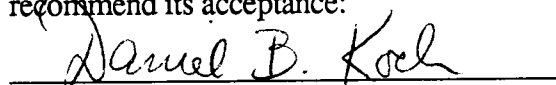


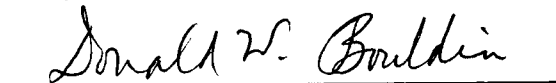
To the Graduate Council:

I am submitting here with a thesis written by Rodney Buford Whitted entitled "A VLSI Implementation Of Jansson's Iterative Deconvolution Algorithm." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Electrical Engineering.


Paul Benjamin Crilly, Major Professor

We have read this thesis and
recommend its acceptance:





Accepted for the Council:


Associate Vice Chancellor
and Dean of The Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Rodney B. Whitted
Date 1 MAY 1992

A VLSI Implementation of Jansson's Iterative Deconvolution Algorithm

A Thesis
Presented for the Master of Science Degree
The University of Tennessee, Knoxville

Rodney Buford Whitted
May, 1992

DEDICATION

This thesis is dedicated to Nancy.

Without your love, hard work and support this would not have been possible.

I love you Sweetheart!!!

ACKNOWLEDGMENTS

I would like to thank my major professor Dr. Paul Crilly, for his guidance and advice during this time at The University. Dr. Donald Bouldin and Dr. Daniel Koch, the remaining members of my thesis committee, are very much appreciated for their time and attention. Thanks also go to the UT Center for Measurement and Controls and the Bell South Foundation for funding this research.

Special thanks must go to my parents, Buford and Shirley Whitted who have always encouraged me to get a good education. Thanks Mom and Dad!

ABSTRACT

This thesis will present a Very Large Scale Integration (VLSI), Digital Signal Processing (DSP) chip that will significantly improve the processing throughput for Jansson's iterative deconvolution algorithm. The chip will enable Jansson's algorithm to be used for real-time DSP applications which formerly due to speed limitations were not possible. This will enable researchers to take further advantage of the features unique to iterative deconvolution.

The chip will process 8-bit, 2's complement input data at a 520.625 KHz sample rate to produce 8-bit results. The deconvolution is computed using a 32-point coefficient vector. The convolution is carried out by a time multiplexed, finite impulse response (FIR) filter array. All processing is done in a bit-parallel, pipelined fashion.

The chip layout was accomplished using a high level hardware description language called SDL which is part of the Lager IV software package. This design approach produced a parameterized chip description, allowing future design changes or improvements to be made rapidly without necessitating a complete redesign effort.

TABLE OF CONTENTS

CHAPTER		PAGE
1	Deconvolution Development	1
	Introduction	1
	Deconvolution Development	4
	Thesis Objective	9
2	Design Architecture	10
	System Level Architecture	10
	Chip Level Architecture	11
	Convolver Subsystem	13
	Arithmetic Subsystem	14
3	Circuit Implementation	16
	Target Architecture	16
	Convolver Subsystem	18
	Iteration Processor	18
	Multiply / Accumulate Circuit	19
	State Memory	21
	Convolver Array	22
	Arithmetic Subsystem	23
	Pipeline Implementation	24
	Programming Control	27
	Chip Integration	29
4	Testing and Design Evaluation	32
	Test Design Considerations	32
	Test Results	33
	Convolver Subsystem	33
	Arithmetic Subsystem	34
	Design Evaluation	34

TABLE OF CONTENTS (continued)

CHAPTER	PAGE
5	Summary, Conclusions and Future Work
	39
	Summary
	39
	Conclusions
	40
	Future Work
	40
	System Modifications
	40
	Performance Enhancements
	42
BIBLIOGRAPHY	44
APPENDIX	47
VITA	166

LIST OF FIGURES

FIGURE	PAGE
1.1 The effects of convolutional blurring and additive noise.	3
1.2 The relaxation function $r\{x_e^k\}$.	7
1.3 Application of Jansson's method to data containing noise.	8
2.1 Linear array of DSP ICs making up a deconvolution system.	11
2.2 Floorplan of iterative deconvolution IC.	12
2.3 General architecture of an FIR filter (convolver/correlator).	13
2.4 Multiplexed architecture of an FIR filter (convolver/correlator).	15
2.5 Pipelined architecture for the arithmetic subsystem.	15
3.1 Components of a cascadable multiplexed IP.	18
3.2 Relationship between clock signals used by IC.	19
3.3 The MAC circuit.	20
3.4 The ACC datapath.	20
3.5 The XSTATE datapath	21
3.6 The HSTATE datapath	22
3.7 Array of iteration processors and formatting block.	23
3.8 The FORMAT circuit.	24
3.9 Arithmetic Pipeline (A_PIPE).	25
3.10 8-bit delay line for the original signal estimate (GSTATE).	26
3.11 Stages 1 and 3 of the arithmetic pipeline as an integrated datapath.	27
3.12 Programming shift register path of deconvolution IC.	28
3.13 State diagram of the program control finite state machine.	29
3.14 Program control logic (CONTROL).	30
3.15 Block diagram of the IC core (CORE).	31
4.1 Data sets used to compare mathematical formats.	36
4.2 Error vs. iteration plots for floating-point and fixed-point formats.	38

Chapter 1 - Deconvolution Background

Introduction

The purpose of all types of measurement instrumentation is plain: to provide a value (or values) for some particular quantity or quality that is of interest to the user. To this end people have developed devices which range from the simple thermometer to the complex electronic components which make up modern radio telescopes. In most cases, even the most sophisticated instruments yield a less than perfect measurement due to external disturbances or because of resolution limitations within the instrument itself. This fact is quite evident in the field of electronic signal processing where signals of interest are subjected to two major types of corruption: 1) spurious electrical noise and 2) the system's response characteristics. These problems create a fundamental limitation to the performance of such systems.

The problems of electronic noise in measurement systems can be addressed in several ways. Perhaps the most common method used to reduce noise is frequency filtering. This involves the use of electronic circuits which reduce or eliminate the unwanted noise in the signal by attenuating the signal energy found in a specific frequency interval. This approach may not be applicable if the measurement information resides in, or near, the same frequency interval as that occupied by the noise. If instruments are operated in electrically noisy environments electromagnetic shielding can be used to protect an instrument. This shielding attenuates the noise energy by shunting it to an electrical ground node before it reaches the instrument electronics.

The spatial relationships between components which form the instrument itself can sometimes lead to unwanted noise in the measurement. Electrical cross-talk between parallel traces on a printed circuit board offers a good example of this kind of noise generation. The designer of measurement instrumentation must address these problems

and attempt to minimize their effects on system performance. Unfortunately, it is not possible to completely eliminate electrical noise and thus measurements produced by the instrument will always be effected.

Although the system response characteristics always degrade a measurement to a certain extent they do not always cause a problem. They do become a problem when measurements are corrupted by undesired contributions which should have been associated with neighboring measurements. When these unwanted components add to a measurement in a certain linear fashion, the distortion produced can be described by convolution mathematics. An example of this phenomenon can be taken from photography. A camera can be thought of as a measurement system whose output is a photograph which records the spatial distribution of light in a scene. Let the focussing of the camera's lens represent the system response of the camera. If the lens is properly focussed then a sharp image of the scene is recorded on the film, corresponding to an instrument with a system response which has very little effect on the measurement. Now, suppose the lens is slightly out of focus, corresponding to a system response which has the effect of redistributing the light (which should form a sharp edge in an ideal image) to different areas of the film thus recording a blurred image.

Several types of instrumentation are greatly affected by convolution processes. In most systems the measurement signal can be modeled by the system's impulse response function distorting the data and additive noise further corrupting the data. This is stated mathematically as follows:

$$g = h * x + v \quad (1.1)$$

where g is the observed signal, x is the ideal or true signal, h is the system impulse response function, v is additive noise and $*$ is the convolution operator. Terms g , h , x and v are functions of time t or sequence i . To illustrate, Figure 1.1a shows an ideal signal and Figure 1.1b shows the effects of distortion and noise.

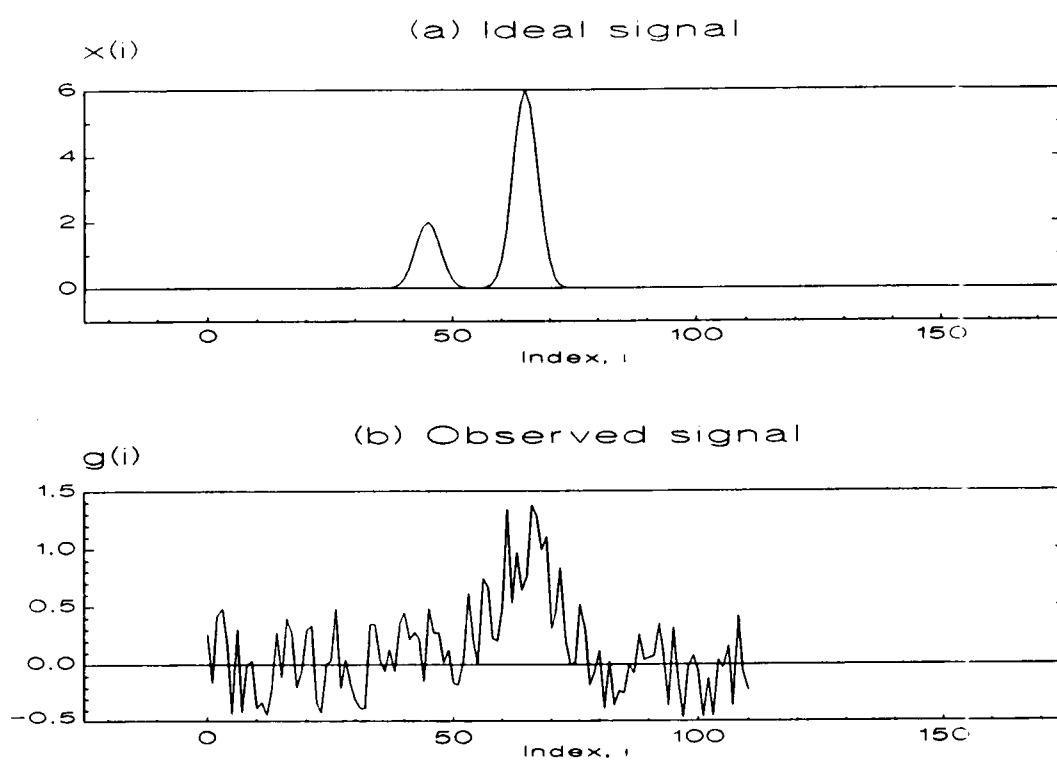


Figure 1.1 The effects of convolutional blurring and additive noise.

Deconvolution Development

Initial attempts to solve the problems caused by the noise and blurring described in Eq. 1.1 used Fourier transform techniques to obtain an estimate of the true signal as follows:

$$\mathbf{x}_e = F^{-1}\{\mathbf{G}/\mathbf{H}\} \quad (1.2)$$

where $\mathbf{x}_e$ is the estimate of $\mathbf{x}$, F^{-1} is the inverse Fourier transform operator, $\mathbf{G}$ and $\mathbf{H}$ are the Fourier transforms of $\mathbf{g}$ and $\mathbf{h}$ respectively. This type of deconvolution is known as inverse filtering. Unfortunately, since most processes are bandlimited, $\mathbf{H}$ will have near-zero values at some frequencies where because of noise, $\mathbf{G}$ may have finite values. This may cause the estimate produced by using Eq.1.2 to contain a significant amount of spurious components.

In 1931, Van Cittert proposed an iterative deconvolution method for restoring degraded spectroscopy data[2]. He suggested the use of the image signal $\mathbf{g}$, in Eq. 1.1 as a first approximation, $\mathbf{x}_e^0$, to the object data signal $\mathbf{x}$. By blurring $\mathbf{x}_e^0$, a first approximation to the image data, $\mathbf{g}_e^0$, could be found. This would be the form $\mathbf{g}$ would take if $\mathbf{x}_e^0$ were the true object. This appears to be a useless exercise since the image data $\mathbf{g}$ is already known, but this blurring does serve a useful purpose. The error in the image data, $\mathbf{g} - \mathbf{g}_e^0$, is in some way related to the error in the object estimate, $\mathbf{x} - \mathbf{x}_e^0$. Van Cittert applied the image estimate error as a correction to the object estimate, yielding a new object estimate:

$$\mathbf{x}_e^{k+1} = \mathbf{x}_e^k + [b(\mathbf{g} - (\mathbf{h} * \mathbf{x}_e^k))] \quad (1.3)$$

where $\mathbf{x}_e^k$ is the k th estimate of the true object $\mathbf{x}$, b is a constant, k is the iteration number,

and

$$\mathbf{x}_e^0 = \mathbf{g}. \quad (1.4)$$

One might expect that this method would give better estimates since the corrections are made gradually and there are no divisions. However, it can be shown that if the iteration number k is large and

$$|\mathbf{H}(f)| > 0 \text{ for all values of } f, \quad (1.5)$$

Van Cittert's method converges to the solution obtained by the inverse filter method. The iterative method has an advantage over inverse filtering since iterations can be stopped if the solution starts to diverge or if spurious peaks become too large[1].

If the data is known not to contain negative values, then constraints can be incorporated into Eq. 1.3 to take advantage of this fact. This is done as follows:

$$\mathbf{x}_e^{k+1} = p\mathbf{x}_e^k + (\mathbf{g} - \mathbf{h} * p\mathbf{x}_e^k) \quad (1.6)$$

where

$$p = \begin{cases} 1 & \mathbf{x}_e^k > 0 \\ 0 & \mathbf{x}_e^k < 0 \end{cases} \quad (1.7)$$

The constraint operator p causes the truncation of all negative artifacts during each iteration[3].

