frequency of the accumulator by increasing the clock frequency until errors appeared at its output. To do this, at clock frequency of 1MHz, the data analyzer recorded the outputs of the accumulator in its memory. Then the logic analyzer was placed in a sample and compare mode where the outputs of the analyzer were sampled and compared to the contents in the logic analyzer's memory; any differences would be high-lighted in an error map. To synchronize the logic analyzer to the test clock frequency, the logic analyzer was set up to sample the accumulator's output at a fixed time interval after the clock transition. Then the data generator's clock frequency was increased until errors appeared in the data generator's memory map. This occurred at 9.98 MHz.

Two problems appeared in the test; one was an intermittent large power supply current at clock frequencies greater than 10MHz, and the other was electrostatic damage to some inputs. The power supply current surges were thought to be caused by latch-up in the pad drivers though this was difficult to determine because of the intermittent nature of the fault. In future versions of the pad driver design, the pMOS drivers were separated from the nMOS drivers by a greater distance. This problem has not since been noted in the CMOS3 designs. The second problem, electrostatic discharge faults, was due to a lack of input protection circuitry. Many of the damaged chips functioned, but the damaged inputs had an input impedance measured by a Fluke 8010A multi-meter as between 1 and 10MΩ (the input impedance of an undamaged input was greater than the 20MΩ range of the meter) and appeared to be a logic 1 in the functional tests of the circuit regardless of the applied input voltage. Rudimentary ESD protection was included in future CMOS designs and the number of cases of ESD appeared to decrease though it was not eliminated. This demonstrated that the present input protection circuitry still needed to be improved for use in commercial circuit designs.

5.3.2. The 8-word Static RAM

This RAM circuit was designed for the storage of eight 20-bit words, however, due to limitations of the number of input and output pins, bits 0 to 16 were connected together to create a circuit that functioned as though it had eight 4-bit words. As well, the address decoder was not included to because it permitted only sequential access of the memory, and random access tests were desired.

Two types of tests for the RAM were performed. One test determined the access time of a read operation, and the other test ensured that various combinations of inputs could be stored without affecting neighbouring memory cell locations. The test circuit for the RAM is shown in Figure 5.23, and a sample of the resulting read access time is shown in Figure 5.24. Allowing for the 10ns delay time of the pad driver, the measured read access time, as measured from the clock input is 60ns. To test the write cycle time, the data generator and data analyzer were used in a similar manner as described in the previous section and the clock frequency increased until the period equalled the read access time. The data generator provided data which were written, read, re-written, and re-read to and from the various memory locations. The data analyzer recorded the results of these operations and no errors were reported when the clock period equalled the memory read time. This showed that the time for a write operation was equal to or less than the read cycle time.

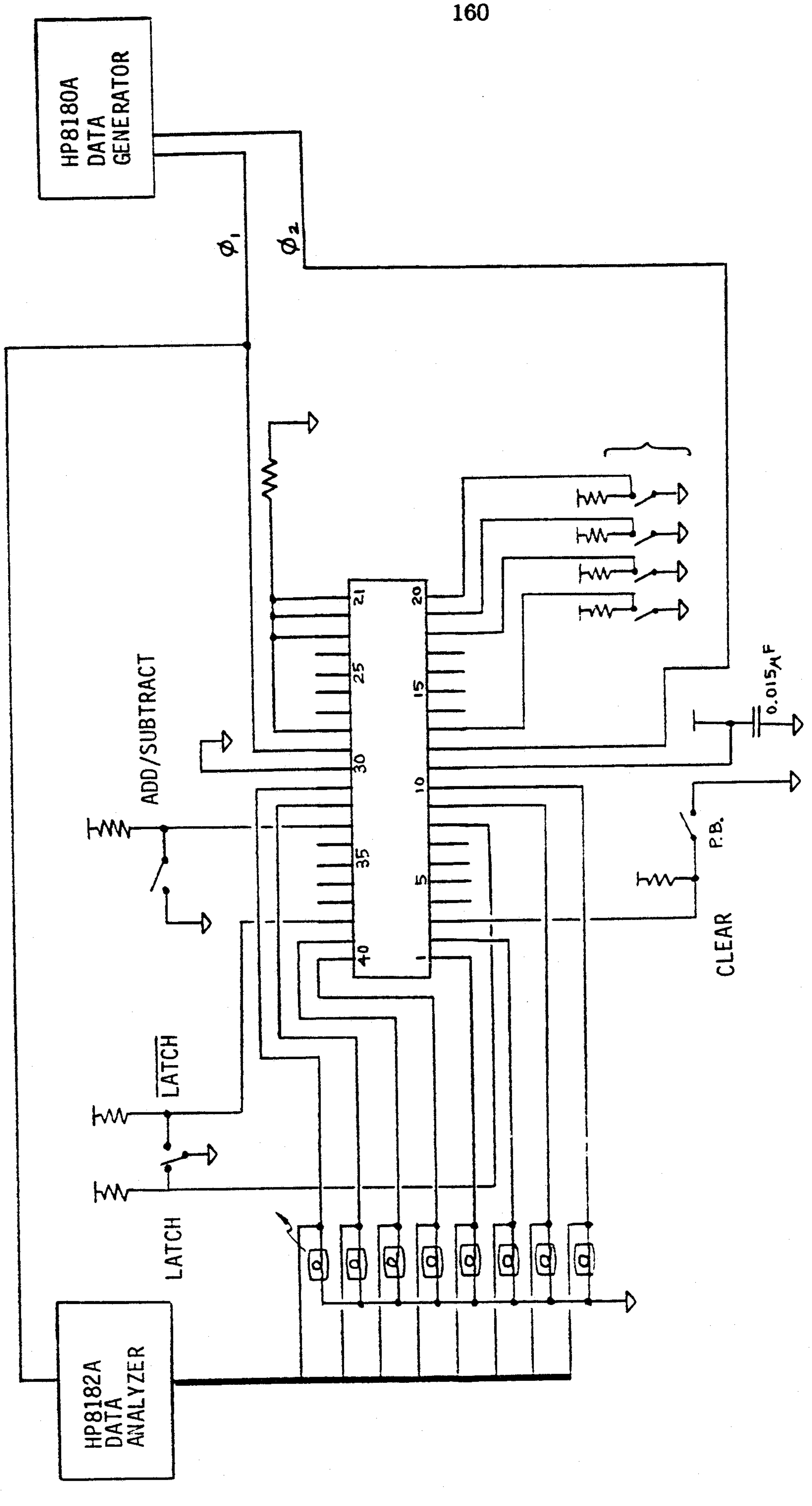


Figure 5-23: Test circuit for the static RAM.

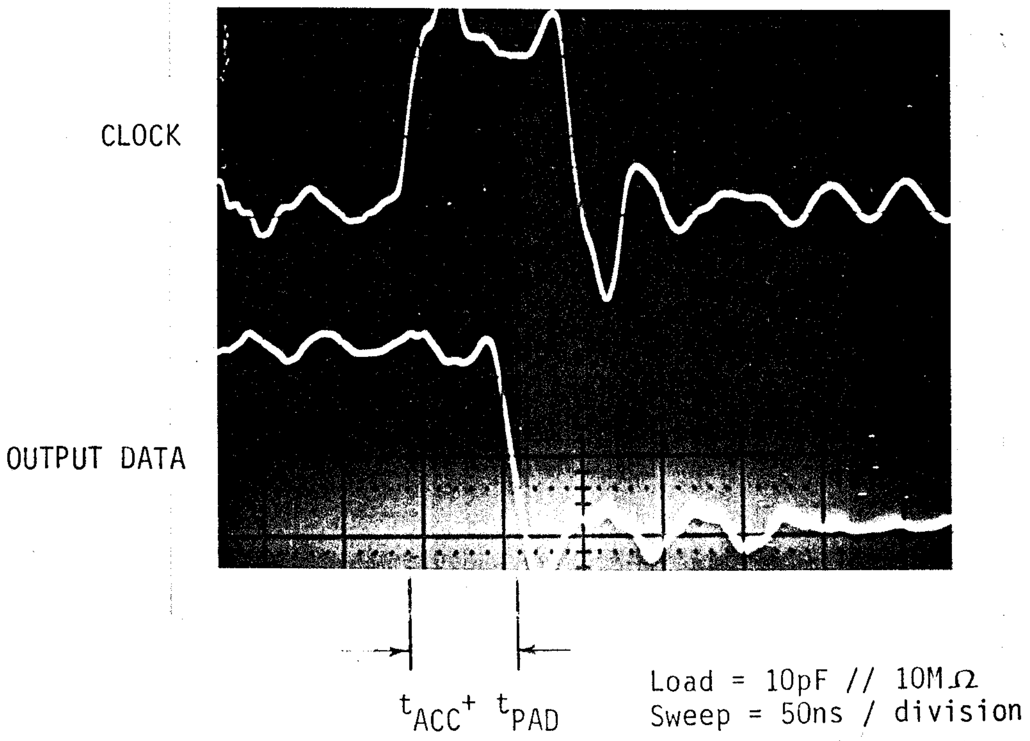


Figure 5-24: Memory read access time.

5.4. Simulation Results for the ECG Processor

The complete circuit description of the ECG processor is included in the NETLIST description in Appendix B. This NETLIST description has been translated by the NETLIST program to a transistor net list which, in turn, was prepared for RNL simulation using PRESIM and the configuration file presented in the first section of this chapter.

One change was made in the NETLIST circuit for the purposes of simulation. The change made was in the value to which the lower ten bits of the address were compared to signal the end of the matrix . In the original design, this value was 960, but for simulation purposes, this value was changed to 3 because the time to cycle through a 961 element matrix was excessive and not necessary. For example, the simulation of forty clock cycles required about 20 minutes of CPU time, and so a test using a 961 element matrix would have required about 33 hours of CPU time. This was unrealistic especially since a simulation would be re-run several times as errors in the circuit were uncovered. This change, however, did not affect the thoroughness of the check of the ECG processor's algorithm because these cycles were repetitions of the same control sequence. A matrix using only 4 elements provided as thorough a test of the ECG processor's algorithm as did a 961 element matrix and required the simulation of only 40 clock cycles per matrix classification.

The input signals for testing the ECG processor were first specified in hand drawn timing diagrams. This information was then translated into text form and entered into a computer file using the input specification code for the GEN_TIME program. The GEN_TIME program then converted this file into an RNL input stimulus file. It would have been possible to specify the RNL input directly, but the GEN_TIME specification format was more compact and less prone to errors.