A similar technique involves modifying the correction term $(\mathbf{g} - \mathbf{h} * \mathbf{x}_e^k)$ by a constant, b that causes corrections to be made to $\mathbf{x}_e^k$ when $\mathbf{x}_e^k$ has negative values. This is done as follows[1]:

$$\mathbf{x}_e^{k+1} = \mathbf{x}_e^k + \mathbf{r}\{\mathbf{x}_e^k\}(\mathbf{g} - \mathbf{h}^* \mathbf{x}_e^k) \quad (1.8)$$

where

$$\mathbf{r}\{\mathbf{x}_e^k\} = \begin{cases} \mathbf{b} & \mathbf{x}_e^k > 0 \\ 0 & \mathbf{x}_e^k < 0 \end{cases} \quad (1.9)$$

While both of the methods just described eliminate or reduce negative artifacts, they accomplish this by truncation. This truncation may also eliminate other important information as well. In addition, the constraints used in these methods only enforce the non-negativity of the resulting estimates.

In most cases, much better restoration results can be obtained by using some form of constraining operation which causes the restored estimate to lie within some physically realizable boundaries. These bounds may be in the form of amplitude constraints which cause the solution to lie between lower and upper amplitude values. Spatial bounds may also be used when the solution is known to be zero-valued outside of a well defined region. It has been shown that the use of these constraining operations allow restoration of Fourier frequencies which were lost during the measurement process due to the imperfect nature of the measuring instrument[1]. Thus the instrument's user gains access to information which would be unavailable otherwise.

Jansson[1] proposed an iterative technique which incorporates both upper and lower amplitude constraints and is given as follows:

$$\mathbf{x}_e^{k+1} = \mathbf{x}_e^k + \mathbf{r}\{\mathbf{x}_e^k\} [\mathbf{g} - \mathbf{h}^* \mathbf{x}_e^k] \quad (1.10)$$

and

$$\mathbf{r}\{\mathbf{x}_e^k\} = \mathbf{b} [1 - 2/(c-a) | \mathbf{x}_e^k - (c+a)/2 |] \quad (1.11)$$

where x_e^k and x_e^{k+1} are the k th and the $(k+1)$ th estimate of x respectively. The relaxation function, $r\{x_e^k\}$, constrains the estimate to be between its physical lower bound of a and its upper bound of c . Eq. 1.11 is shown graphically in Figure 1.2. The relaxation constant b is chosen such that the number of iterations are kept to the minimum while at the same time allowing the solution to converge. Jansson's method has advantages over similar iterative methods in that the constraining operation is done gradually and without truncation of the non-physical values.

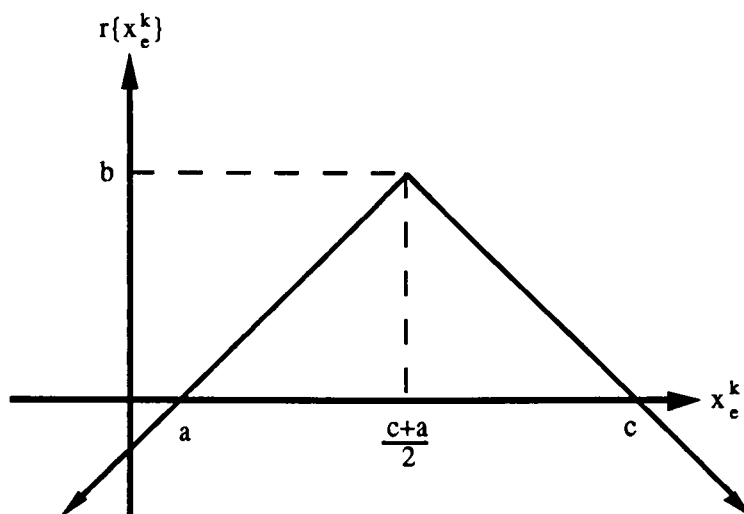


Figure 1.2: The relaxation function $r\{x_e^k\}$.

To illustrate the performance of Jansson's method, Figure 1.3a shows an ideal signal waveform. Figure 1.3c shows the results of restoring the corrupted waveform in Figure 1.3b by applying Eqs. (1.10) and (1.11). There are numerous other cases where Jansson's method has been successfully employed to restore degraded data[1,4-7].

Although there are numerous deconvolution techniques, the fundamental issues for a given method of signal recovery are how well it performs with noisy data, what upper bound is allowed on the distortions before the method fails to provide adequate estimates and what *a priori* knowledge about the data is required. In general, restoration methods

that take advantage of all known *a priori* information about the ideal signal will provide correspondingly better results. Another significant issue that limits the use of a given algorithm is its throughput. This is a significant issue in real time signal processing situations and is often a deciding factor in choosing one restoration algorithm over another. In evaluations of error performance, quantitative accuracy, repeatability, rate of convergence, and ability to work with noisy data, Crilly has shown that the method proposed by Jansson provides the best overall performance[7].

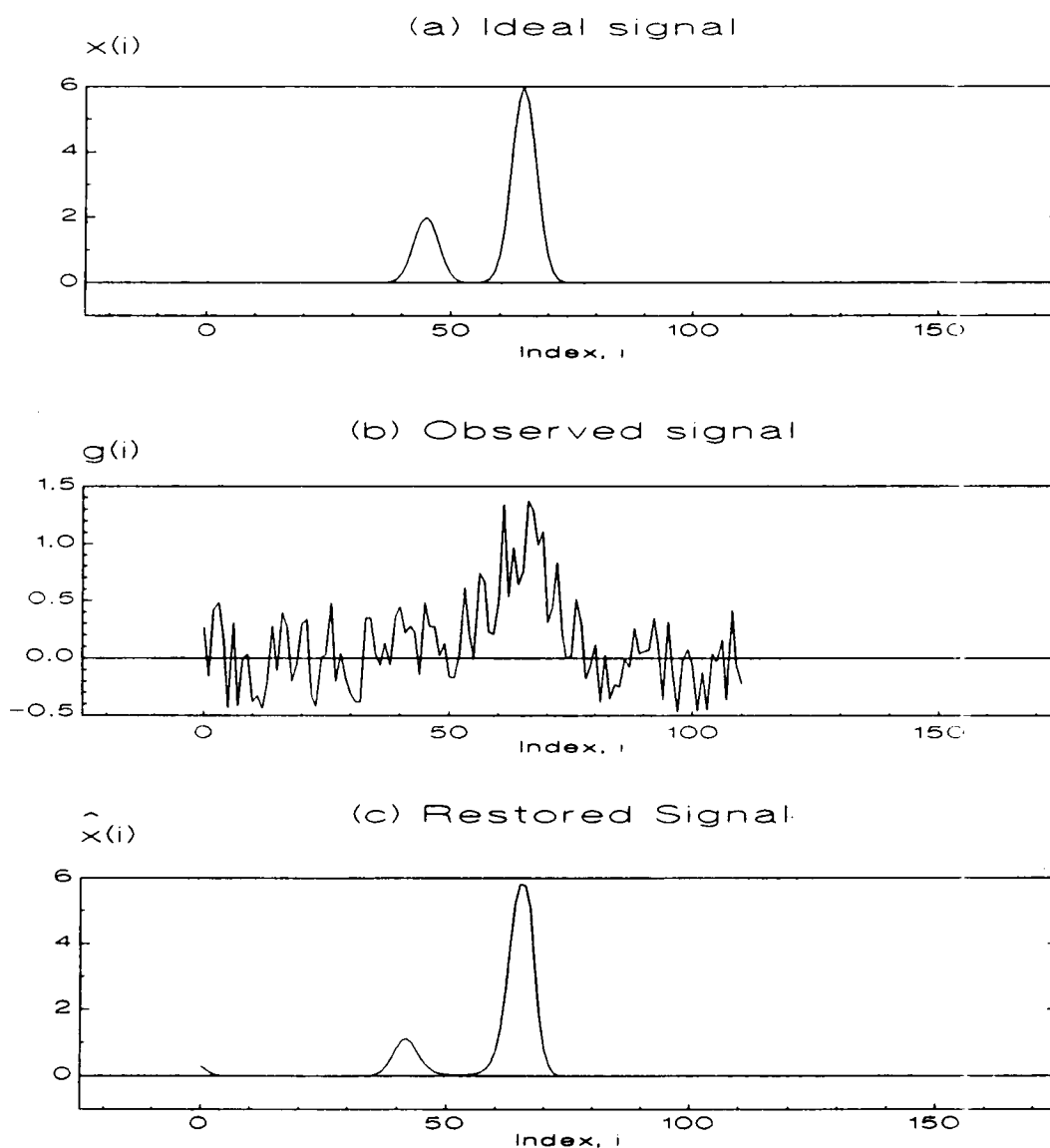


Figure 1.3: Application of Jansson's method to data containing noise.

Thesis Objective

In the past, iterative deconvolution algorithms have been implemented with general purpose computers. While this is satisfactory for spectroscopic and chromatographic data, there are numerous other applications where this approach is unacceptably slow. It is believed that by enhancing the speed performance of these algorithms, an increased number of applications can benefit from the use of numerical deconvolution.

This thesis describes the design of a dedicated VLSI integrated circuit (IC) which will compute Jansson's iterative deconvolution algorithm in real-time. The choice of implementing the algorithm as an application specific integrated circuit (ASIC) was made due to: 1) the desire to implement the algorithm in real-time, and 2) the decreasing cost of design (in terms of both time and money) for custom ICs made possible by the availability of software tools for custom IC design.

The discussions in Chapter 2 will focus on the architectural concerns of implementing Jansson's method. The choice of computational structures used to compute the convolution as well as the arithmetic operations is explored. The actual implementation of the chosen architectures in silicon is the topic of Chapter 3. The role of computer-aided design (CAD) software is discussed. Block diagrams of the ICs components are used to describe the design and its hierarchy. Chapter 4 deals with the testing and evaluation of the IC design. Comparisons of IC performance with reported performance of past implementations are made. Finally, Chapter 5 looks ahead to future avenues of research by describing some possible changes to the current design which would be necessary to accommodate differing system configurations. The Appendix contains a short description of how the IC is to be operated along with computer files used to layout, and test, the IC. Simulation results and plots of the circuit layouts are included as well.

Chapter 2 - Design Architecture

System Level Architecture

The equations for Jansson's algorithm are given below as a reminder.

$$\mathbf{x}_e^{k+1} = \mathbf{x}_e^k + r\{\mathbf{x}_e^k\} [\mathbf{g} - \mathbf{h} * \mathbf{x}_e^k] \quad (2.1)$$

and

$$r\{\mathbf{x}_e^k\} = b[1 - 2/(c-a) | \mathbf{x}_e^k - (c+a)/2 |] \quad (2.2)$$

Since Jansson's method is an iterative procedure (i.e., estimates are dependent only upon past estimate values) the overall system architecture was chosen to be that of a linear array.

The array is shown, along with input and output data conversion devices and a computer controller, in Figure 2.1. For a system performing a given number of iterations, k , there would be k ICs in the array. Data enters the system at the left and is digitized by the analog-to-digital converter (ADC). The output of the ADC is input to the first IC which produces the first estimate, $\mathbf{x}_e^1$. This estimate is input to the second IC in the array which computes the second estimate, $\mathbf{x}_e^2$ and passes it to the next IC in line. This pipelined flow of data produces the result of the k th iteration at the output of the rightmost IC after a fixed latency. Once the pipeline has filled the IC array produces a new output each cycle of the sample clock. The output of the last IC in the array is converted back to analog form by the digital-to-analog converter (DAC) at the right. The inputs and outputs of the system occur at the system sample rate which is set by the maximum sample rate of the DSPs themselves.

The computer shown in Figure 2.1 is used to compute the relaxation function values to be stored in each of the DSPs. It is also used to program the ICs prior to

beginning a deconvolution. This programming involves loading the convolution coefficients (i.e., values of the impulse response function, $h(i)$), the relaxation function values and some scaling information needed to internally reformat data during the computations. This programming would be done simultaneously for all ICs in the array.

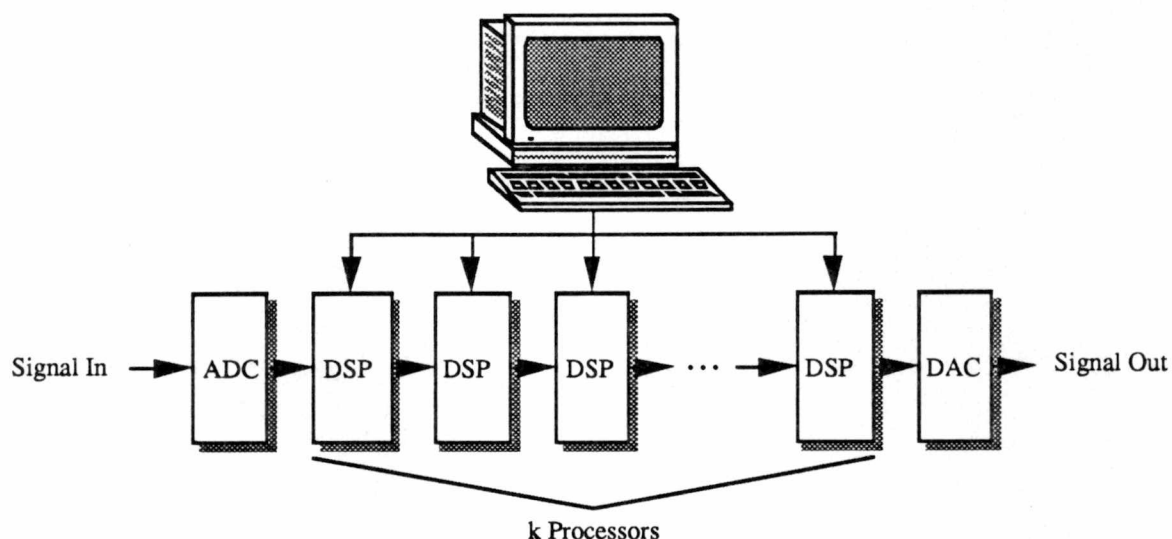


Figure 2.1 Linear array of DSP ICs making up a deconvolution system.

Chip Level Architecture

As stated earlier, Jansson's method involves the calculation of two quantities: the deconvolution estimate, $\mathbf{x}_e^{k+1}$ and a relaxation function, $\mathbf{r}\{\mathbf{x}_e^k\}$. The deconvolution estimate consists of a convolution operation, a subtraction, a multiplication (using the result of the relaxation computation) and an addition. Thus, the architecture of the deconvolution IC can be divided into two major subsystems: a convolution subsystem, and an arithmetic subsystem. Because the relaxation function requires both division and multiplication operations, it was decided early in the design that the relaxation function values would be calculated by the controlling computer. The computed values would then be stored in random access memory (RAM) for recall during the deconvolution process. Figure 2.2 shows the interconnection of the various components of the two subsystems.

Due to the potentially large number of processing ICs required to make up a deconvolution system, a major design consideration was die size. The objective was to build a complete iteration processor on a single substrate. To this end some tradeoffs between size and performance had to be made. To begin with, the design envisioned the use of bit level systolic array architectures[8-11] as the computational components. This type of architecture would have given very high performance but at an enormous price in substrate area. Therefore, it was decided that less costly (in terms of area) computational structures would be used. These circuit components trade substrate area for computation speed (i.e., the substrate area is decreased, but the time needed to compute iterations is increased).

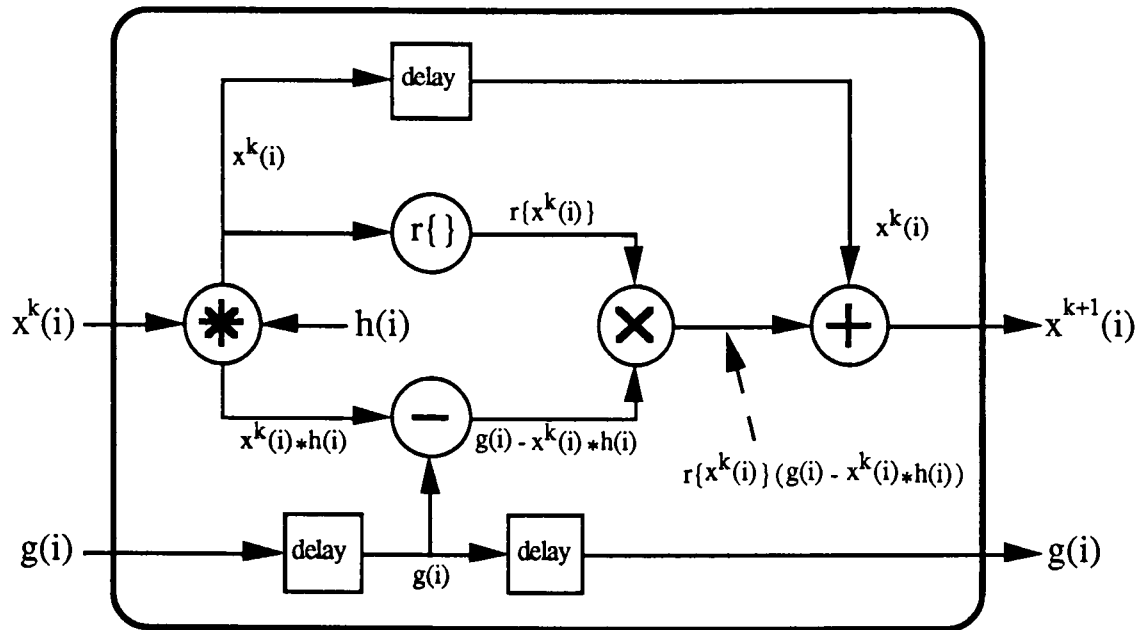


Figure 2.2 Floorplan of iterative deconvolution IC.

Convolver Subsystem

The convolution filter can be written mathematically as a linear difference equation, which expresses the filter output sequence, $y(t)$, at times $t=nT$ in terms of its previous inputs, $x(t)$, and a vector of coefficients, $\mathbf{h}$ as follows[12]:

$$y(t) = h_1 x(t) + h_2 x(t-T) + \dots + h_N x(t - NT + T) \quad (2.3)$$

where N is the number of points in the coefficient vector. An architecture for computing this equation is shown in Figure 2.3. For all but the shortest of convolution sequences this structure requires an large number of multipliers and adders to implement it as shown. To conserve substrate area the number of devices must be minimized. This can be done by recognizing that Eq.(2.3) contains a set of repeated operations (multiply and add). Such a formulation is called a recurrence. Each such repeated operation is termed an iteration within the recurrence. The ability to express a problem as a recurrence permits a tradeoff between circuit area and performance since the designer may choose how many processing units, called iteration processors (IP), are used to compute the recurrence[12].

The use of a single IP to compute a recurrence would result in a full serial architecture. Using a number of IPs less than the number of iterations, N , results in a multiplexed architecture. While using N IPs would result in a full array architecture. The bit-level systolic array is an example of a full array architecture.

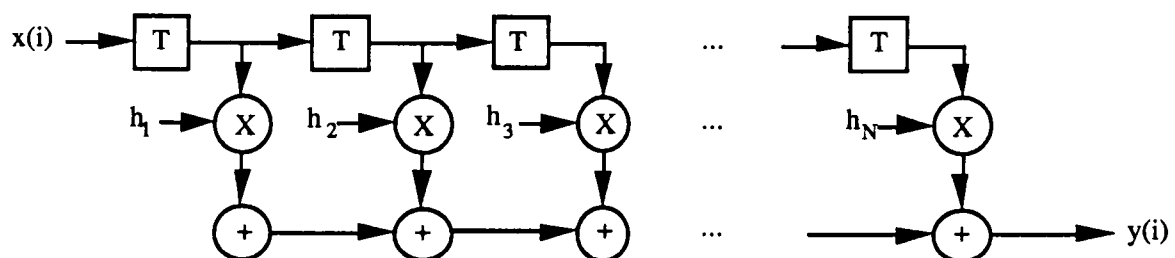


Figure 2.3 General architecture of an FIR filter (convolver/correlator)

Given the number of iterations in the recurrence, N , the rate at which an IP computes an iteration, W , and the input sampling rate, S , it is possible to determine, k , the number of IPs needed to compute the recurrence in real-time:

$$k = NS/W \quad (2.4)$$

The number of iterations, m , performed by each IP during a recurrence would be:

$$m = N/k \quad (2.5)$$

Figure 2.4 shows a multiplexed architecture where $k=2$. The IP and FIFO units in this architecture operate at the IP word rate, W , while the multiplexers for $x(t)$ and the accumulator input operate at the sample rate, S . The multiplexer for h would be activated to load the values of the coefficients, N of them, before the system is put into operation. A multiplexed convolver based upon this architecture[12] is used in this design.

Arithmetic Subsystem

The arithmetic subsystem is concerned with the computation of Jansson's algorithm using the values of the convolution, $(x_e^k * h)$, computed in the convolver subsystem. This evaluation consists of: 1) computing the difference, $g - (x_e^k * h)$, and finding the value of the relaxation function $r\{x_e^k\}$; 2) computing the product of the values found in step 1; and 3) summing the product from step 2 with the previous estimate x_e^k , to form the new estimate x_e^{k+1} .

This process can be implemented using a pipelined structure having three stages, each stage corresponding to the steps listed above. Figure 2.5 shows such a pipelined architecture. In this figure, the vertical dashed lines represent pipeline storage elements.

These latching elements would be clocked at the same rate as the output of the convolver subsystem, which is clocked at the sample rate, S .

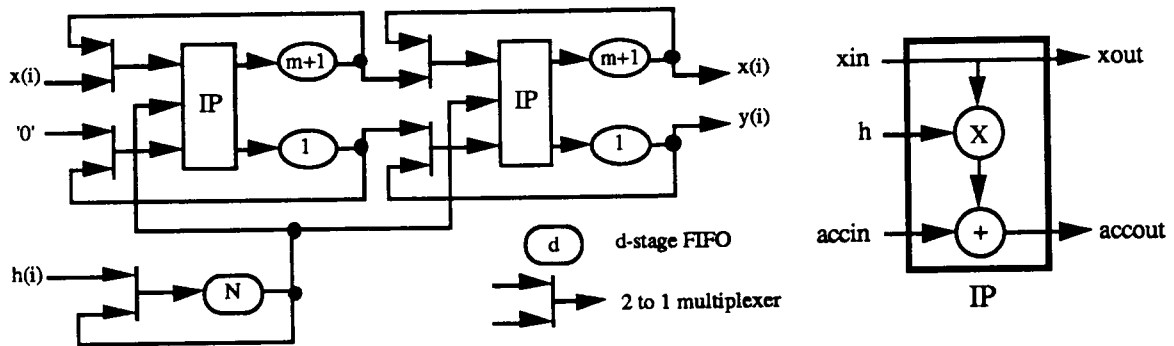


Figure 2.4 Multiplexed architecture of an FIR filter (convolver/correlator)

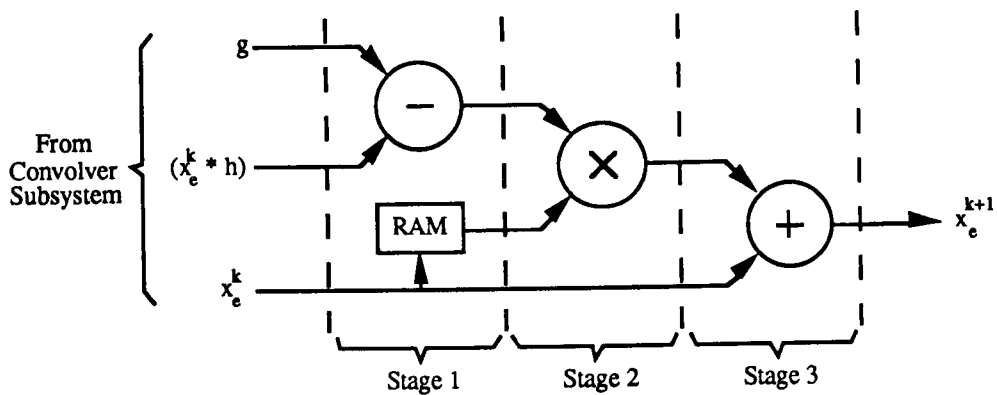


Figure 2.5 Pipelined architecture for the arithmetic subsystem.

Chapter 3 - Circuit Implementation

Target Architecture

The ultimate goal of any design is to provide the maximum processing power using a minimum amount of resources. One wishes to implement the given algorithm with a minimum number of devices and with a minimum number of interconnections. Thus, the convolver structure used in this design was chosen to be a pipelined, multiplexed forward-flow architecture as described in the previous chapter. This architecture yields a repetitive structure, and processor utilization is 100%. The pipelined architecture is continued into the arithmetic section thus allowing for 100% processor utilization in this section as well.

The first concern of circuit implementation is the determination of the system wordlength. For this project the system word length was chosen to be 8 bits. One of the actions of an FIR filter is to amplify signal components which correlate with the reference vector. This is done by accumulation, where the addition of many small signal components may become significant. For this reason it is important that the low order bits of the convolution product be used during the accumulation process. Secondly, the word growth of the accumulator value must be taken into consideration. Word-growth results when many values of a fixed bit width l , are added together and the value of their sum requires more than l bits. For this design, with the system wordlength equal to 8 bits and the number of recurrences to be calculated equal to 32, the convolution accumulator must be 20 bits wide to allow for worst case word growth. The 20-bit output of the convolver must be converted back to an 8-bit value so that the system word-length is not exceeded. This will be done by a reformatting circuit at the end of the convolver array.

Provisions must be made for programming the coefficient vector and the relaxation function values as well. Programming of the coefficients should be initiated by an external request. The programming data are input to the IC via a dedicated port and is controlled by

a finite state machine (FSM) which generates appropriate control signals. The relaxation function values are to be stored in off-chip RAM. Each DSP IC will have its own look-up-table (LUT) RAM which is programmed separately from the DSP ICs. The system control computer should supply the programming data to each IC and its associated RAM.

The following sections discuss the implementation of the target architecture. The circuit implementations described in this chapter were all created using the LAGER IV silicon assembly system, developed by the University of California, Berkeley[13,14]. This system is a set of software tools which allows the IC designer to describe the desired circuit in a high-level design language called SDL. The circuit layout is then generated by the software following this program description. The LAGER system is capable of generating layouts using Standard Cells, Datapath cells, TimLager cells or a combination of all of these. This IC design uses all of these layout styles to implement the deconvolution algorithm. The SDL files for this design may be found in the Appendix.

Because of the overall complexity of the deconvolution IC, its design had to be broken into smaller, more manageable parts in order to describe it using the LAGER tools. This partitioning leads to a hierarchical design in which higher level (more complex) circuits are built using lower level (less complex) circuits. The lowest level would correspond to the various circuits found in the LAGER cell libraries. In the following sections some attempt has been made to formalize this hierarchy when circuit diagrams are given. In each diagram, circuit interconnections are shown between blocks making up the circuitry. The square blocks in the diagrams are used to represent cells which come from the LAGER libraries and hence are at the lowest level of the hierarchy. The rounded blocks in the diagrams represent circuit elements which are at a lower level of the hierarchy than the circuit being described in that particular diagram. This formalism is intended to give the reader some feel for the design process required when dealing with a complex implementation.

Convolver Subsystem

Iteration Processor

The convolver subsystem is made up of a cascaded array of iteration processors as described in the preceding chapter. Thus, the implementation of the complete subsystem is reduced to the problem of creating a single, cascable IP and then connecting the required number of these together to realize the convolution function. Figure 3.1 shows the components of the IP at its highest level of hierarchy. Each IP consists of a multiply / accumulate (MAC) operator, surrounded by state memory for the inputs (HSTATE, XSTATE). Clock buffering (CB, HCLK, XCLK, ACLK) is also provided to decrease the demand on the global clock distribution networks.