The specification of the RNL inputs pointed out one area in which RNL simulation could be improved. As the programs are set up at present, the user must know the sequence of events before the simulation actually takes place. For complex circuits with long simulation times, however, this can be tedious, and what would be useful in this situation would be an input stimulus which was conditional upon the current simulation state. This would actually combine the low level simulation at the transistor level provided by RNL with high level algorithmic descriptions made possible by the conditional input stimulus statements. With this approach, the user could use the simulation to both verify the circuit and predict how the circuit would behave for a given input stimulus.

Once the RNL input signal file was created using GEN_TIME, the RNL simulation was performed and the RNL output recorded in a logic analyzer tabular format. The recorded data included values of data buses internal to the ECG processor as well as its I/O ports. Once the RNL output was recorded, its output values were compared to the expected values listed on the timing diagrams. Of particular interest were whether the correct addresses were computed by the ECG processor, and whether the final similarity values were as predicted.

The input test sequence included a reset period, a sequence where the matrix data was read, and a sequence where the results were written to external memory. During the reset period, the ECG processor polled the first 'stop-byte' location until it found a non-zero value. Then it wrote a zero value to back this location, and read the following address pointers into the appropriate address and control registers. Then the recognition matrix data was read into the ECG processor. This data included the unknown matrix matrix [3,0,55,0] and the reference matrices [0,0,0,0], [16,0,0,0], [0,0,1,0], [1,0,2,0], [92,0,255,0], [12,0,0,0], [0,0,200,0], and [6,0,1,0]. In the next phase of operation, the resulting similarity values computed by the simulation were were written to external memory. These values were 0, 4, 7, 8, 114, 3, 98, and 9, and they agreed with hand computations. The addresses, read/write signals, and shared/dedicated memory select lines also had their expected values.

The clock frequencies used for this simulation were 2.5MHz and 3.73MHz. The higher clock frequency was included in this test because, as was shown in the first section, a NETLIST based simulation predicts faster operation than actual circuit performance, and to account for the simulation's overly fast circuit speed prediction, the clock frequency had to be increased.

5.5. The Expected Performance of the ECG Processor

To estimate the performance of the ECG processor, it is necessary to find the time required to compute the similarity of one unknown matrix to eight reference matrices. This cannot be determined exactly because the computation time depends upon the number of non-zero elements in the unknown matrix, but a minimum time, a maximum time, and an expected time may be found. If all elements equal zero, then six clock cycles are required for each matrix element giving a total of 5800 clock cycles per classification. If all elements are non-zero, then 12 clock cycles are taken per matrix element and the total number of clock cycles per classification is 11600 clock cycles. From an inspection of a recognition matrix in [5], approximately 75% of the elements are zero which means the expected similarity computation time is 7300 clock cycles. Given a 400ns memory cycle time, this equals 2.9ms per classification, or a rate of 340 classifications per second. This computational rate, however, is not fully utilized in the current version of the ECG processor because its memory addressing capability is limited to 32 leads and the required average classification rate for 32 leads assuming an average heart rate of 60 beats per minute is 32 classifications per second. This high computational rate gives the ECG processor a comfortable margin against a sudden increase in the average heart rate and allows for future versions of the chip to have the memory addressing capability extended for possible use in multiple lead ECG monitoring systems.

To give a better understanding of the ECG processor's computational abilities, assembly language programs were written for the Intel 8086 and Motorola 68000 microprocessors to perform the identical functions as the ECG processor. As well, the computing ability of the FPS-100 array processor was also investigated as another means of computing the similarity function. The classification rate for each of these approaches was estimated by finding the time required to execute the assembly language programs. The results are summarized in Table 5.5. In all cases, 75% of the matrix elements were assumed to equal zero. The FPS-100 array processor estimation was based upon the 750ns per element computation time for computing the dot product between two matrices [46], and instead of the recognition matrix elements, the VAX 11/780 would pass the square-root values to the array processor. The square-root extraction would be performed by table look-up and the resulting data transfer rate would be 250ns per element. An attempt was made to measure the data transfer rate, but the VAX 11/780 system timer was too coarse with only a 10ms resolution.

	INTEL 8088 4.77 MHz clock (IBM PC)	MOTOROLA 68000 8 MHz clock	VAX 11/780 with FPS-100 Array Processor	ECG PROCESSOR 2.5 MHz clock
ESTIMATED CLASSIFICATION TIME FOR EIGHT CLASSES (milliseconds)	260	50	16	2.9
ESTIMATED COMPARISON RATE (matrices/second)	25	150	500	2760
NUMBER OF LEADS (average heartrate of 60 bpm with 8 classes per lead)	3	19	60	32

Table 5-5: The ECG performance versus microprocessors.

5.6. Summary

The beginning of this chapter discussed the simulation parameters for MOS IC's and compared some simulation results to measured values to check the validity of both the simulator and the simulation parameters. The RNL simulation based upon the extracted circuit predicted the ring oscillator frequency to within 2.6%.

The second part of the chapter showed the development of the logic circuits necessary to build the ECG processor. This included standard logic gates as well as static memory, PLA circuits, and input/output circuits.

The third section discussed the testing and test results of two fabricated IC's.

The fourth section told of how the ECG processor simulation was performed and reported that the simulation was successful.

The final part of this chapter discussed the expected performance of the

ECG processor and compared its performance to general purpose processors performing identical functions. The ECG processor's performance greatly exceeded the other means of computing the similarity value.

Chapter 6
Conclusions and Future Work

The objective of this thesis was to develop an IC which would improve the performance of an ECG analysis system. This was done by shifting the most computationally intensive part of the ECG analysis algorithm from the host computer to an IC developed specifically for these calculations. The performance of this device, measured in terms of the rate of comparing morphology of heart beats, was 2760 comparisons per second. This provided a comparison rate of 10 to 25 times faster than what could be achieved by either an Intel 8088 or a Motorola 68000 microprocessor based design.

As a starting point for this design, information was presented about ECG analysis in general and the analysis algorithm used for a proposed analysis system. Two important aspects of this algorithm directly related to the IC design were then discussed: one was the technique for generating recognition matrices; and the other was the algorithm used to compute the similarity between two recognition matrices. Two methods were described for generating the recognition matrices. These methods used integer arithmetic and were suitable for implementation on a microprocessor.

Chapter 3 introduced the principles of IC design and described the IC design software which was used to develop the ECG processor. Particular attention was given to logic and circuit simulation computer programs because they were the key to a successful IC design. Programs like ISPS, DABL, the Logician Design Editor, RNL, and SPICE were investigated as possible means by which the ECG Processor could be developed. As well, a computer program that used a look-up table to model the transistor characteristics and charge storage for all circuit nodes was written to perform circuit simulation for IC design. For a given test circuit, the simulation results of the new simulator compared well to those of SPICE while using only one-fiftieth of the computer time required by SPICE. Another important program developed for this project was a graphics program for plotting SPICE results in a logic analyzer format. A summary of 23 computer programs used as part of this IC design project is given at the end of Chapter 3.

In the first part of Chapter 4, the similarity computation algorithm was simplified from the Bhattacharyya distance function to a summation of the products of square-roots. Once the exact computational requirements were specified, the interface between the ECG processor and the host computer was discussed. The shared memory approach was selected as the means of moving data between the ECG processor and the host computer, and a memory map showing how the data was to be stored was developed. As well, other functions related to the ECG analysis algorithm such as the matrix update functions were described.

In the next section of Chapter 4, the major components of the IC design were identified, and techniques for square-root extraction, multiplication, data storage, and external memory address computation were discussed. For the square-root function, an 8-bit pipelined square-root extractor based upon a non-restoring root extraction algorithm was developed. In the case of the multiplier, more than one method of multiplication were considered. The first multiplier was an 8-bit by 8-bit signed multiplier which used canonical signed digit recoding to achieve an average multiplication time 41% faster than a similar design not using recoding. The cost for this faster multiplication was a 22% increase in the multiplier layout area. The second multiplier design that was selected for the final version of the ECG Processor design was a 4-bit by 4-bit pipelined unsigned multiplier. Finally, the required support hardware and software were outlined in order to describe how the ECG processor would become part of an ECG analysis system.

In Chapter 5, the circuits necessary to perform the functions presented in Chapter 4 were developed. First, the accuracy of the simulation software was checked by comparing simulation results to the measurements taken from a ring oscillator circuit. Circuit descriptions were then given for logic gates, add/subtract cells, shift-register cells, PLA circuits, memory circuits, input protection circuits, and output pad drivers. The testing of two sub-circuits, the static RAM and the accumulating adder/subtractor, was described, and the test results presented. In the final part of this chapter, the simulation for the complete ECG processor was discussed and, according to the simulation, the circuit would work as intended.

The main contributions of this thesis are:

1. A NETLIST description of a 8346 transistor circuit which computes the similarity between two recognition matrices,
2. A simulation of the NETLIST description to check its correctness,
3. The layouts for a pipelined square-root extractor, a 4x4 pipelined multiplier, an 8x8 CSD shift-and-add multiplier, an 8-word 16-bit word register file, and a PLA controller for the ECG processor,
4. The fabrication and testing of five sub-circuits including the register file, the 8-bit by 8-bit CSD recoded multiplier, the square-root extractor, a PLA-based finite-state machine, and the accumulating add/subtract unit of the CSD multiplier.

As a reference for future designs, the layout areas in fabrication scale units and the transistor counts for the major blocks of the IC design are summarized in Table 6.1.

6.1. Future Work

The purpose of designing this IC was to develop a circuit which would aid a computer system in the real-time analysis of ECG's. While the objective of designing this circuit has been met, much work remains in building a complete ECG analysis system. In applying this ECG Processor to ECG analysis, the following areas require further study:

Component	Area	Transistor count	Transistor density
4x4 Multiplier	$2.06 \times 10^6 \mu^2$	1431	695 tr/mm^2
Root Extractor	$1.40 \times 10^6 \mu^2$	1266	904 tr/mm^2
Control Unit	$1.07 \times 10^6 \mu^2$	763	713 tr/mm^2
Address Unit	-	1042	-
Register File	$1.05 \times 10^6 \mu^2$	1320	1257 tr/mm^2

Table 6-1: Major component areas and transistor counts.

1. Complete the layout and test of the ECG Processor. This involves combining the layouts for the multiplier, the square-root extractor, the controller, and the register file circuits into a circuit matching the NETLIST circuit description. This layout can then be checked by extracting the circuit description and performing an RNL simulation. The RNL simulation can use the same input stimulus developed for testing the NETLIST description.