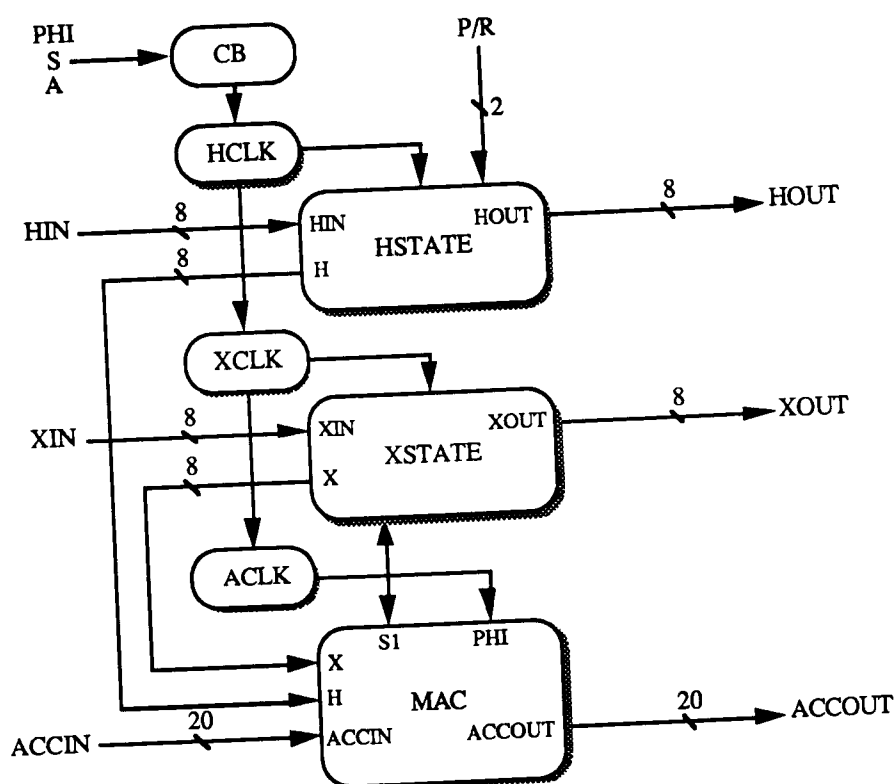


Figure 3.1 Components of a cascable multiplexed IP.

The IP accepts signal inputs on the left-hand side of the diagram on the rising edge of the S1 clock. The IP then computes the convolution products and accumulates them for m recurrences. The IC uses three sets of two-phase, non-overlapping clocks: PHI1, PHI2; S1, S2; and A1, A2 which are generated from the three clock signals PHICLK, SCLK and ACLK shown in Figure 3.2. IP processing occurs at the processing rate governed by the PHI1 clock. The S1 clock is $(1/m)$ th the frequency of the PHI1 clock and has a duty cycle of $1/m$. The signal value and the accumulated result are then output on the right-hand side of the IP to be clocked into the next IP in the convolver array on the rising edge of the S1 clock.

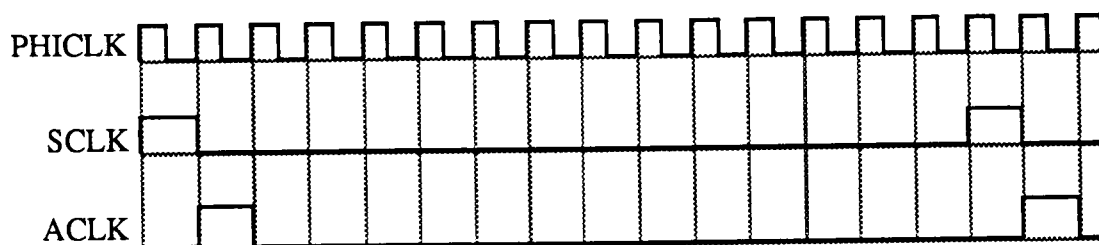


Figure 3.2 Relationship between clock signals used by IC.

Multiply / Accumulate Circuit

The multiply / accumulate (MAC) circuit is the heart of the convolution IP. A block diagram of the MAC is shown in Figure 3.3. The MULT circuit is a TimLager macro cell of a 2's complement multiplier array. The multiplier accepts two 8-bit, 2's complement values X and H and produces a 15-bit, 2's complement product. This product is sign-extended to 20 bits before being input to the accumulator, ACC. The sign-extension is achieved by tying the five most significant bits of the accumulator input to the sign bit of the product.

The accumulator is implemented as a 20-bit wide datapath circuit. The accumulator has two inputs: the convolution product from the multiplier and the accumulator cascading

input. These inputs are latched on entering by two pipeline registers (PL and ML). These registers maximize the throughput of the MAC by cutting the critical path into two operations: 1) multiplication and 2) accumulation. The accumulator datapath is shown in Figure 3.4.

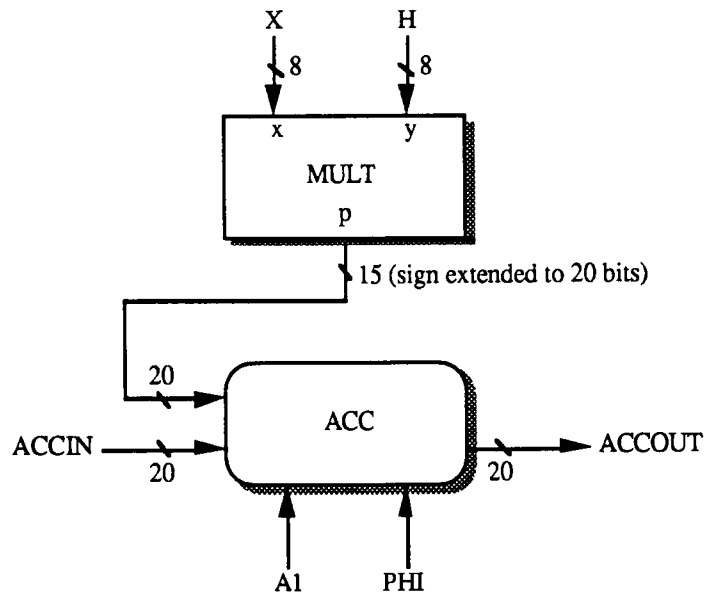


Figure 3.3 The MAC circuit.

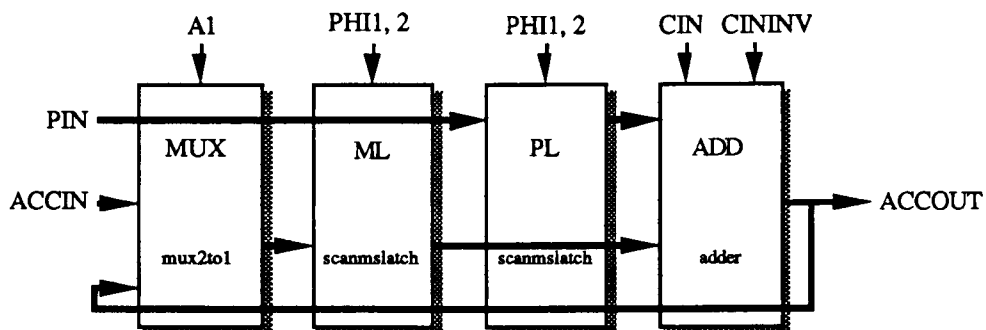


Figure 3.4 The ACC datapath.

State Memory

Each IP accepts an input sample and computes m recurrences of the convolution operation and then passes the sample value to the next IP. Since each IP must compute more than one cycle of the recurrence some amount of memory must be provided to store the 'state' of the IP. This memory takes the form of a first-in, first-out (FIFO) register having its output port tied to its input port via a multiplexer. Each IP contains two such state memories, one for the reference vector (HSTATE) and one for the signal input vector (XSTATE).

The state memory for the signal input must have $m+1$ memory locations to store the state of the input. The $m+1$ memory locations are necessary to allow the state memory to retain previously input values while at the same time accepting new input values. Every sample period the XSTATE datapath latches a new input value (call it X) into the FIFO via a multiplexer driven by the $S1$ clock. The contents of the first FIFO location is output to the multiplier array. The FIFO shifts X one location to the right every cycle of the $\Phi I1, 2$ clock. After $m+1$ cycles, X is presented to the input of the next IP in the convolver array, and is fed back to the input of the FIFO via the second multiplexer input.. Figure 3.5 shows a block diagram of the XSTATE datapath.

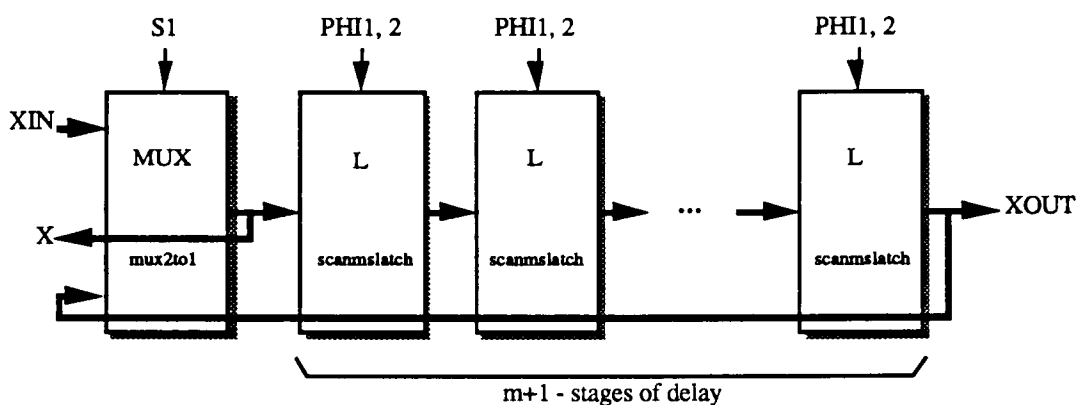


Figure 3.5 The XSTATE datapath.

The HSTATE vector memory is very similar to the XSTATE datapath. HSTATE contains m memory locations. The multiplexer in this datapath is controlled by the P/R control from the program control portion of the IC. When P/R is asserted the FIFO accepts new inputs. When P/R is inactive the output of the FIFO is connected to the input of the FIFO via the multiplexer. This feature is used to program the reference vector values prior to system operation. HSTATE also contains a latch for propagating the XIN input value through the convolver so that the timing relationship between the signal estimate and the convolver output is maintained. These values must be time aligned for the arithmetic processor to produce the correct results. Figure 3.6 shows the HSTATE datapath.

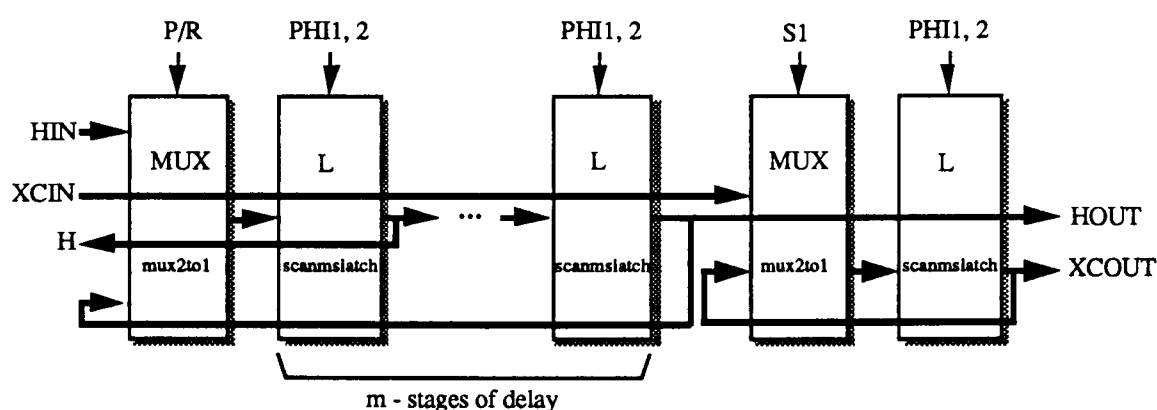


Figure 3.6 The HSTATE datapath.

Convolver Array

Now that an iteration processor has been designed a convolver array can be built by cascading two or more of them together. As mentioned earlier a reformatting circuit must be added to the output so that the accumulator value can be converted to an 8-bit value for input to the arithmetic section of the IC. The formatting circuit allows the user to select a scale factor for multiplying the impulse response vector. This flexibility is provided so that the resolution of the digitized h-vector values can be maximized. The convolver array is shown in Figure 3.7. Note that the input to the accumulator of the first IP is connected to ground, thus the accumulator adds the convolution products of the first IP to zero.

The formatting circuit is shown in Figure 3.8. It consists of a 20-bit wide latch followed by four 20-bit wide multiplexers. The output of the accumulator is latched on the falling edge of the S2 clock and is connected to the first input of the first multiplexer (MUX1). This same output, shifted down 1 bit, is connected to the second input of MUX1. The output of MUX1 is in turn connected to the first input of MUX2. The same input, shifted down 1 bit, is input to the second input of MUX2. The remaining multiplexers are interconnected in the same way. Each multiplexer is controlled by two control lines: SEL1 and SEL2. Asserting the SEL1-control line causes the multiplexer to pass the value of its first input, while asserting the SEL2-control line causes the shifted input value to be output to the next multiplexer in line. The HSCALE register in Figure 3.8 stores the h-scale factor and its output is connected to the four sets of multiplexer controls.