2. Fully develop the shared memory interface. One of the approaches for shared memory design presented in this thesis could be selected and tested by using a microprocessor in place of the ECG Processor. In this way, development of the interface need not wait for the completion of the ECG Processor. To design an even more compact system, circuits for refreshing dynamic RAM and providing dual-port memory access could also be included on the same chip as the ECG processor.

3. Write computer programs to control the ECG Processor. The time critical operation of generating the recognition matrices from the ECG signal data should be written in assembly language, but the other programs like maintaining the classification and update queues, reading the ECG Processor's results, and completing the similarity calculations could be written in a programming language like 'C'.

4. Further study is needed on the required accuracy of the ECG Processor's computations. The accuracy of the current design could possibly be inadequate for reliable morphology recognition. However, its computational rate is higher than required, and so future versions of this chip could trade a slower computational rate for increased accuracy.

References

1. Feldman, C.L., "Trends in Computerized ECG Monitoring", in *Computer-Processed Electrocardiograms,* van Bemmel, J.H., Willems, J.L., eds., North-Holland Publishing Company, 1977.

2. Lown, B., Fakhro, A.M., Hood, W.B., et al, "The Coronary Care Unit. New perspectives and directions", *Journal of the American Medical Association,* Vol. 1991967, pages 188-198

3. Boudillon, P.J., Kilpatrick, D., "Interpretation of Electrocardiograms by Clinicians and by Computer", in *Computer-Processed Electrocardiograms,* van Bemmel, J.H., Willems, J.L., eds., North-Holland Publishing Company, 1977.

4. Thomas, L.J., Clark, K.W., Mead, C.N., Ripley, K.L., Spenner, B.F., Oliver, G.C., "Automated Cardiac Dysrhythmia Analysis", *Proceedings of the IEEE,* Sept 1979.

5. Bolton, R.J., "Representation and Pattern Recognition of Hilbert Transformed Electrocardiograms", Tech. report ee 83/9, University of Queensland, 1983.

6. Alexander, B., "MOS and CMOS Arrays", in *Gate Arrays,* Read, J.W., ed., McGraw Hill Book Company, 1985.

7. Adams, A.C., "Dielectric and Polysilicon Film Deposition", in *VLSI Technology,* Sze, S.M., ed., McGraw Hill Book Company, 1983.

8. Denyer, P.B., "Silicon compilers and VLSI", in *Semi-custom IC Design and VLSI,* Hicks, P.J., ed., Peter Peregrinus, 1983.

9. Okuda, N., Sugai, M., Goto, N., "Semicustom and Custom LSI Technology", *Proceedings of the IEEE,* Vol. 74, December 1986.

10. Grierson, J.R., "Selection of semi-custom technique, supplier, and design route", in *Semi-custom IC Design and VLSI,* Hicks, P.J., ed., Peter Peregrinus, 1983.

11. Slana, M.F., "Computer Elements for the 80's", *COMPUTER*, April 1979.

12. Univeristy of Washington, UW/NW VLSI Consortium, *VLSI Design Tools Reference Manual Release 3.0,* University of Washington, 1985.

13. Daisy Systems Corporation, *Chipmaster Reference Manual,* Daisy Systems Corp., 1986.

14. Mead, C., Conway, L., *Introduction to VLSI Systems,* Addison-Wesley, 1980.

15. Clark, J.W., "The Origin of Biopotentials", in *Medical Instrumentation Application and Design,* Webster, J.G., ed., Houghton Mifflin Company, 1978.

16. Goldman, M.J., *Principles of Electrocardiography,* Lange Medical Publications, 1982.

17. Summerall, C.P., Mangiaracina, J., McNeely, J., *Monitoring Heart Rhythm,* John Wiley and Sons, 1976.

18. Oliver, G.C., Ripley, K.L., Miller, J.P., Martin, T.F., "A Critical Review of Computer Arrhythmia Detection", in *Computer Electrocardiography: Present Status and Criteria,* Pordy, L., ed., Futura Press, 1977.

19. Noble, F.M., "The ARGUS Monitoring System: A Reappraisal", in *Computer-Processed Electrocardiograms,* van Bemmel, J.H., Willems, J.L., eds., North-Holland Publishing Company, 1977.

20. Meisel, W.S., *Computer-Oriented Approaches to Pattern Recognition,* Academic Press, 1972.

21. Mead, C.A., "Structural and Behavioral Composition of VLSI", in *VLSI 83 : VLSI Design of Digital Systems,* Aas, E.J., Anceau, F., eds., Elsevier Science Publishers, 1983.

22. Woods, M.H., "MOS VLSI Reliability and Yield Trends", *Proceedings of the IEEE*, Vol. 74, December 1986.

23. Niessen, C., "Hierachial Design Methodologies and Tools for VLSI Chips", *Proceedings of the IEEE*, Vol. 71, No. 1, January 1983.

24. Williams, T.W., Parker, K.P., "Design for Testability - A Survey", *Proceedings of the IEEE*, January 1983.

25. Jack, M.A., "Design for Testability", in *Semi-custom IC Design and VLSI,* Hicks, P.J., ed., Peter Peregrinus, 1983.

26. Radke, C.E., "Experience in VLSI Testing", *IEEE Design and Test of Computers*, February 1986.

27. Chawla, R.C., Gummel, H.K., Kozak, P., "MOTIS - An MOS Timing Simulator", *IEEE Transactions on Circuits and Systems*, December 1975.

28. Barbacci, M.R., "Instruction Set Processor Specification (ISPS): The Notation and Its Applications", *IEEE Transactions on Computers*, January 1981.

29. Intel, *Memory Components Handbook,* Intel Corporation, 1984.

30. Shojiro, A., "Semiconductor Memory Trends", *Proceedings of the IEEE*, December 1986".

31. Ramamoorthy, C.V, Goodman, J.R., Kim, K.H., "Some Properties of Iterative Square-rooting Methods", *IEEE Transactions on Computers*, August 1972.

32. Dean, K.J., "Cellular Logical Array for Extracting Square Root", *Electronics Letters*, July 1968.

33. Guild, H.H., "Cellular logical array for non-restoring square-root extraction", *Electronics Letters*, February 1970.

34. Majithia, J.C., Kitai, R., "A Cellular Array for the Nonrestoring Extraction of Square Roots", *IEEE Transactions on Computers*, December 1971.

35. Peled, A., "On the Hardware Implementation of Digital Signal Processors", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, February 1976.

36. Bird, V., "Two's complement to CSD Code Converter". Log book of University of Saskatchewan Summer Student

37. Jain, A.J., "Implementation Approaches in the Computer-Aided Design of CSD Digital Filters", Master's thesis, University of Saskatchewan, 1986.

38. Weste, N., Eshraghian, K., *Principles of CMOS VLSI Design: A Systems Perspective,* Addison-Wesley, 1985.

39. Cantrell, T., "Static RAM uses smarts to control dual-port access", *Electronic Design*, June 1986.

40. Bolton, R.J., Goulet, K.J., "A Custom Integrated Circuit for ECG Processing", *Computers in Cardiology*, October 1986.

41. Soden, J.M., Hawkins, C.F., "Test Considerations for Gate Oxide Shorts in CMOS ICs", *Design and Test of Computers*, August 1986.

42. Canadian Microelectronics Corporation, *Guide to the Integrated Circuit Implementation Services of the Canadian Microelectronics Corporation,* Canadian Microelectronics Corporation, 1986.

43. Ward, D.E., Doganis, K., "Optimized Extraction of MOS Model Parameters", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, October 1982.

44. Hodges, D.A., Jackson, H.G., *Analysis and Design of Inegrated Circuits,* McGraw-Hill, 1983.

45. Glasser, L.A., Dobberpuhl, D.W., *The Design and Analysis of VLSI Circuits,* Addison-Wesley Publishing Company, 1985.

46. Floating Point Systems Inc., *FPS-100 Math Library Volume 1,* Floating Point Systems, 1980.

Appendix A
ECG Processor Circuit Diagram

Figure A.1 shows the block diagram for the ECG processor. This diagram is based upon the top level of the hierarchical NETLIST description of this circuit; the component blocks and signal names correspond to those used in the NETLIST description. As part of the documentation important to understanding the operation of the ECG processor, a brief summary of the major component blocks and signals is given below.

A.1. Summary of the major circuit blocks

- 16-bit shift register - used to coordinate activity in the similarity calculation unit. A 1 bit is shifted through this register for every non-zero unknown matrix element. This bit identifies the location of this matrix element as it passes through the square-root pipeline, and it marks the first element from the unknown matrix as it passes through the multiplier and is added to the register file.

- addu - the external memory address unit. It is made up of registers and counters which count through the addresses of all matrix elements.

- addcon - a parallel loading shift register which selects the row for the register file. It provides sequential access for all words in this memory.

- control - a 5-input, 22-output PLA with 5 feedback lines. It provides the control signals which synchronize major on-chip functions. The inputs are read when ϕ_1 is high, and the outputs change when ϕ_2 goes high.

- mult - a 4x4 bit unsigned pipelined multiplier.

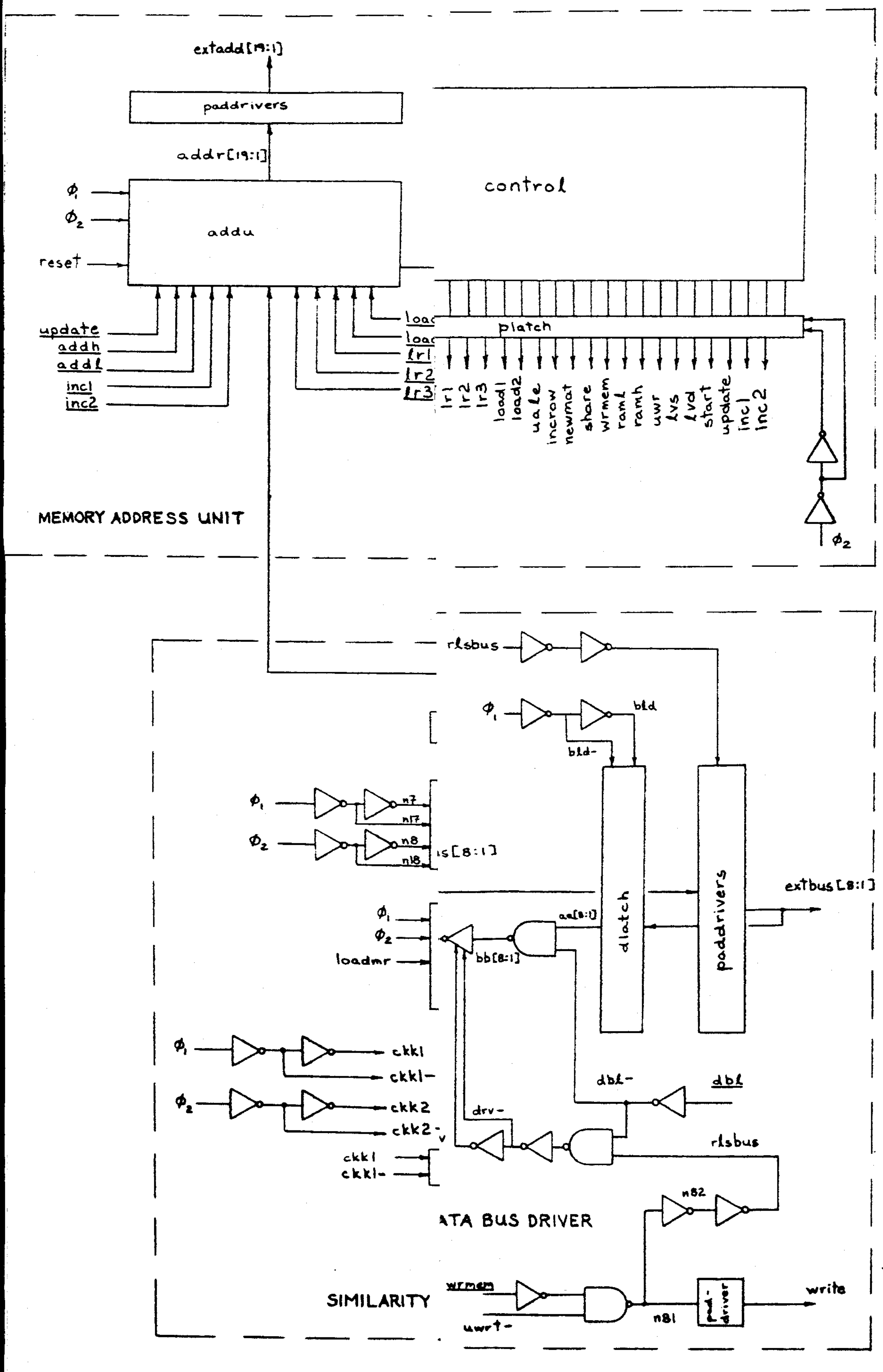


Figure A-1: The ECG

- ram - a static register file of 8 16-bit words.
- root - a pipelined square-root extractor.
- zerotest - an 8-input NAND gate which indicates when the bus contents are equal to zero.

A.2. Summary of the ECG Processor signals

<u>Signals generated by the ECG processor</u>

- ϕ_1 - phase 1 of a non-overlapping two phase clock signal.
- ϕ_2 - phase 2 of a non-overlapping two phase clock signal.
- extadd - the external address bus.
- extbus - the external data bus.
- shareout - indicates when an access is being made to shared memory.
- reset - a signal which resets the control unit to state 0 and loads the address counters with the with the address of the first 'stop-byte'.
- write - a signal which indicates a write operation to external memory.

<u>Signals from the control unit</u>

- addh - when asserted, it allows the 9 most significant bits of the address to change to their next computed value.
- addl - when asserted, it allows the 10 least significant bits of the address to change to their next computed value.
- clr- - the complement of the reset signal. When asserted, it forces the next state of the controller to be state 0.
- dbl - when asserted, it forces the the internal data bus to assume the value 0.
- incl - increments the 10-bit counter of the memory address unit by 1.

- inc2 - increments the 3-bit counter of the memory address unit by 1.
- incrow - used to advance the selected row of the the register file.
- load1 - resets the 10-bit counter of the memory address unit.
- load2 - resets the 3-bit counter of the memory address unit.
- lr1 - loads address register 1 (the next lead number to be classified) of the memory address unit with data from the data bus.
- lr2 - loads address register 2 (the next lead number and reference matrix for the update function) of the memory address unit with data from the data bus.
- lr3 - loads address register 3 of the memory address unit and the matrix update control register with the contents from the data bus.
- lvd - a signal to load a matrix update unit latch with a value from the reference matrix.
- lvs - a signal to load a matrix update unit latch with a value from the unknown matrix.
- newmat - asserted once at the start of every matrix calculation. It resets the selected row of the register file to row 1, and it causes the latch of the register file output to clear its contents so that the accumulation operations begin without adding to the results of previous accumulation operations.
- share - asserted when an access to shared memory is being made.
- start - is asserted when reading a matrix element from the unknown matrix.
- ramh - when asserted, it connects the upper 8 bits of the register file to the data bus.
- raml - when asserted, it connects the lower 8 bits of the register file to the data bus.
- update - signals the memory address unit to output the address for the next operands of the matrix update function.

- wrmem - used to signal a write operation to external memory.

<u>Signals generated by the matrix update unit</u>

- u5[8:1] - the output of the 8-bit adder.
- u6[8:1] - the output of the 8-bit adder NOR'd with the adder's carry out. This rounds down all overflows to 255.
- ucon[7:2] - the contents of the control register which controls the operation of the matrix update unit.
- uld/uld- - signals to load the control register.
- uld1/uld1- - signals to load an unknown matrix element value into a latch.
- uld2/uld2- - signals to load an unknown matrix element value into a latch.
- uwrt- - a signal which is asserted when the results of the matrix update function are to be written to external memory. It will only be asserted if the control bits are set accordingly.

<u>Signals from the memory address unit</u>

- addr[19:1] - the address value used as input to the pad drivers.
- eom - the end-of-matrix signal which is asserted when the last matrix element is being accessed.

<u>Signals from the data bus driver</u>

- aa[8:1] - the contents of the external data bus latch.
- bb[8:1] - the contents of the external data bus latch AND'ed by the dbl- signal.
- bld/bld- - causes the external data bus latch to be transparent when ϕ_1 is high.
- bus[8:1] - the internal data bus of the ECG processor.
- rlsbus - when asserted, the external data bus is driven by the

ECG processor (for the write operation), otherwise, the ECG processor data bus port is in a high impedance state.

Signals from the similarity calculation unit

- a[4:1] - the output of the square-root extractor.
- accum - resets the register file to row 1 and begins the accumulation process for the next eight values produced by the multiplier.
- b[8:1] - the output of the multiplier.
- c[16:1] - the contents of the register file output latch.
- ckk1/ckk1- - a buffered ϕ_1 signal and its complement.
- ckk2/ckk2- - a buffered ϕ_2 signal and its complement.
- d[16:1] - the sum of e[8:1] and c[16:1].
- done - signals the last of 8 values produced by the multiplier circuit.
- e[8:1] - the contents of the multiplier output latch.
- end - asserted when the data bus contents equal zero.
- endload - signals the last of 8 values to be read from the reference matrices.
- loadmr - asserted when the multiplier's multiplier register is to be loaded.
- n2 - asserted when a value read from the unknown matrix does not equal zero. Provides the input to the 16-bit shift register.
- out1[16:1] - the output of the register file.
- out2[16:1] the complemented output of the register file.
- rdy - indicates when the selected row of the register file may change.
- row8 - signals that row 8 of the register file has been selected.

- rowone - when asserted, the first row of the register file is selected.
- reginc - enables the incrementing of the row select circuit for the register file.
- toggl - increments the selected row of the register file by one.
- wr - indicates a write operation to the register file.
- zero- - not asserted when the data bus contents are equal to zero.

Appendix B
NETLIST Description of the ECG Processor

To complete the documentation of the ECG processor, a NETLIST description is presented here. The description is hierarchical and it is stored on disk with an hierarchical directory structure. A summary of this directory structure may be found after the NETLIST description.