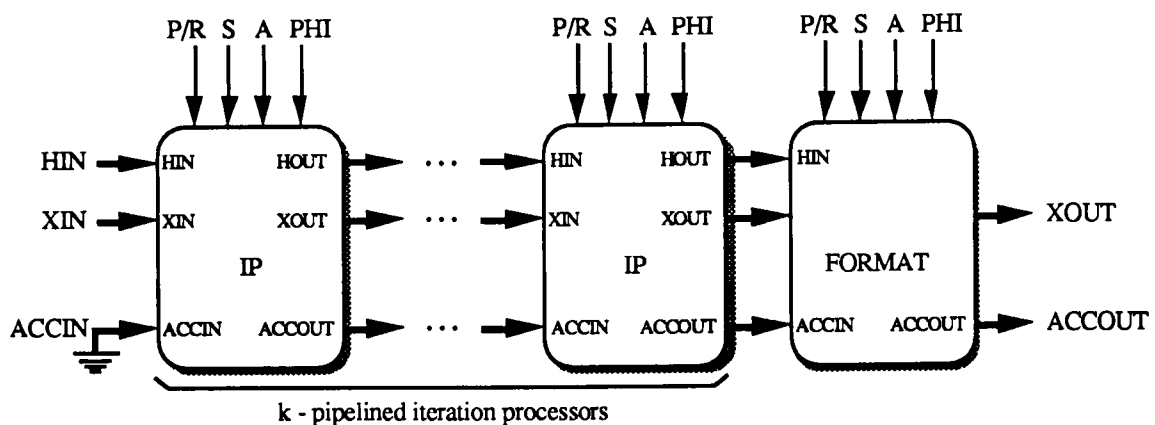


Figure 3.7 Array of iteration processors and formatting block.

Arithmetic Subsystem

Although the convolver structures discussed above constitute a major portion of the design, the task of computing the iterative algorithm is carried out by the arithmetic subsystem. In keeping with the overall design philosophy of pipelined operation, the arithmetic portion of this design is a datapath design having three major stages with pipeline registers between them. These stages are:

- Subtraction and relaxation function evaluation

- Multiplication
- Additive estimate correction

Pipeline Implementation

The first and third stages of the pipeline are integrated into a single 8-bit wide datapath. This datapath includes everything needed to compute the algorithm except for the multiplier which makes up the second stage, the LUT memory, and the **g**-vector delay line. These components are connected to the datapath to form the complete circuit which is shown in Figure 3.9.

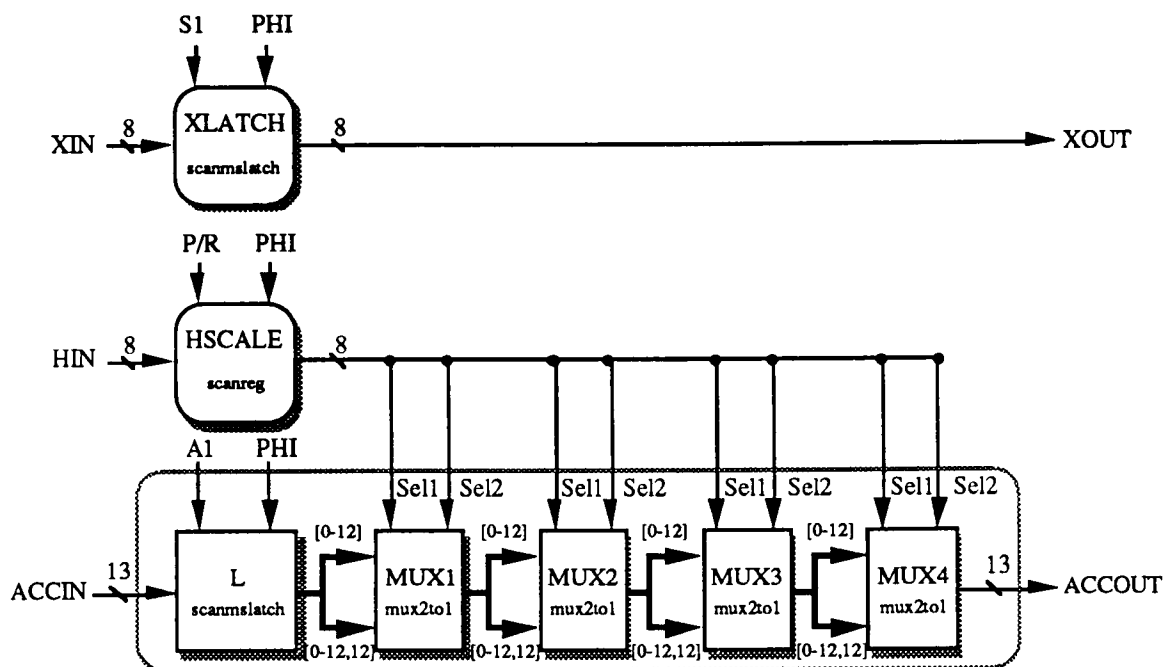


Figure 3.8 The FORMAT circuit.

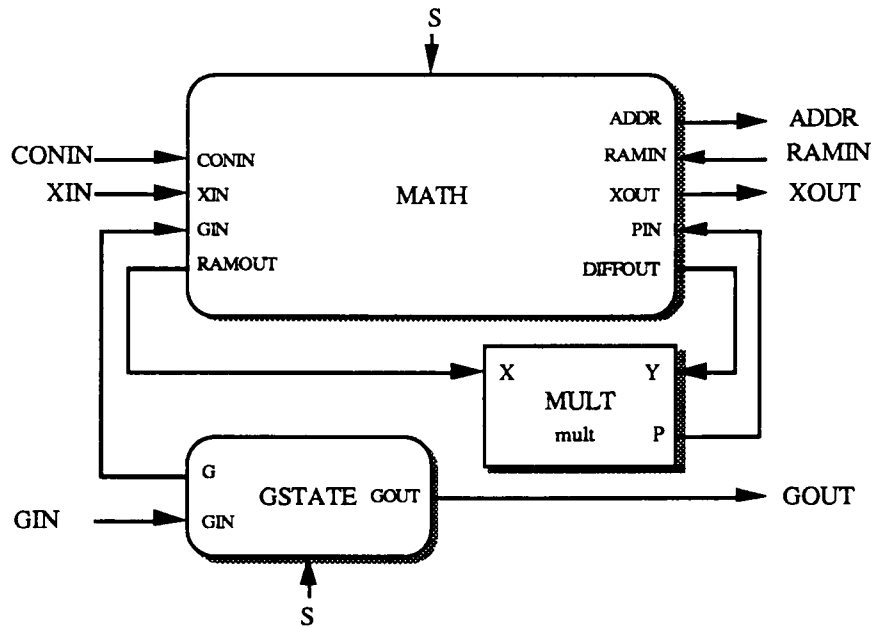


Figure 3.9 Arithmetic Pipeline (A_PIPE).

The first stage of the arithmetic pipeline is responsible for computing the difference between the original signal estimate, g and the result of the convolution operation, $(x_e^k * h)$. The input labeled CONIN is the formatted output of the convolver structure. The input G is the original signal estimate value, g . This value comes from an 8-bit delay line which is used to maintain correct time alignment between g , x_e^k , and the convolver output. This delay line is shown in Figure 3.10. The delay elements before the G output connection compensate for signal propagation through the convolver subsystem, while those delay elements following this connection compensate for signal propagation through the arithmetic pipeline. This delay line is important because each IC in the deconvolution system must use the original signal estimate to compute the new signal estimate from the previous estimate.

At the same time as the above operations are being performed the value of the relaxation function is determined. The XIN input is the previous signal estimate value

being output from the convolver structure. This value is used as the address of the relaxation function value to be fetched from the off-chip RAM. The output of the RAM enters the datapath via the RAMIN bus and is latched by the RL latch.

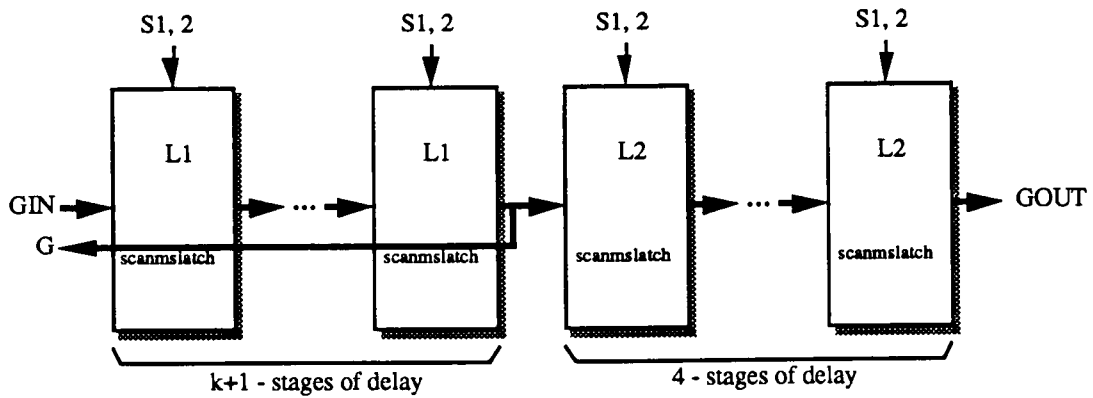


Figure 3.10 8-bit delay line for the original signal estimate (GSTATE)

The second stage of the arithmetic pipeline is the multiplier. This multiplier is identical to the one used in the MAC circuit found in the iteration processors of the convolver structure. The 8-bit values DIFFOUT and RAMOUT are multiplied, producing a 15-bit product, 8 bits of which enter the datapath via the PIN bus.

The final stage of the arithmetic pipeline computes the new signal estimate, x_e^{k+1} , by adding the result of the multiplication to the previous signal estimate, x_e^k . This stage consists of an input latch for the multiplication product, a series of latches, which compose the pipeline registers for the signal estimate value x_e^k , and an adder circuit followed by an output latch. The block diagram for the integrated datapath is shown in Figure 3.11

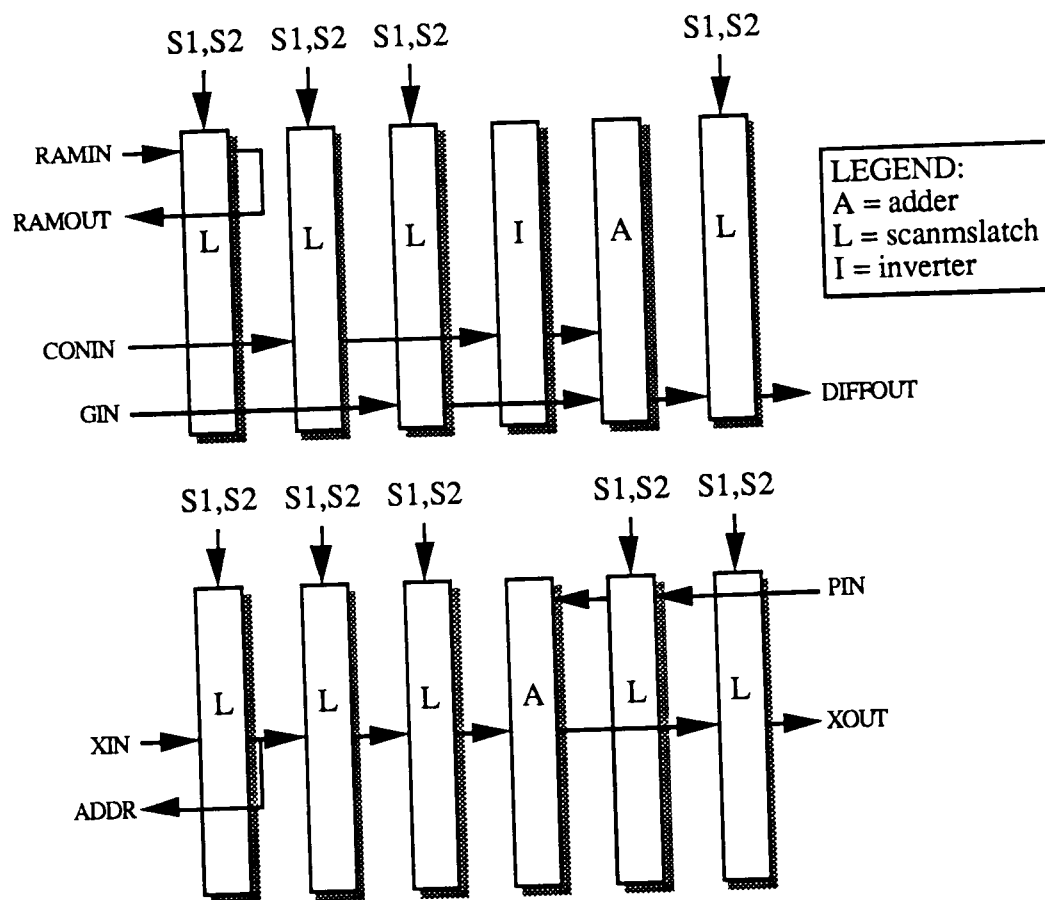


Figure 3.11 Stages 1 and 3 of the arithmetic pipeline as an integrated datapath.

Programming Control

Before the deconvolution IC can be used to restore signals it must be programmed with the values of the impulse response function h , the scale factor for the h -function, and values for the relaxation function, $r\{x_e^k\}$. The programming is accomplished by sequentially shifting programming values into the programming port. These values pass through a chain of memory locations forming a shift path through the programmable portions of the IC. This shift path is shown in Figure 3.12

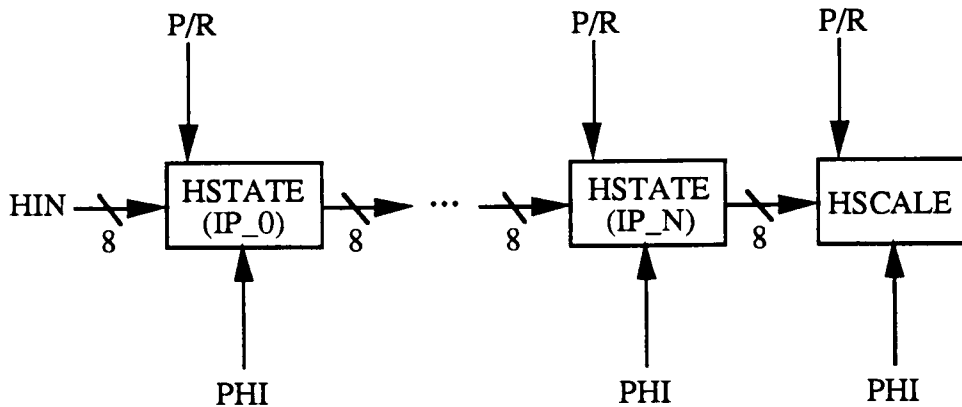


Figure 3.12 Programming shift register path of deconvolution IC.