B.1. The NETLIST Circuit Description

The top of the hierarchical description

```
; ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
; master control store - a PLA which provides the control signals
;   for all parts of the ECG processor
;
; +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

(load "/pub/goulet/net4/control/control.net")
(load "/pub/goulet/net4/lib/dlatch.net")
(load "/pub/goulet/net4/control/platch.net")
(load "/pub/goulet/net4/lib/linv.net")
(load "/pub/goulet/net4/lib/lnand.net")
(load "/pub/goulet/net4/lib/rs.net")


(node bus ref addr)
(node input output
        zero eom end accum endload row8
        addl addh dbl lr1 lr2 lr3 load1 load2 uale incrow
        newmat share shareout wrmem write raml ramh uwr lvs lvd start
        inc1 inc2 in out reset clr- update
        ck1 ck1- ck2 ck2- phi1 phi2
        n80 n81 n82 rlsbus uwrt-)
        (control output input)

        (connect in.1 zero)
        (connect in.2 eom)
        (connect in.3 end)
        (connect in.4 endload)
        (connect in.5 row8)
        (connect out.27 inc2)
        (connect out.26 inc1)
        (connect out.25 update)
        (connect out.24 start)
        (connect out.23 lvd)
        (connect out.22 lvs)
        (connect out.21 uwr)
        (connect out.20 ramh)
        (connect out.19 raml)
        (connect out.18 wrmem)
        (connect out.17 share)
        (connect out.16 newmat)
        (connect out.15 incrow)
        (connect out.14 uale)
        (connect out.13 load2)
        (connect out.12 load1)
        (connect out.11 lr3)
        (connect out.10 lr2)
        (connect out.9 lr1)
        (connect out.8 dbl)
        (connect out.7 addh)
        (connect out.6 addl)
; output latches (dynamic)
        (repeat i 1 27 (platch output.i ck2 ck2- out.i))

; input latches (feedback latches are resettable to force state
;                   zero upon reset)
        (repeat i 1 5 (dlatch in.i ck1 ck1- input.i))
        (repeat i 6 10 (rdlatch in.i clr- ck1 ck1- input.i))

; connect feedback
        (repeat i 1 5 (connect in.(+ (- 5 i) 6) out.i))

; clock drives
        (linv6 phi1 ck1-)
        (linv6 ck1- ck1)
        (linv10 phi2 ck2-)
        (linv10 ck2- ck2)

        (linv reset clr-)
;
```

```
; ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
; memory address counters (mac) - stores address pointers for
;       matrices and counters for matrix calculations
;
; ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
(load "/pub/goulet/net4/lib/lxor.net")
(load "/pub/goulet/net4/address/counter/countb.net")
(load "/pub/goulet/net4/address/addu.net")
(load "/pub/goulet/net4/lib/paddriver.net")

(node extaddr)

       (addu bus lr1 lr2 lr3 load1 load2 inc1 inc2 addl addh
             update reset phi1 phi2 addr eom)
;
; address line drivers
       (repeat i 1 19 (paddr addr.i extaddr.i))


;
; give a realistic load
       (repeat i 1 19 (capacitance extaddr.i 10))


;
; paddriver for shared memory request line
       (paddr share shareout)
       (capacitance shareout 10)


;
; paddriver for write memory pin
       (linv wrmem n80)
       (lnand uwrt- n80 n81)
       (paddr n81 write)
       (linv n81 n82)
       (linv6 n82 rlsbus)

       (capacitance write 10)


; ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
; data bus interface (dbi) - provides the drive and tri-state capability
;         for both the external data bus and the internal data bus
;
; ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
(node extbus aa bb oe n60 n61 n62 dbl- bld bld- drv drv-)


       (linv rlsbus n62)
```

```
        (linv10 n62 oe)
        (repeat i 1 8 (paddrt bus.i oe extbus.i))

        (repeat i 1 8 (dlatch extbus.i bld bld- aa.i)
                       (lnand dbl- aa.i bb.i)
                       (tbuf bb.i drv drv- bus.i))

        (linv dbl dbl-)

        (lnand dbl- rlsbus n61)
        (linv10 n61 drv-)
        (linv10 drv- drv)
        (linv6 phi1 bld-)
        (linv6 bld- bld)
;
; give a realistic load to the bus
        (repeat i 1 8 (capacitance extbus.i 10))
; ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
;
; SIMILARITY CALCULATION UNIT v3.0
; feb 16, 1987
;
; inputs: bus = input and output to SCU
;   newmat = indicates beginning of new matrix-sets accumulator to zero
;   incrow = causes the next higher address of ram to be accessed on phi
;   raml = connects low order word of ram to bus
;   ramh = connects high order word of ram to bus
;   sysset = initializes a flip flop on system reset
;
; outputs: last = goes high when last value is being added to ram
;          done = goes high one cycle after last
;
; ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

(node phi1 phi2 ckk1 ckk1- ckk2 ckk2-)

(node n1 n2 n3 n3a n4 n4a n5 n6 n7 n8 n9 n10 n11 n12 n13 n14 n15 n16
     n17 n18)
(node n50 n51 n52 n53 n54)
(node wr a b c d e row rdy out1 out2 so cry sysset)
(node zero- loadmr last reginc done toggl rowone enl enl-)
(node enh enh-)

(load "/pub/goulet/net4/lib/lnor.net")
(load "/pub/goulet/net4/lib/addcell.net")

(load "root/latch.net")
```

```
(load "root/root.net")

(load "ram/6ram.net")
(load "ram/cdrive.net")
(load "ram/rdrive.net")
(load "ram/mono.net")
(load "ram/precharg.net")
(load "ram/addcon/mono2.net")
(load "ram/addcon/shift.net")
(load "ram/addcon/addcon.net")
(load "ram/ram.net")

(load "mult/rlatch.net")
(load "mult/madcell.net")
(load "mult/mult.net")

(load "zerotest/zerotest.net")

; test for bus=0
        (zerotest bus zero-)
        (linv6 zero- zero)

; shift register which provides control signals
        (linv10 phi1 ckk1-)
        (linv10 ckk1- ckk1)
        (linv10 phi2 ckk2-)
        (linv10 ckk2- ckk2)
        (lnand start zero- n1)
        (linv n1 n2)
        (latch n2 ckk1 ckk1- ckk2 ckk2- so.1)
   (repeat i 2 16 (latch so.(- i 1) ckk1 ckk1- ckk2 ckk2- so.i))

; taps are placed on the shift register to obtain timed control signals
        (lnand phi1 so.4 n3)
        (linv6 n3 loadmr)
        (lnand phi1 so.7 n3a)
        (linv6 n3a endload)
        (lnand phi1 so.8 n4)
        (linv6 n4 accum)
        (linv so.9 n4a)
        (linv6 n4a reginc)
        (lnand phi1 so.15 n5)
        (linv6 n5 last)
        (lnand phi1 so.16 n6)
        (linv6 n6 done)

; square-root extractor
        (linv phi1 n17)
        (linv phi2 n18)
        (linv20 n17 n7)
        (linv20 n18 n8)
        (root bus n7 n8 a)

; multiplier
        (mult a loadmr phil phi2 b)

; ram used as accumulating registers
        (ram d wr rdy phil wr row out1 out2)

        (linv row.8 n54)
        (linv6 n54 row8)

        (addcon rowone toggl sysset rdy row)

        (rs accum done n10 n11)

        (linv6 n10 end)

        (lnand n10 phi2 n14)
        (linv n14 wr)

        (lnor accum newmat n12)
        (linv n12 rowone)

        (rs reginc done n50 n51)
        (lnor n50 incrow n52)
        (linv n52 n53)
        (lnand n53 phil n13)
        (linv n13 toggl)

; resttable latch used for accumulator calculations
        (rs newmat done n15 n16)
        (repeat i 1 16 (rdlatch out1.i n16 ckk1 ckk1- c.i))


; latch for multiplier result
        (repeat i 1 8 (dlatch b.i ckk1 ckk1- e.i))

; adder for accumulator
        (repeat i 9 16 (connect GND e.i))
        (connect GND cry.0)
        (repeat i 1 16 (addcell e.i c.i cry.(- i 1) d.i cry.i))

; tri-state buffers connecting RAM to bus
        (linv6 ramh enh-)
        (linv6 enh- enh)
        (linv6 raml enl-)
        (linv6 enl- enl)
        (repeat i 1 8 (tbuf out2.i enl enl- bus.i))
        (repeat i 9 16 (tbuf out2.i enh enh- bus.(- i 8)))

;
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;
; Matrix Update Unit (MUU)
;
; permits the clearing (elements=0), transfer, and addition of
; two matrices with one being a reference matrix, and the other
; being the unknown matrix (in shared ram)
;
; +++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

(node n70 n71 n72 n73 n74 n75 n76 uld uld- uld1 uld1- uld2 uld2-
        half1 half1- half2 half2- u1 u2 u3 u4 u5 u6 ucon ucry
        ubus ubus-)

;
; control register - stores control bits
        (linv6 lr3 uld-)
        (linv6 uld- uld)
        (repeat i 2 7 (dlatch bus.i uld uld- ucon.i))

;
; two resettable input registers
        (linv6 lvs uld1-)
        (linv6 uld1- uld1)
        (linv ucon.4 n75)
        (repeat i 1 8 (rdlatch bus.i n75 uld1 uld1- u1.i))

        (linv6 lvd uld2-)
        (linv6 uld2- uld2)
        (linv ucon.5 n76)
        (repeat i 1 8 (rdlatch bus.i n76 uld2 uld2- u2.i))

;
; multiplexers provide divide by 2 function
        (linv6 ucon.2 half1-)
        (linv6 half1- half1)
        (repeat i 1 8   (ptrans half1 u1.i u3.i)
                        (etrans half1- u1.i u3.i)
                        (ptrans half1- u1.(+ i 1) u3.i)
                        (etrans half1 u1.(+ i 1) u3.i))
        (connect GND u1.9)
        (linv6 ucon.3 half2-)
        (linv6 half2- half2)
        (repeat i 1 8   (ptrans half2 u2.i u4.i)
                        (etrans half2- u2.i u4.i)
                        (ptrans half2- u2.(+ i 1) u4.i)
                        (etrans half2 u2.(+ i 1) u4.i))
        (connect GND u2.9)

; adder with an output limit of 255 (if overflow, output=255)
;  actual output is inverted
        (repeat i 1 8 (addcell u3.i u4.i ucry.(- i 1) u5.i ucry.i)
                        (lnor ucry.8 u5.i u6.i))
        (connect GND ucry.0)

;
; tri-state output buffer
;    < the tri-state lines are under-driven to  slow the enable >
;    < output transition.  This was done to allow the busdriver >
;    < unit time to release the bus before the update adder was >
;    < connected to the bus (yes, I know, not good design       >
;    < practice, but it was either this or a long wire from one >
;    < side of the chip to another)                              >

        (linv uwrt- ubus)
        (linv ubus ubus-)
        (repeat i 1 8 (tbuf u6.i ubus ubus- bus.i))

;
; write signal generate under control of control register

        (linv share n70)
        (linv n70 n71)
        (lnand n70 ucon.6 n72)
        (lnand n71 ucon.7 n73)
        (lnand n72 n73 n74)
        (lnand n74 uwr uwrt-)
```

<u>The pipelined square-root extractor</u>

```
; a circuit for a pipelined square-root extractor
;
; input: in(8:1) in.8 is most significant digit
;        ck1 phase 1 clock signal
;        ck2 phase 2 clock signal
; output: out(4:1)
;

(load "/pub/goulet/net4/root/latch.net")
```

```
(load "/pub/goulet/net4/lib/cascell.net")
(load "/pub/goulet/net4/root/row1.net")
(load "/pub/goulet/net4/root/row2.net")
(load "/pub/goulet/net4/root/row3.net")
(load "/pub/goulet/net4/root/row4.net")

(macro root (in ck1 ck2 out)
(local  n2 n3 n4 n5 n6   ck1- ck2- )

        (row1 in ck1 ck1- ck2 ck2- n2)
        (row2 n2 ck1 ck1- ck2 ck2- n3)
        (row3 n3 ck1 ck1- ck2 ck2- n4)
        (row4 n4 ck1 ck1- ck2 ck2- n5)
; output buffers
        (repeat i 1 4 (linv n5.i n6.i) (linv n6.i out.i))
        (linv20 ck1 ck1-)
        (linv20 ck2 ck2-)
)
; 1st stage of an eight stage root extractor

(macro row1 (in ck1 ck1- ck2 ck2- n)
        (local x dum c )

        (repeat i 1 7 (latch in.i ck1 ck1- ck2 ck2- n.i))
        (latch in.8 ck1 ck1- ck2 ck2- x.1)
        (cascell VDD x.1 n.7 GND n.8 n.9)
        )
; 2nd stageof an eight stage root extractor

(macro row2 (in ck1 ck1- ck2 ck2- n)
        (local x dum c)

        (repeat i 1 5 (latch in.i ck1 ck1- ck2 ck2- n.i))

        (latch in.6 ck1 ck1- ck2 ck2- x.1)
        (latch in.7 ck1 ck1- ck2 ck2- x.2)
        (latch in.8 ck1 ck1- ck2 ck2- x.3)
        (latch in.9 ck1 ck1- ck2 ck2- n.7)
        (cascell VDD x.1 n.5 GND n.6 c.1)
        (cascell x.2 GND c.1 VDD n.8 c.2)
        (cascell GND x.3 c.2 n.7 dum.1 n.9)
        )

; 3rd stage of an eight stage root extractor

(macro row3 (in ck1 ck1- ck2 ck2- n)
        (local x dum c)
        (repeat i 1 3 (latch in.i ck1 ck1- ck2 ck2- n.i))
        (latch in.4 ck1 ck1- ck2 ck2- x.1)
        (latch in.5 ck1 ck1- ck2 ck2- x.2)
        (latch in.6 ck1 ck1- ck2 ck2- x.3)
        (latch in.7 ck1 ck1- ck2 ck2- n.8)
        (latch in.8 ck1 ck1- ck2 ck2- x.4)
        (latch in.9 ck1 ck1- ck2 ck2- n.6)

        (cascell VDD x.1 n.3 GND n.4 c.1)
        (cascell x.2 GND c.1 VDD n.5 c.2)
        (cascell n.8 x.3 c.2 n.6 n.7 c.3)
        (cascell GND x.4 c.3 n.6 dum.1 n.9)
        )
; stage 4 of an eight stage root extractor

(macrc row4 (in ck1 ck1- ck2 ck2- n)
        (local x dum c)

        (latch in.1 ck1 ck1- ck2 ck2- x.7)
        (repeat i 2 5 (latch in.i ck1 ck1- ck2 ck2- x.(- i 1)))
        (latch in.6 ck1 ck1- ck2 ck2- n.3)
        (latch in.7 ck1 ck1- ck2 ck2- x.5)
        (latch in.8 ck1 ck1- ck2 ck2- n.4)
        (latch in.9 ck1 ck1- ck2 ck2- n.2)

        (cascell VDD x.1 x.7 GND dum.2 c.1)
        (cascell x.2 GND c.1 VDD dum.3 c.2)
        (cascell n.3 x.3 c.2 n.2 dum.4 c.3)
        (cascell n.4 x.4 c.3 n.2 dum.5 c.4)
        (cascell GND x.5 c.4 n.2 dum.1 n.1)
        )
; latch

(macro latch (in ck1 ck1- ck2 ck2- out)
        (local n1 n2 n3 n4 n5 n6 n7 n8 n9 n10 n11)

        (ptrans in VDD n8 3 3)
        (ptrans ck1- n8 n1 3 3)
        (etrans ck1 n1 n9 3 3)
        (etrans in n9 GND 3 3)

        (ptrans n1 VDD n2 5 3)
        (etrans n1 n2 GND 3 3)

        (ptrans n2 VDD n3 3 3)
        (ptrans ck1 n3 n1 3 3)
        (etrans ck1- n1 n4 3 3)
        (etrans n2 n4 GND 3 3)
        (ptrans n2 VDD n10 3 3)
        (ptrans ck2- n10 n5 3 3)
        (etrans ck2 n5 n11 3 3)
        (etrans n2 n11 GND 3 3)

        (ptrans n5 VDD out 6 3)
        (etrans n5 out GND 4 3)

        (ptrans out VDD n6 3 3)
        (ptrans ck2 n6 n5 3 3)
        (etrans ck2- n5 n7 3 3)
        (etrans out n7 GND 3 3)

        )
```

<u>The pipelined 4x4 bit multiplier</u>

```
; a 4x4 bit pipelined multiplier

(macro mult (a loadmr phil phi2 c)

        (local a1 a2 a3 b1 b2 b3 c1 c2 co1 co2 co3 ck1 ck2 ck1- ck2- mr1
m m- l l-)

; register for the MULTIPLIER
        (linv6 loadmr l-)
        (linv6 l- l)
        (repeat i 1 4  (dlatch a.i l l- mr1.i))
; buffers for the MULTIPLIER register output
        (repeat i 1 4 (linv6 mr1.i m-.i)
                        (linv6 m-.i m.i))
        )


; resettable latch
        (repeat i 1 4 (rlatch a.i m.1 ck1 ck1- ck2 ck2- a1.i))

; shift by 1 register
        (repeat i 1 4 (latch a.i ck1 ck1- ck2 ck2- b1.(+ i 1)))

; adder 1
        (connect GND a1.6) (connect GND a1.5) (connect GND b1.1)
        (connect GND b1.6) (connect GND co1.0)
  (repeat i 1 6 (maddcell a1.i b1.i co1.(- i 1) m.2 m-.2 c1.i co1.i))

; intermediate result register
        (repeat i 1 6 (latch c1.i ck1 ck1- ck2 ck2- a2.i))
        (latch co1.6 ck1 ck1- ck2 ck2- a2.7)
```

```
; shift by 2 register
        (repeat i 2 5 (latch b1.i ck1 ck1- ck2 ck2- b2.(+ i 1)))

; adder 2
        (connect GND b2.1) (connect GND b2.2) (connect GND b2.7)
        (connect GND co2.0)
  (repeat i 1 7 (maddcell a2.i b2.i co2.(- i 1) m.3 m-.3 c2.i co2.i))

; intermediate result register
        (repeat i 1 7 (latch c2.i ck1 ck1- ck2 ck2- a3.i))
        (latch co2.7 ck1 ck1- ck2 ck2- a3.8)

; shift by 3 register
        (repeat i 3 6 (latch b2.i ck1 ck1- ck2 ck2- b3.(+ i 1)))

; adder 3
        (repeat i 1 3 (connect GND b3.i))
        (connect GND b3.8) (connect GND co3.0)
  (repeat i 1 8 (maddcell a3.i b3.i co3.(- i 1) m.4 m-.4 c.i co3.i))

; some drivers for the clock signals
        (linv20 phi1 ck1-)
        (linv20 ck1- ck1)
        (linv20 phi2 ck2-)
        (linv20 ck2- ck2)
)
; a latch with a clear input
(macro rlatch (in clr- ck1 ck1- ck2 ck2- out)
        (local n1 n2 n3 n4 n5 n6 n7 n8 n9 n10 n11 n12)

        (ptrans clr- VDD n1 3 3)
        (ptrans in VDD n8 3 3)
        (ptrans ck1- n8 n1 3 3)
        (etrans ck1 n1 n9 3 3)
        (etrans in n9 n12 3 3)
        (etrans clr- n12 GND 3 3)

        (ptrans n1 VDD n2 5 3)
        (etrans n1 n2 GND 3 3)

        (ptrans n2 VDD n3 3 3)
        (ptrans ck1 n3 n1 3 3)
        (etrans ck1- n1 n4 3 3)
        (etrans n2 n4 GND 3 3)

        (ptrans n2 VDD n10 3 3)
        (ptrans ck2- n10 n5 3 3)
        (etrans ck2 n5 n11 3 3)
        (etrans n2 n11 GND 3 3)

        (ptrans n5 VDD out 6 3)
        (etrans n5 out GND 4 3)

        (ptrans out VDD n6 3 3)
        (ptrans ck2 n6 n5 3 3)
        (etrans ck2- n5 n7 3 3)
        (etrans out n7 GND 3 3)

        )
```

<u>The description of the external memory address unit</u>

```
;
; netlist for the address unit macro
;

(node cnt hadd deda sima)

(macro addu (datab lr1 lr2 lr3 load1 load2 inc1 inc2 eal eah
               dedadd reset phi1 phi2 addr eom)

(local n1 n2 n3 n4 n5 n6 n7 n8 n9 n10 n11 n12 n13 n14 n15 n16
        n17 n18 n19 n20 n21 n22 n23 n24 n25 n26 n27 n28 n29 n30
        n31 n32 n33 n34 n35 dum1 dum2 qb qb- qb2 qb2- init dum3
        n36 n37 n38 n39 n40 n41)
;
; low order address latch
        (lnand eal phi2 n1)
        (linv10 n1 n2)
        (linv10 n2 n3)
        (repeat i 1 10 (dlatch cnt.i n2 n3 addr.i))
;
; comparator signals end of matrix calculations
;(changed to '3' for test purposes-should be 961)
        (ptrans cnt.1 VDD n4 9 3)
        (ptrans cnt.2 VDD n4 9 3)
        (etrans VDD n4 GND 3 18)

        (etrans cnt.3 n5 GND 6 3)
        (etrans cnt.4 n5 GND 6 3)
        (etrans cnt.5 n5 GND 6 3)
        (etrans cnt.6 n5 GND 6 3)
        (etrans cnt.7 n5 GND 6 3)
        (etrans cnt.8 n5 GND 6 3)
        (etrans cnt.9 n5 GND 6 3)
        (etrans cnt.10 n5 GND 6 3)
        (ptrans GND VDD n5 3 12)

        (linv n4 n6)
        (lnand n6 n5 n35)
        (linv n35 eom)
;
; counter for low order address
;       load-counter logic
        (lnand phi1 load1 n7)
        (linv10 n7 n8)
        (linv10 n8 n9)
;       increment-counter logic
        (lnand phi1 inc1 n10)
        (linv10 n10 n11)
        (linv10 n11 n12)
;       counter-logic
;       increment counter logic
        (lnand phi2 inc1 n36)
        (linv10 n36 n37)
        (linv10 n37 n38)
        (repeat i 1 10  (countb init.i qb.(- i 1) n8 n9 n11 n12
                         n37 n38 qb.i cnt.i))
        (connect qb.0 VDD)
;
;       logic to control the address loaded into the counter
; (modified to load '20' for test purposes - should be 978
        (rs reset inc1 dum3 n13)
        (linv6 n13 n15)
        (connect init.10 GND)
        (connect init.9 GND)
        (connect init.8 GND)
        (connect init.7 GND)
        (connect init.6 GND)
        (connect init.5 n15)
        (connect init.4 GND)
        (connect init.3 n15)
        (connect init.2 GND)
        (connect init.1 GND)


;
; high order address word latch
        (lnand phi2 eah n16)
        (linv10 n16 n17)
        (linv10 n17 n18)
        (repeat i 1 9 (dlatch hadd.i n17 n18 addr.(+ i 10)))
;
;   multiplexor to choose between similarity calculation address and
;   update function address
```

```
        (linv10 dedadd n19)
        (linv10 n19 n20)
        (repeat i 1 9
        (ptrans n20 sima.i hadd.i 3 3)
        (etrans n19 sima.i hadd.i 3 3)
        (ptrans n19 deda.i hadd.i 3 3)
        (etrans n20 deda.i hadd.i 3 3)
        )
;
; high order similarity address counter
        (lnand phi1 lr1 n21)
        (linv6 n21 n22)
        (linv6 n22 n23)
        (repeat i 1 6 (dlatch datab.i n22 n23 sima.(+ i 3)))

        (lnand phi1 inc2 n24)
        (linv6 n24 n25)
        (linv6 n25 n26)
        (lnand phi1 load2 n27)
        (linv6 n27 n28)
        (linv6 n28 n29)
        (lnand phi2 inc2 n39)
        (linv6 n39 n40)
        (linv6 n40 n41)
        (repeat i 1 3 (countb GND qb2.(- i 1) n28 n29
                        n25 n26 n40 n41 qb2.i sima.i))
        (connect qb2.0 VDD)
;
; high order update function address latch
        (lnand lr2 phi1 n30)
        (linv10 n30 n31)
        (linv10 n31 n32)
        (repeat i 1 8 (dlatch datab.i n31 n32 deda.i))
        (lnand lr3 phi1 n33)
        (linv n34 n33)
        (dlatch datab.1 n33 n34 deda.9)

        )
;
; a single bit of a ripple carry counter
;   -- increments on phi2, output changes on phi1
;
(macro countb (di ci l l- ck1 ck1-  ck2 ck2- co s)

        (local n1 n2 n10 n11 n13 n12 n14 n15 n18 n19 n20
        a b)

        (dlatch s ck2 ck2- a)
        (lnand a ci n1)
        (linv6 n1 co)
        (lxor a ci b)
;
; a loadable latch - loads on phi1
        (ptrans b VDD n10 5.4 3)
        (ptrans ck1- n10 n12 5.4 3)
        (etrans ck1 n12 n14 3 3)
        (etrans b n14 GND 4 3)

        (ptrans di VDD n15 5.4 3)
        (ptrans l- n15 n12 5.4 3)
        (etrans l n12 n18 3 3)
        (etrans di n18 GND 4 3)

        (linv n12 s)
        (ptrans s VDD n19 3 3)
        (ptrans l n19 n13 3 3)
        (ptrans ck1 n13 n12 3 3)
        (etrans ck1- n12 n11 3 3)
        (etrans l- n11 n20 3 3)
        (etrans s n20 GND 3 3)

)
```