As seen in Figure 3.12, program data is shifted through the **h**-vector state memories of each IP in the convolver array. The data exiting the last HSTATE memory enters the HSCALE register contained in the output formatting circuitry of the convolver structure. During normal IC operation the shift path is broken by bringing the P/R control line to a low state.

In order to assure correct programming of the IC, a finite-state-machine (FSM) is used to control the programming cycle. When given the start programming signal the FSM performs the following tasks:

- signals the user that programming is in progress
- establishes the program shift path
- counts the number of input data values

After the correct number of data values have been entered the FSM restores the shift path to its normal operating mode and informs the user that programming is complete. In addition to these duties, the program controller in this design also places the IC into a non-operating mode by disabling the system clocks. The disabling of the clocks prevents the **h**-vector state memories, contained in the convolver structure, from assuming an unknown state before the system begins to receive input signals. Normal operation is restored by momentarily pulling the RST_B pin low, thus resetting the IC. Figure 3.13 shows the

state diagram of the program control FSM. The complete program control circuit is shown in Figure 3.14. This figure shows the interconnection of the various elements required to implement the operations discussed above.

Chip Integration

The next step in the design is the connection of the various subsystems together to form the complete implementation of the deconvolution algorithm. This is done by writing an SDL description of the interconnections between the major blocks described above. The block diagram of the integrated system is shown in Figure 3.15. To complete the IC design one must place the core into a frame of I/O pads. This is accomplished by writing an SDL description of the connections between the circuit and the pad terminals.

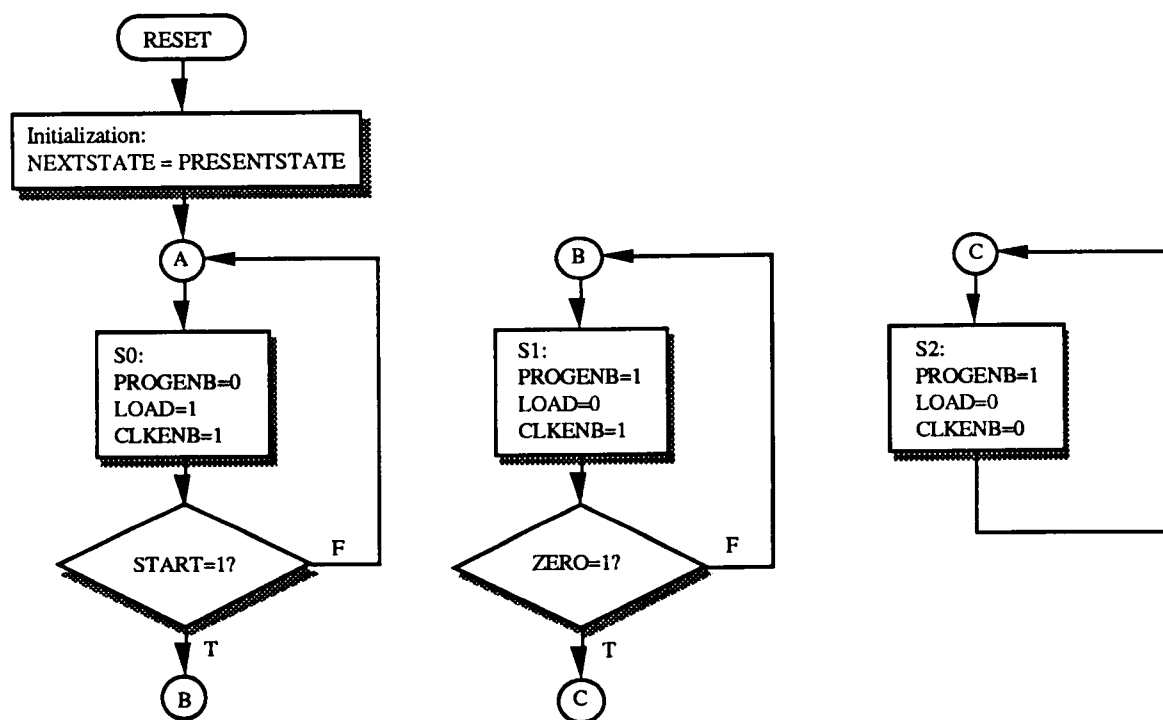


Figure 3.13 State diagram of the program control finite state machine.

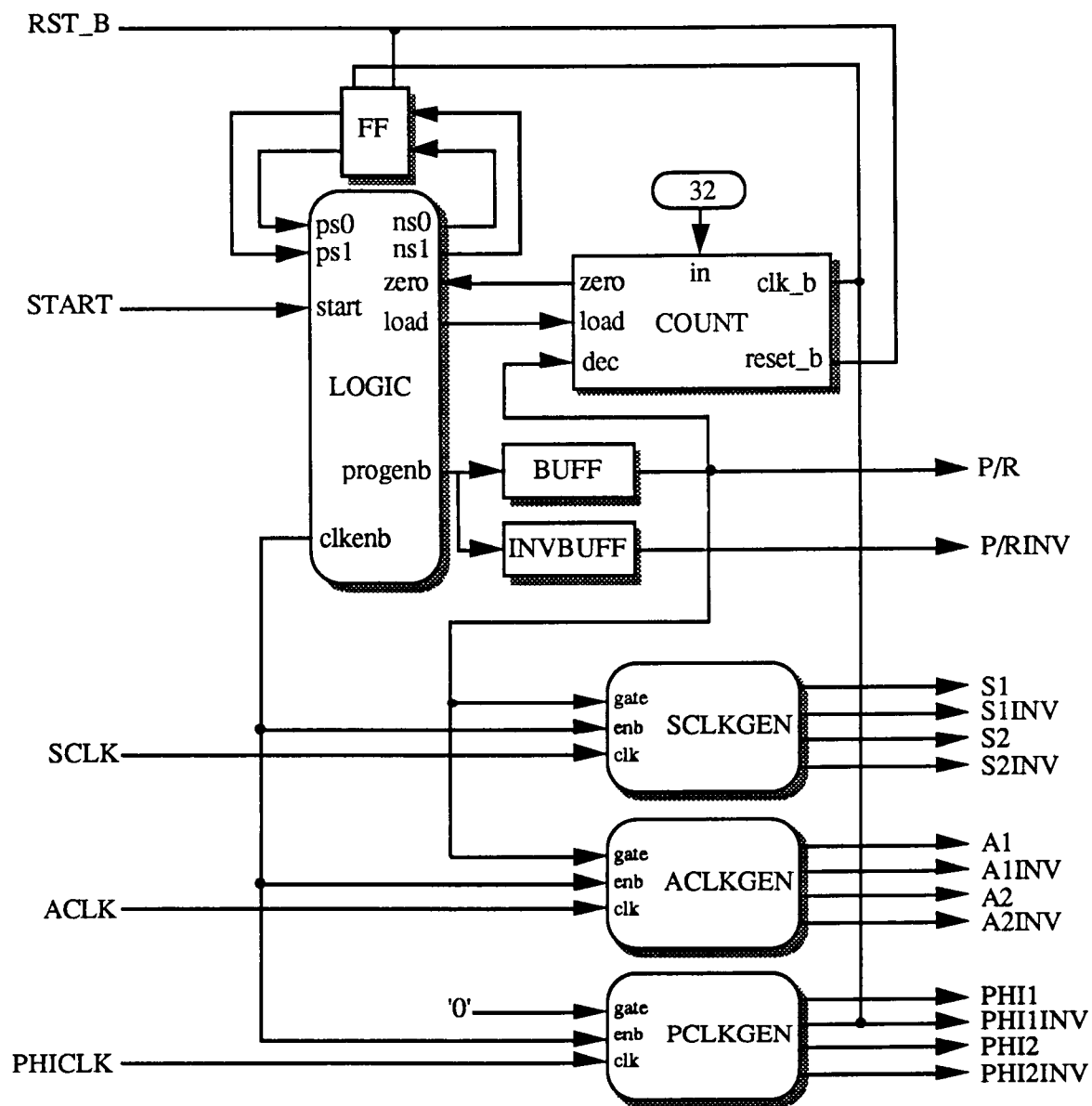


Figure 3.14 Program control logic (CONTROL).

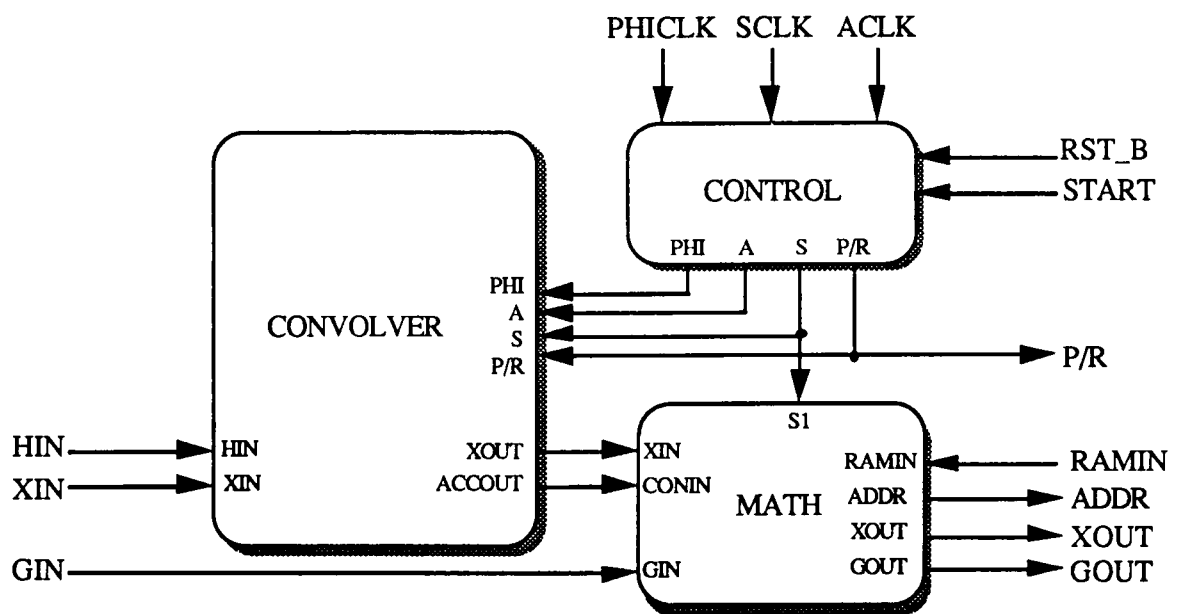


Figure 3.15 Block diagram of the IC core (CORE).

Chapter 4 - Testing and Design Evaluation

Test Design Considerations

The descriptions of circuit implementations in the previous chapter do not completely describe the final design layout in that they do not include some of the values for important design parameters such as: k , the number of IPs in the convolver array, or m , the number of iterations computed by each IP. Before test patterns can be created these parameters had to be given a permanent value. The following lists the design parameter and its final value as used in the final layout of this IC:

• System wordlength, WL	8
• Number of convolution coefficients, N	32
• Number of iteration processors, k	2
• Number of iterations computed by IP, m	16

These parameter values have a direct effect on the timing of the IC. The multiplexed convolver structure determines the relationship between the PHI clocks used to operate the convolver memories, and the S and A clocks which control the pipelined flow of signals through the IC.

The task of verifying the correct operation of each of the pieces of the design discussed in the previous chapter was carried out using a logic-level circuit simulator called IRSIM. IRSIM uses information from the circuit layout database which has been extracted by another part of the LAGER design system called DMpost. This extracted data contains capacitance values as well as interconnect information for the nodes of the circuit. IRSIM processes the simulation files produced by DMpost and displays the circuits behavior as a waveform, much like that produced by a logic analyzer.

IRSIM is able to simulate the circuits relatively quickly, compared to more complex simulation programs such as SPICE. Questions often arise concerning the timing behavior predicted by logic level simulators and the predictions made by the SPICE simulator. In past designs the circuit performance predicted by IRSIM has proven to be more pessimistic than that predicted by SPICE (i.e., SPICE predicts that circuitry runs faster than what IRSIM predicts). For this reason, no SPICE simulations were made of the circuitry reported in this thesis.

The following sections give test results without referring to the actual test output waveforms. The reader may find the test output waveforms in the Appendix, along with the command files used to simulate the various pieces of the IC.

Test Results

Convolver Subsystem

The overall performance of the IC is determined by the performance of the convolution subsystem. The throughput of the convolver is effected by the number of IPs used in its construction (i.e., the more IPs used, the greater the convolver throughput). For a given number of IPs, the convolvers performance depends upon the performance of a single IP. The speed of the multiply / accumulate function of the IP determines the speed of the IP as a whole.

Testing verified the operation of the IP at a PHI clock frequency of 8.333 MHz. This was achieved only after extensive clock buffering for the vector state memories, HSTATE and XSTATE. As discussed in Chapter 2, once the computation rate of the IP is known, the sample rate of the convolver can be determined. The convolver sample rate is found to be 520.625 KHz for the design using the parameters given above. Increased sample rates may be obtained by increasing the number of IPs in the convolver or by improving the speed of the MAC circuit.