<u>A description of the register file</u>

```
;  circuit for a complete 8 word 16-bit word memory


(macro ram (in wr rdy- phi1 phi2 row out1 out2)
        (local  d d-  sel  rdy w w- p-)

        (linv10 wr w-)
        (linv10 w- w)

        (repeat i 1 8
                (repeat j 1 16
                        (6ram d.j d-.j sel.i)))

        (repeat i 1 16
                (cdrive in.i w w- p- d.i d-.i))

        (repeat i 1 8
                (rdrive row.i rdy  sel.i))

        (precharg phi1 phi2 wr d.1 d-.1 p- rdy rdy-)
```

```
; output buffers
        (repeat i 1 16 (linv d-.i out1.i)
                        (linv d.i out2.i)       )
        )
; 6-transistor ram memory cell

(macro 6ram (d d- sel)
        (local a a-)

        (threshold a .4 .8)
        (threshold a- .4 .8)

        (etrans sel a d 3 3)
        (etrans sel a- d- 3 3)

        (ptrans a VDD a- 3 3)
        (etrans a a- GND 6 3)
        (ptrans a- VDD a 3 3)
        (etrans a- a GND 6 3)


        )
; bit(column) line driver for ram memory

(macro cdrive (i w w- p- d d-)

        (local n1 n2 n3 n4 n5)

        (ptrans i VDD n1 5.4 3)
        (etrans i n1 GND 3 3)

        (ptrans i VDD n2 20 3)
        (ptrans w- n2 d- 20 3)
        (etrans w d- n3 10 3)
        (etrans i n3 GND 10 3)

        (ptrans n1 VDD n4 20 3)
        (ptrans w- n4 d 20 3)
        (etrans w d n5 10 3)
        (etrans n1 n5 GND 10 3)

        (ptrans p- VDD d- 15 3)
        (ptrans p- VDD d 15 3)

        )

; row select driver
(macro rdrive (row rdy  sel)
        (local n1 n2 )


        (ptrans rdy VDD n1 5.4 3)
        (ptrans row VDD n1 5.4 3)
        (etrans row n1 n2 3 3)
        (etrans rdy n2 GND 3 3)

        (ptrans n1 VDD sel 10 3)
        (etrans n1 sel GND 5 3)

        )

; a monostable multivibrator (one-shot) for use in self-timed circuits

(macro mono (i o)
        (local n1 n2 n3 n4 n5 n6)

(threshold o .2 .5)     ; low high-threshold because charge sharing causes
                        ; output to generate an 'x' state when input goes
                        ; from 1 to 0

        (ptrans i VDD n1 3 3)
        (etrans i n1 GND 3 6)
        (ptrans n1 VDD n2 3 3)
        (etrans n1 n2 GND 3 6)
        (ptrans n2 VDD n3 3 3)
        (etrans n2 n3 GND 3 6)
        (ptrans n3 VDD n5 3 3)
        (etrans n3 n5 GND 3 6)
        (ptrans n5 VDD n6 3 3)
        (etrans n5 n6 GND 3 6)

        (ptrans n6 VDD o 5.4 3)
        (ptrans i VDD o 5.4 3)
        (etrans n6 o n4 3 3)
        (etrans i n4 GND 3 3)

        )

; self-timed circuit to control precharge of bit lines and row select
; enable signals. Signal generation begins on a leading clock edge


(macro precharg (phi1 phi2 w d d- p- rdy rdy-)
        (local n1 n2 n3 n4 n5 n6 n7 n8)

        (mono phi1 n1)
        (mono phi2 n2)
        (lnand n1 n2 n3)

        (lnand d d- n4)
        (lnand n4 n5 n6)
        (linv w n5)

        (rs n3 n6 n8 n7)
        (linv10 n8 p-)
        (linv6 n7 rdy-)
        (linv10 rdy- rdy)              )
```

The zerotest circuit

```
;
; zerotest has a low output only when all 9 inputs are low

(macro zerotest (word zero-)
        (local n1  n5   n10)

        (etrans word.1 GND n1 5 3)
        (etrans word.2 GND n1 5 3)
        (etrans word.3 GND n1 5 3)
        (etrans word.4 GND n1 5 3)
        (ptrans GND VDD n1 3 4)

        (etrans word.8 GND n5 5 3)
        (etrans word.7 GND n5 5 3)
        (etrans word.6 GND n5 5 3)
        (etrans word.5 GND n5 5 3)
        (ptrans GND VDD n5 3 4)

        (ptrans n1 VDD zero- 5.4 3)
        (ptrans n5 VDD zero- 5.4 3)
        (etrans n1 zero- n10 3 3)
        (etrans n5 n10 GND 3 3)

        )
```

Definition of the addcell macro

```
; one-bit full adder
; a & b are 1 bit from each operand, ci is carry in, co is carry out

; s = sum

(macro addcell (a b ci s co)
        (local n2 n3 n4)

        (lxor b a n2)
        (lnand6 ci n2 n3)
        (lnand6 a b n4)
        (lnand6 n3 n4 co)
        (lxor ci n2 s)  )
```

<u>Definition of the controlled add/subtract cell</u>

```
; one-bit full adder-subtractor
; a & b are 1 bit from each operand, ci is carry in, co is carry out
; p=0 for add, p=1 for subtract (b-a) , s is sum

(load "/pub/goulet/net/lib/lxor.net")
(load "/pub/goulet/net/lib/lnand.net")

(macro cascell (a b ci p s co)
        (local n1 n2 n3 n4)

        (lxor a p n1)
        (lxor b n1 n2)
        (lnand ci n2 n3)
        (lnand n1 b n4)
        (lnand n3 n4 co)
        (lxor ci n2 s)  )
```

<u>Definition of some buffer circuits</u>

```
;
; a resettable non-inverting buffer
;
(macro buffer1 (in out)
        (local n1 )

        (ptrans in VDD n1 3 3)
        (etrans in n1 GND 3 3)
        (ptrans n1 VDD out 7.2 3)
        (etrans n1 out GND 3.6 3)

        )
;
; a selection of non-inverting buffers
;
(macro rbuffer1 (in out clr-)
        (local n1 n2)

        (ptrans in VDD n2 3 3)
        (ptrans clr- VDD n2 5.4 3)
        (etrans in n2 n1 3 3)
        (etrans clr- n1 GND 3 3)
        (ptrans n2 VDD out 7.2 3)
        (etrans n2 out GND 3.6 3)

        )
```

<u>Definition of some inverter circuits</u>

```
; file includes some miscellaneous inverters of varying drive capability
;
; the number in the inverter name stands for the width of the n-channel
; transistor

(macro linv (a b)
        (ptrans a VDD b 5.4 3)
        (etrans a b GND 3 3)
        )

(macro linv6 (a b)
        (ptrans a VDD b 12 3)
        (etrans a b GND 6 3)
        )


(macro linv10 (a b)
        (ptrans a VDD b 18 3)
        (etrans a b GND 10 3)
        )

(macro linv15 (a b)
        (ptrans a VDD b 25 3)
        (etrans a b GND 15 3)
        )

(macro linv30 (a b)
        (ptrans a VDD b 50 3)
        (etrans a b GND 30 3)
        )

(macro linv40 (a b)
        (ptrans a VDD b 70 3)
        (etrans a b GND 40 3)
        )

(macro linv20 (a b)
        (ptrans a VDD b 35 3)
        (etrans a b GND 20 3)
        )
;
; tri-state inverter
(macro tbuf (a en en- b)
        (local n1 n2)
        (ptrans a VDD n1 10 3)
        (ptrans en- n1 b 10 3)
        (etrans en b n2 5 3 )
        (etrans a n2 GND 5 3)
)
```

<u>Definition of a d-latch</u>

```
;
; dlatch - static d type latch

(macro dlatch (d c c- q)
        (local n1 n2 n3 n4 n5)

        (ptrans d VDD n1 5.4 3)
        (ptrans c- n1 n3 5.4 3)
        (etrans c n3 n2 3 3)
        (etrans d n2 GND 3 3)

        (ptrans q VDD n4 3 3)
        (ptrans c n4 n3 3 3)
        (etrans c- n3 n5 3 3)
        (etrans q n5 GND 3 3)

        (ptrans n3 VDD q 5.4 3)
        (etrans n3 q GND 3 3)


        )
;
; rdlatch - a static d - type latch with a clr- input

(macro rdlatch (d clr- c c- q)
        (local n1 n2 n3 n4 n5 n6 n7)

        (ptrans d VDD n1 5.4 3)
        (ptrans c- n1 n3 5.4 3)
        (etrans c n3 n2 3 3)
        (etrans d n2 n6 3 3)
        (etrans clr- n6 GND 3 3)

        (ptrans clr- VDD n3 3 3)
        (ptrans q VDD n4 3 3)
        (ptrans c n4 n3 3 3)
        (etrans c- n3 n5 3 3)
        (etrans q n5 n7 3 3)
        (etrans clr- n7 GND 3 3)

        (ptrans n3 VDD q 5.4 3)
        (etrans n3 q GND 3 3)


        )
```

Definition of some logic gates

```
; 3 input nand gate
(macro l3nand (a b c d)
        (local n1 n2)

        (ptrans a VDD d 5.4 3)
        (ptrans b VDD d 5.4 3)
        (ptrans c VDD d 5.4 3)
        (etrans a d n1 3 3)
        (etrans b n1 n2 3 3)
        (etrans c n2 GND 4 3)
        )
; 2-input nand gate  c = a.b
;
(macro lnand (a b c)
        (local n1)

        (ptrans a VDD c 5 3)
        (ptrans b VDD c 5 3)
        (etrans a c n1 4 3)
        (etrans b n1 GND 5 3)
        )

(macro lnand6 (a b c)   ; larger version of nand gate
        (local n1)

        (ptrans a VDD c 10 3)
        (ptrans b VDD c 10 3)
        (etrans a c n1 6 3)
        (etrans b n1 GND 6 3)
        )

;
; nand gate with the function = not( (a+b).c )
```

```
(macro lornand (a b c d)
        (local n1 n2)

        (ptrans a VDD n1 5.4 3)
        (ptrans b n1 d 5.4 3)
        (ptrans c VDD d 5.4 3)
        (etrans c d n2 3 3)
        (etrans a n2 GND 3 3)
        (etrans b n2 GND 3 3)
        )
; 2-input nor gate  c = a + b
(macro lnor (a b c)
        (local n1)

        (ptrans a VDD n1 6 3)
        (ptrans b n1 c 6 3)
        (etrans a c GND 3 3)
        (etrans b c GND 3 3)
        )

; 2-input XOR gate --- 10-transistor design; fully restored logic level
; c = a (xor) b
(macro lxor (a b c)
        (local n1 n2 n3 n5)

        (ptrans b VDD n1 8 3)
        (ptrans a n1 n2 8 3)
        (etrans b n2 GND 3 3)
        (etrans a n2 GND 3 3)
        (ptrans n2 n3 c 8 3)
        (etrans n2 c GND 3 3)
        (etrans a c n5 3 3)
        (etrans b n5 GND 3 3)
        (ptrans a VDD n3 8 3)
        (ptrans b VDD n3 8 3)

        )

; 2-input xnor gate ;10 transistor design - fully restored logic levels
; c = a (xnor) b
(macro lxnor (a b c)
        (local n1 n2 n3 n4)
        (ptrans a VDD n1 8 3)
        (ptrans b n1 c 8 3)
        (etrans n2 c n3 3 3)
        (etrans b n3 GND 3 3)
        (etrans a n3 GND 3 3)
        (ptrans n2 VDD c 5 3)
        (ptrans b VDD n2 5 3)
        (ptrans a VDD n2 5 3)
        (etrans b n2 n4 3 3)
        (etrans a n4 GND 3 3)

        )
```

<u>Definition of the output pad drivers</u>

```
;
; pad-driver netlists used for 3-u designs
;
; tri-state pad-driver

(macro paddrt (in oe pad)

        (local nl n2 n3 n4 n5 n6 n7 n8 )

        (ptrans in VDD n8 5.4 3)
        (etrans in n8 GND 3 3)

        (ptrans oe VDD nl 5.4 3)
        (ptrans n8 VDD nl 5.4 3)
        (etrans oe nl n2 3 3)
        (etrans n8 n2 GND 3 3)

        (ptrans oe VDD n5 5.4 3)
        (etrans oe n5 GND 3 3)

        (ptrans n8 VDD n3 12 3)
        (ptrans n5 n3 n4 12 3)
        (etrans n8 n4 GND 3 3)
        (etrans n5 n4 GND 3 3)

        (ptrans n4 VDD n6 30 3)
        (etrans n4 n6 GND 16 3)

        (ptrans nl VDD n7 30 3)
        (etrans nl n7 GND 16 3)

        (ptrans n6 VDD pad 138 3)
        (etrans n7 pad GND 96 3)
)

;
; plain pad driver
;
(macro paddr (in pad)
        (local n1 n2 n3)
        (ptrans in VDD n1 5.4 3)
        (etrans in n1 GND 3 3)
        (ptrans n1 VDD n2 15 3)
        (etrans n1 n2 GND 10 3)
        (ptrans n2 VDD n3 50 3)
        (etrans n2 n3 GND 30 3)
        (ptrans n3 VDD pad 138 3)
        (etrans n3 pad GND 96 3)
)
```

Definition of the SR latch

```
; rs flip-flop using 2 nor gates

(macro rs (s r q q-)
        (local n1 n2)

        (ptrans s VDD n1 7.2 3)
        (ptrans q n1 q- 7.2 3)
        (etrans s q- GND 3 3)
        (etrans q q- GND 3 3)

        (ptrans r VDD n2 7.2 3)
        (ptrans q- n2 q 7.2 3)
        (etrans r q GND 3 3)
        (etrans q- q GND 3 3)

        )
```

B.2. Summary of the Directory Structure

This is a summary of the hierachical directory structure for the NETLIST description as listed by the unix command *ls*.

```
*** The Topmost directory name is NET4 ***

address         ram             root            control
lib             test4.net       zerotest        mult


net4/address:
addu.net        counter

net4/address/counter:
countb.net
```

```
net4/control:
control.peg      control.net      platch.net

net4/lib:
addcell.net      delay1.net       linv.net         lxnor.net        rs.net
buffers.net      delay2.net       lnand.net        lxor.net
cascell.net      dlatch.net       lnor.net         paddriver.net

net4/mult:
rlatch.net       madcell.net      mult.net

net4/ram:
6ram.net         precharg.net     addcon           cdrive.net
ram.net          mono.net         rdrive.net

net4/ram/addcon:
addcon.net       mono2.net        shift.net

net4/root:
latch.net        row2.net         row3.net         root.net
row4.net         row1.net

net4/zerotest:
zerotest.net
```

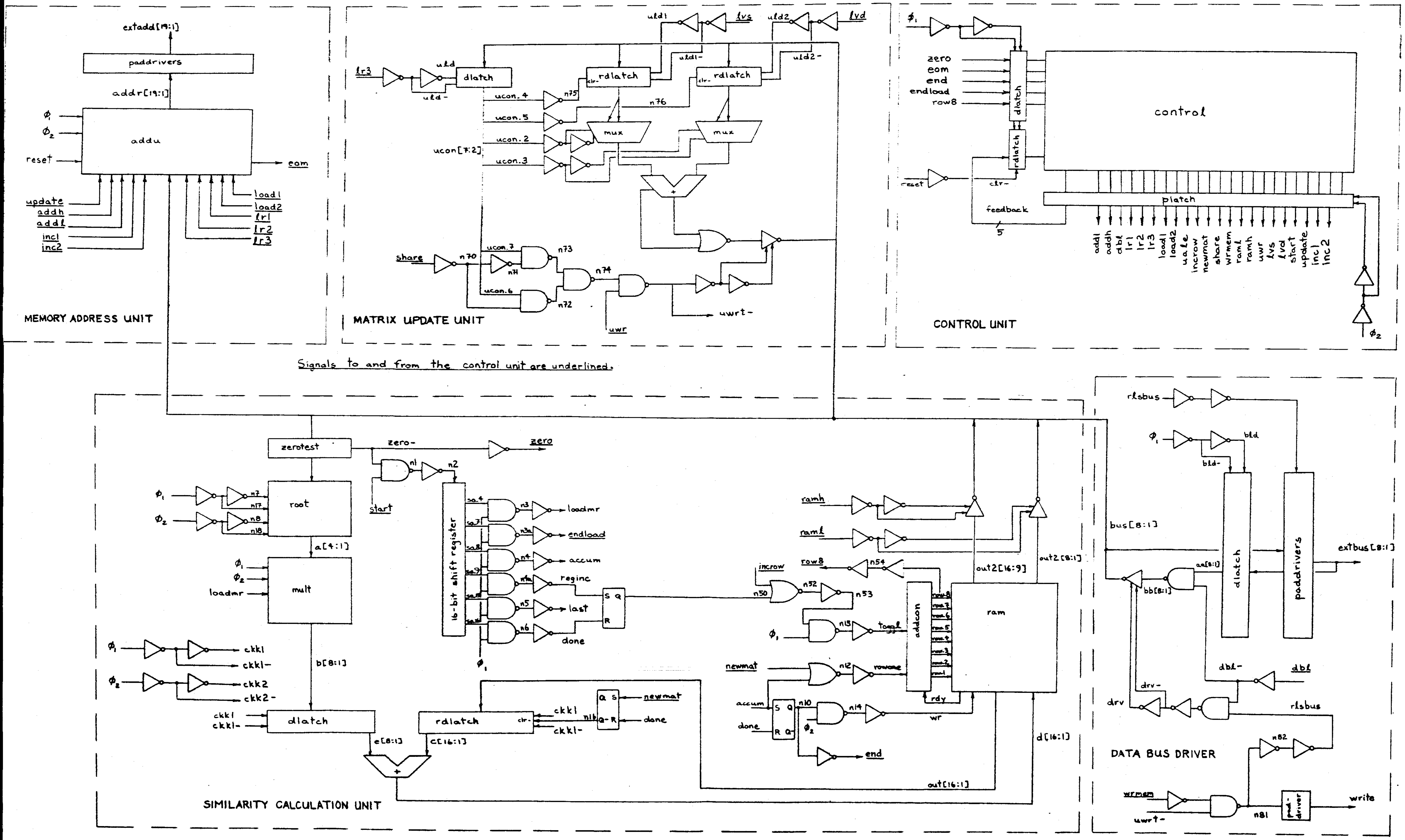


Figure A.1: The ECG Processor block diagram