Arithmetic Subsystem

The arithmetic subsystem is a pipelined datapath clocked at the input sample rate set by the S clocks. Proper operation occurs as long as the circuits within each stage of the pipeline produce the correct results at a 520.625 KHz rate. All of the circuits were verified by IRSIM to operate at this frequency. This was achieved without the necessity of the extensive buffering found in the convolver subsystem. If in the future the IC were redesigned to accommodate a higher sample rate, then some form of buffering may be required to guarantee proper operation.

Design Evaluation

As mentioned earlier, Jansson's algorithm has been implemented on general purpose computers in the past. When comparing the performance of these various implementations one must take into consideration many factors. For a performance comparison to be really meaningful, all of the various implementations should be tested using the same set of input vectors, use the same number of coefficients, perform the same number of iterations, and use the same type of arithmetic operations (i.e., floating-point vs. fixed-point). While this is the ideal situation for comparison it is not possible in this case. In particular, the number of coefficients in the convolution and the type of arithmetic used differ between the software implementations and the deconvolution IC. In the following discussion, the pertinent implementation information is given so that the reader is fully aware of the differences in the deconvolution systems.

Performance figures found in the literature have been produced by software implementations of Jansson's method. The computer systems used took advantage of math coprocessors and performed the mathematics using a floating-point format. Each of the implementations used an 80 point convolution coefficient vector and performed 200 iterations of the algorithm. The sample rates of these implementations were as follows:

- IBM PC/AT, 8 MHz, 200 iterations 3 sec/sample
- IBM PS/2 Model 70 PC, 25MHz, 200 iterations 0.42 sec/sample

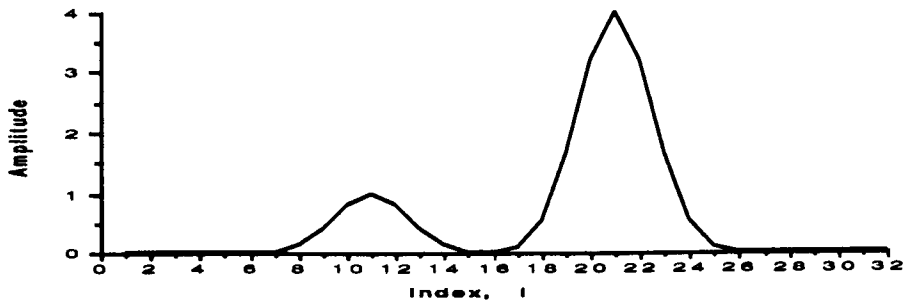
The IC design described in this thesis differs in two main respects from the software implementations described above: 1) the IC uses a 32 point convolution coefficient vector; and 2) the mathematics are performed using a fixed point format (8-bit values). The IC demonstrates the following sample rate:

- Deconvolver IC, 8.33MHz 1.92 μ sec/sample

To compare this figure directly to those given previously would not be proper without taking into consideration the effects of the differing system aspects mentioned above. While the number of coefficients in the convolution used by the IC could be changed to 80, this would require additional IPs in the design (i.e., the design presently contains 2 IPs for a 32 point convolver, 5 IPs could be used to give an 80 point convolver). The addition of the extra IPs would not change the effective sample rate, but would increase the latency of the IC, and thus the latency of the overall system. The number of iterations performed would not change the sample rate either, but would increase the overall system latency.

The use of fixed-point mathematics in the IC must also be considered when a comparison of performance is being made. The deconvolver IC accepts 8-bit wide input values and produces 8-bit wide outputs values. The software implementations use floating-point values and produce floating-point values. Thus, one must take into account the quality of the results produced by both approaches. In order to do this a computer program was written to make this comparison. The program uses three sets of data points, produced mathematically, for input vectors. The first set is the ideal signal vector which is used to determine the amount of error present in a given estimate produced by the algorithm. The second set is the impulse response vector which is used as the coefficient

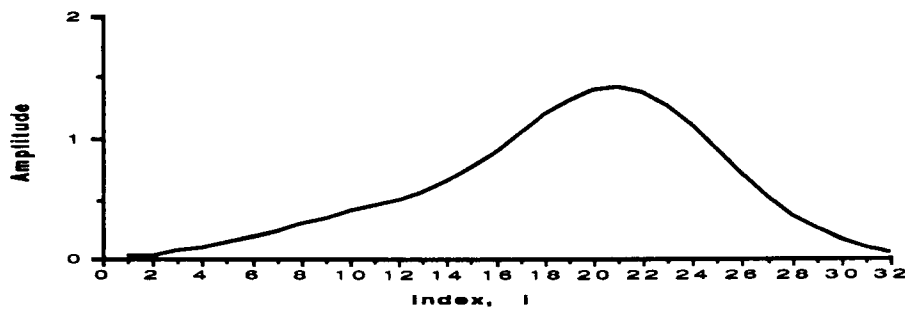
vector of the convolver. The final set is the observed signal vector, obtained by mathematically convolving the ideal signal vector with the impulse response vector. These waveforms are shown in Figure 4.1.



(a) Ideal signal vector



(b) Impulse response vector



(c) Observed signal vector

Figure 4.1 Data sets used to compare mathematical formats.

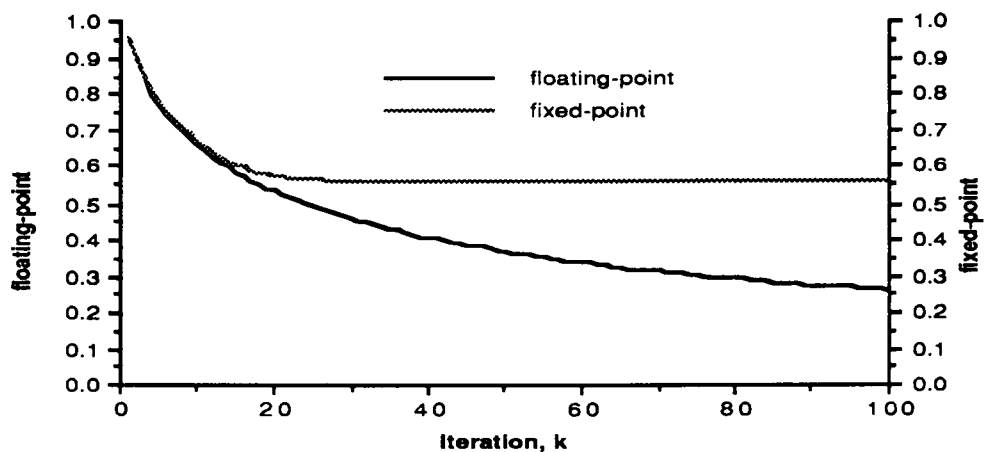
The data sets were used as shown in Figure 4.1 to compute 100 iterations of Jansson's method using a floating-point format. To simulate the ICs fixed-point format, the values in the data sets were 'quantized' as follows. The maximum value of the ideal signal vector, V_{max} was used to determine the range over which the waveforms would be quantized. This range was then divided by $2^8 = 256$ to determine the value of the least significant bit (LSB) of the quantized values. Each point in each data set was then divided by the LSB value and the result rounded to the nearest integer value to simulate the action of an analog-to-digital converter. Finally, 100 iterations of Jansson's method were computed using fixed-point arithmetic.

The error between ideal signal vector, $x(i)$ and the signal estimate vector $x_e^k(i)$ was computed after each iteration using the following formula:

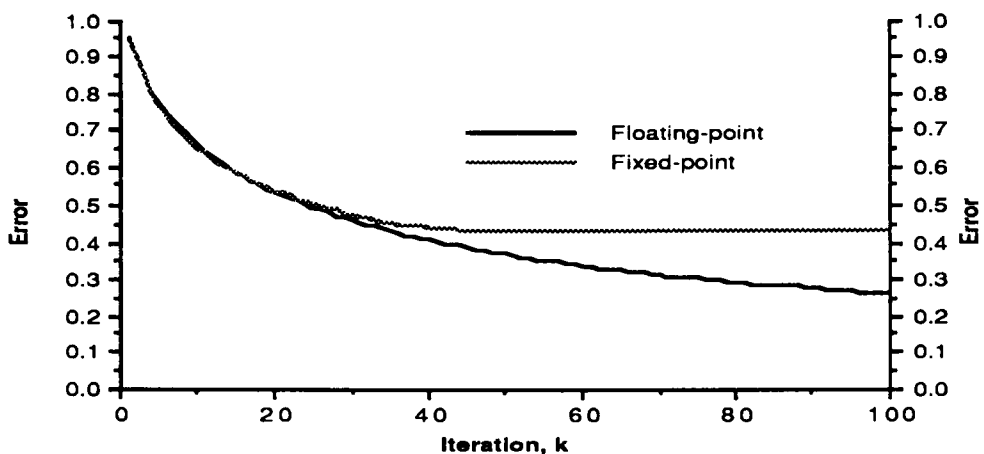
$$\text{Error} = \sqrt{(\sum_i (x(i) - x_e^k(i))^2)} \quad (4.1)$$

Plotting the error value against the iteration number gives a measure of how Jansson's algorithm converges toward the ideal signal vector.

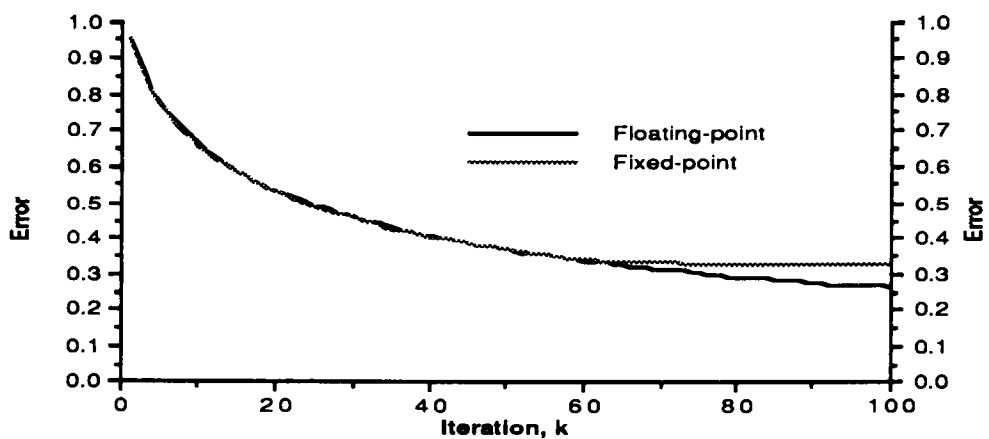
The error vs. iteration plots for both the floating-point and fixed-point formats are shown in Figure 4.2. Figure 4.2a shows the error curve resulting from 8-bit fixed-point arithmetic, which is what is used in the IC described by this thesis. The error curves in figure 4.2b,c correspond to 9-bit and 10-bit fixed-point arithmetic respectively. The error at higher iterations numbers is reduced due to the increased resolution afforded by the larger system wordlengths. Thus, in order to get an estimate with the same quality as that produced by floating-point implementations, one must increase the system wordlength.



(a) 8-bit fixed-point arithmetic



(b) 9-bit fixed-point arithmetic



(c) 10-bit fixed-point arithmetic

Figure 4.2 Error vs. iteration plots for floating-point and fixed-point formats.

Chapter 5 - Summary, Conclusions and Future Work

Summary

This thesis has reported on the development of a digital signal processor for the purpose of increasing the throughput of Jansson's iterative deconvolution algorithm. In the process of this development, Jansson's algorithm was chosen to be implemented in ASIC form due to its superior overall performance, compared to other iterative deconvolution methods found in the literature.

Once the algorithm was defined, a mapping of the algorithm into a design architecture was performed. This architecture took the form of a pipelined processor composed of two main parts: a convolution structure and an arithmetic processor. The convolver architecture took the form of a time-multiplexed, forward flow, pipelined array, while the arithmetic processing structure took the form of a simple pipelined processor.

The actual IC layout was synthesized using a high-level design system called LAGER. After describing the desired circuit components in LAGER's descriptive language SDL, automatic layout was performed by the software. The use of this system for layout allows future modifications to the design to be made with a minimum of redesign effort.

The resulting circuit was tested and found to be functional at a clock rate of 8.333 MHz, which corresponds to an input sample rate of 520.625 KHz. The IC core contained approximately 21,000 transistors, occupying a substrate area of 55 mm² at a feature size of 2μm.

Conclusions

Use of this IC for future deconvolution applications will benefit from a significant increase in algorithm throughput. The proposed architecture is general enough that the deconvolution system can be tailored to the user's needs with a minimum of effort. These points, coupled with the excellent restoration properties of Jansson's method, should prompt more researchers to take advantage of numerical deconvolution.

Future Work

System Modifications

The architecture used for this IC design can be configured in many ways, depending upon the needs of the user. The three most important system parameters are: 1) the number of convolution coefficients; 2) the wordlength of the system; and 3) the number of iterations to be performed. The following discussions focus on the changes that must be made to the design files to set these parameters to a desired value

To change the number of points in the convolver coefficient vector, the designer must refer to the original design equations. Given the number of coefficients in the convolution, N , the rate at which an IP computes an iteration, W , and the input sampling rate, S , it is possible to determine, k , the number of IPs needed to compute the recurrence in real-time:

$$k = NS/W \quad (5.1)$$

The number of iterations, m , performed by each IP during a recurrence would be:

$$m = N/k \quad (5.2)$$

To change the number of coefficients used in the design files, three areas must be changed: 1) the state memories, HSTATE and XSTATE; 2) the clock tree structures HCLK and XCLK; and 3) the accumulator, ACC. The length of the state memory HSTATE is equal to m , to change this the *delay* parameter, in HSTATE.sdl, must be set equal to m . The length of the state memory XSTATE is equal to $m+1$, to change this the *delay* parameter, in XSTATE.sdl, must be set equal to $m+1$. Next the number of buffers in the clock trees which service the state memories must be changed, since these structures are not parameterized, this change may be made in differing ways. The current clock trees have three levels of buffering. The first level consists of a single buffer for each clock signal, the output of each of these drives three buffers in the next level. The second level buffers each drive three buffers in the third, and final, level. The outputs of the third level buffers each drive two clock lines of the 8-bit wide state memory datapath. It is up to the designer to provide sufficient buffering on the clock lines. The third area which must be changed is the width of the accumulator datapath. For a given system wordlength, the width of the accumulator must increase if the number of coefficients increases. This is done to allow for worst-case word-growth during the accumulation process. The formula used to determine this width is:

$$\text{accumulator width} = \log_2 [N(2(\text{system wordlength})-1)] \quad (5.3)$$

To change the number of bits in the system wordlength several areas must be changed: 1) the width of the state memories, HSTATE and XSTATE; 2) the width of the multiplier in the MAC circuit; 3) the width of the accumulator, ACC; 4) the width of the latches and multiplexers in the FORMAT circuit; 5) the width of the GSTATE delay-line; 6) the width of the arithmetic pipeline, MATH and 7) the size of the multiplier found in the A_PIPE circuit. Changes to the width of datapath circuits (i.e., HSTATE, XSTATE, ACC, MATH, GSTATE, FORMAT) can be made by changing the N parameter in the respective SDL files. To change the size of the multipliers, change the m , and n parameters.

Changing these will require changes to the s and $csindex$ parameters as well. The s parameter is the number of carry groups in the carry-select adder which produces the product value, while the $csindex$ parameter is a vector which represents the number of carries to include in each group. The value of s must be 2 or greater. The number of values in the $csindex$ vector must be equal to s and their sum must equal $m+n-3$.

To change the number of iterations performed by the deconvolution system, one simply needs to change the number of ICs used to form the array shown in Figure 2.1. The throughput of the system is constant for an arbitrary number of iterations, but since the array is pipelined, there is a latency before the output begins. Once the pipeline is filled, however, the array produces a new output each sample time. The only effect caused by the changing of the number of iterations is an increase or decrease in the latency before the initial output is produced.

Performance Enhancements

Some possible performance enhancements for future applications have been identified. The most useful of these would be the inclusion of hardware on the IC which would allow dynamic reconfiguration of the overall system. Hardware could be included which would allow the number of convolution coefficients to be changed by bypassing the formatting circuits and arithmetic processors to chain the outputs of the convolver array with the inputs of the next convolver array to form a larger array.

Increased IP performance might be realized by using a different multiplier structure than the one used in this design. In particular, the 'bmult_csa' macro-cell in the TimLager library could be used. This multiplier operates up to 32 MHz, but has a couple of problems: 1) the size of the multiplier is limited to 8x8 or 16x16; and 2) it is a pipelined structure. This second problems would require redesign of the MAC circuit to accommodate the latency of the multiplier.

Another possible performance enhancement to the deconvolution system proposed here would be to fabricate the ICs at a smaller feature size. The timing values given here are for 2 μ m feature sizes. The halving of feature size would yield an approximate doubling of the system speed and thus increasing the sample rate to more than 1MHz.

Multichip modules may be considered as yet another strategy for increasing performance of the deconvolution system. Instead of the system being composed of an array of individual ICs, the bare die may be placed upon a single substrate within the multichip module and interconnected without the need for I/O pads between each IC. The delays caused by these interchip connections on a PC board may cause problems at very high clock rates.

BIBLIOGRAPHY

Bibliography

1. P. Jansson, Deconvolution With Applications In Spectroscopy, New York: Academic Press, 1984, ch. 3-4, pp. 67-134.
2. P. H. Van Cittert, "Zum Einfluss der Spaltbreite auf die Intensitätsverteilung in Spektrallinien II", Z. Phys., vol. 69, 1931, pp. 298-308.
3. R. Schafer, R. Mersereau and M. Richards, "Constrained Iterative Restoration Algorithms", Proc. IEEE, vol. 69, 1981, pp. 432-450.
4. P. Crilly, "Evaluation of Jansson's Method For Resolving Overlapped Gas Chromatographic Peaks", Journal of Chemometrics, vol. 1, July 1987, pp. 175-183.
5. P. Crilly "Numerical Deconvolution of Gas Chromatograph Peaks Using Jansson's Method", Journal of Chemometrics, vol. 1, 1987, pp. 79-90.
6. P. Crilly "Enhancing the Deconvolution of Noisy Chromatographic Data By Jansson's Method", Journal of Chemometrics, vol. 4, Jan. 1990, pp. 51-59.
7. P. Crilly, "A Quantitative Evaluation of Various Iterative Deconvolution Algorithms", IEEE Trans. on Instrumentation and Measurement, in press, 1991.
8. J. V. McCanny and J. G. McWhirter, "Some Systolic Array Developments in the United Kingdom", IEEE Computer Magazine, vol. 20, pp. 58-59, July 1987.
9. J. V. McCanny and J. G. McWhirter, "Implementation of Signal Processing Functions Using 1-Bit Systolic Arrays", Electronics Letters, vol. 18, no. 6, pp. 241-243, March 1982.

10. J. V. McCanny, R. A. Evans and J. G. McWhirter, "Use of Unidirectional Dataflow in Bit Level Systolic Array Chips", Electronics Letters, vol. 22, no. 10, pp. 540-541, March 1986.
11. R. B. Whitted and P. B. Crilly, "A Digital Signal Processing Architecture for Iterative Deconvolution Restoration Algorithms", IEEE Trans. on Instrumentation and Measurement, vol. 41, no 1, pp. 147-151, February 1992.
12. P. Denyer and D. Renshaw, VLSI Signal Processing: A Bit-Serial Approach, Reading, Massachusetts: Addison-Wesley Publishing Co., 1985, ch 5-6,9, pg 98-119, 200-235.
13. Bouldin, Donald W., Computer-Aided Design of VLSI Systems II-Class Notes, Ferris Hall, The University of Tennessee, Knoxville, TN, Spring 1991.
14. Shung, B. and Jain, R. and Rimey, K. and Wand, E. and Srivastava, M. and Richards, B. and Lettang, E. and Azim, K. and Thon, L. and Hilfinger, P. and Rabaey, J. and Brodersen, R., "An Integrated CAD System for Algorithm-Specific IC Design", IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, vol. 10, no. 4, pp 447-463, April 1991.

APPENDIX

APPENDIX CONTENTS

Chip Operation

IC Signal Input/Output List

Files Used in Chip Layout and Testing

Chip Operation

This section is provided as a guide to the use of the deconvolution chip. The following discussions give the user some background on signal requirements, operation steps and descriptions of some software used to generate input values.

Input / Output Signals

The chip accepts three principle input signals:

- The signal estimate to be deconvolved, x_e^k
- The original signal estimate, g
- The impulse response function, h

and produces two output signals:

- The new signal estimate, x_e^{k+1}
- The original signal estimate, g

Each of these signals are 8-bit 2s complement integers. The inputs must be $\leq 1V$ p-p before they are digitized. For the impulse response function, h , the signal may be

amplified by a constant value before digitization to produce adequate signal resolution for the convolver. The allowable scale constants are: 1, 2, 4, 8, 16. This scaling of the h-vector effects the convolution result and the chip has been design to compensate for this by programming the convolver output format circuit prior to operation.

Programming

The programming mode is used to load information required by the chip to compute the deconvolution algorithm. The required information includes:

- the values of the impulse response function
- the value of the h-function scale factor

These data are all 8-bit values and are input to the chip sequentially via the programming port. These data values are entered in the following order:

- Scale factor control value for the impulse response function
- Impulse response function values.

The value of the scale factor control is chosen from the following:

- scale factor=1 control word = AA (hex)
- scale factor=2 control word = A9 (hex)
- scale factor=4 control word = A5 (hex)
- scale factor=8 control word = 95 (hex)
- scale factor=16 control word = 55 (hex)

The impulse response function values must be entered in the following manner. The

coefficient vector is divided into N/m groups where N is the number of coefficients in the vector and m is the number of iteration performed by each IP in the convolver. For the design in this thesis, the number of groups would be 2, with each group having 16 members. Starting with the highest numbered group, the values in that group are entered, in order, until m values have been input. The next highest numbered group is input next, in order, etc. until all groups have been input to the chip.

The steps required to program the chip are given below.

- Reset the chip by bringing the RST_B line to a low state momentarily
- Pull the START line high momentarily to initiate the program cycle
- P/R line will go to high state to signal program cycle active
- P/R line goes low and chip clocks are disabled
- Reset chip again to begin operation

IC SIGNAL INPUT/OUTPUT LIST

<u>SIGNAL NAME</u>	<u>DESCRIPTION</u>	<u># PINS</u>
P0-P7	Programming port	8
GI0-GI7	Original estimate input	8
GO0-GO7	Original estimate output	8
XI0-XI7	k th signal estimate input	8
XO0-XO7	$k+1$ th signal estimate output	8
PHICK	IP processor clock input	1
SCLK	Sample clock input	1
ACLK	Delayed sample clock input	1
START	Program start control	1
RST_B	IC reset control	1
P/R	Program/run mode indicator	1
Vdd	IC supply voltage	≥ 4
GND	IC ground	≥ 4

FILES USED IN CHIP LAYOUT AND TESTING

```
;ACC.sdl -- IP accumulator for deconvolution chip
;Rodney Whitted, Mar. 1992
```

```
(parent-cell ACC)
(parameters (N 20) (test 1))
(structure-processor dpp)
(layout-generator Flint a)
(subcells
  (adder          ACCUM ((N 20)))
  (mux2to1        MUX      ((N 20)))
  (scanmslatch    (PL ML) ((N 20)))
)
```

```
(instance ACCUM(
  (IN1      plout)
  (IN2      mlout)
  (OUT      ACCOUT)
  (CIN      CNTL_GND)
  (CININV   CNTL_VDD)
  (Vdd      Vdd)
  (GND      GND)
))
```

```
(instance MUX(
  (IN1      ACCIN)
  (IN2      ACCOUT)
  (OUT      mlin)
  (SEL1     A1)
  (SEL2     A1INV)
  (Vdd      Vdd)
  (GND      GND)
))
```

```
(instance ML(
  (IN      mlin)
  (OUT     mlout)
  (LOAD    PHI2[0])
  (LOADINV PHI2INV[0])
  (PHI1    PHI1[0])
  (PHI1INV PHI1INV[0])
  (SHIFT   CNTL_GND)
  (SHIFTINV CNTL_VDD)
  (Vdd     Vdd)
  (GND     GND)
))
```

```
(instance PL(
  (IN      PIN)
  (OUT     plout)
  (LOAD    PHI2[1])
  (LOADINV PHI2INV[1])
  (PHI1    PHI1[1])
  (PHI1INV PHI1INV[1])
  (SHIFT   CNTL_GND)
  (SHIFTINV CNTL_VDD)
  (Vdd     Vdd)
  (GND     GND)
)
```

)

```
(instance parent(
  ((terminal      PIN      (TERM_EDGE RIGHT))  PIN)
  ((terminal      ACCIN   (TERM_EDGE LEFT))   ACCIN)
  ((terminal      ACCOUT  (TERM_EDGE LEFT))   ACCOUT)

  ((terminal      PHI1    (TERMTYPE CLOCK))    PHI1  (width 2))
  ((terminal      PHI1INV (TERMTYPE CLOCK))    PHI1INV (width 2))
  ((terminal      PHI2    (TERMTYPE CLOCK))    PHI2  (width 2))
  ((terminal      PHI2INV (TERMTYPE CLOCK))    PHI2INV (width 2))
  ((terminal      A1      (TERMTYPE CLOCK))    A1)
  ((terminal      A1INV   (TERMTYPE CLOCK))    A1INV)

  ((terminal      CNTL_VDD (TERM_EDGE TOP))    CNTL_VDD)
  ((terminal      CNTL_GND (TERM_EDGE TOP))    CNTL_GND)

  ((terminal      Vdd     (TERMTYPE SUPPLY))    Vdd)
  ((terminal      GND     (TERMTYPE GROUND))    GND)
```

;DEFINE CONDITIONAL TEST TERMINALS

```
((terminal      mlin) (temp_mlin (CONDITIONAL (= test 1))
  (ALIAS "mlin")))

((terminal      mlout) (temp_mlout (CONDITIONAL (= test 1))
  (ALIAS "mlout")))

((terminal      plout) (temp_plout (CONDITIONAL (= test 1))
  (ALIAS "plout")))
```

)

(end-sdl)

|ACC.cmds -- command file for testing accumulator circuit of convolution IP
 |Rodney Whitted, Mar. 1992

h Vdd CNTL_VDD
 l GND CNTL_GND

*** 8.333 MHz Simulation***
 stepsize 5

vector PHI1 PHI1[{1:0}]
 vector PHI1INV PHI1INV[{1:0}]
 vector PHI2 PHI2[{1:0}]
 vector PHI2INV PHI2INV[{1:0}]

clock PHI1 00 00 00 00 00 00 00 00 00 00 00 00 00 00 11 11 11 11 11 11 11 11 11 11
 clock PHI1INV 11 11 11 11 11 11 11 11 11 11 11 11 11 11 00 00 00 00 00 00 00 00 00 00
 clock PHI2 00 11 11 11 11 11 11 11 11 11 11 11 00 00 00 00 00 00 00 00 00 00 00 00
 clock PHI2INV 11 00 00 00 00 00 00 00 00 00 00 00 11 11 11 11 11 11 11 11 11 11 11 11

vector PIN PIN[{19:0}]
 vector ACCIN ACCIN[{19:0}]
 vector ACCOUT ACCOUT[{19:0}]

vector plout plout[{19:0}]
 vector mlout mlout[{19:0}]
 vector mlin mlin[{19:0}]

set PIN 00010001000100010001

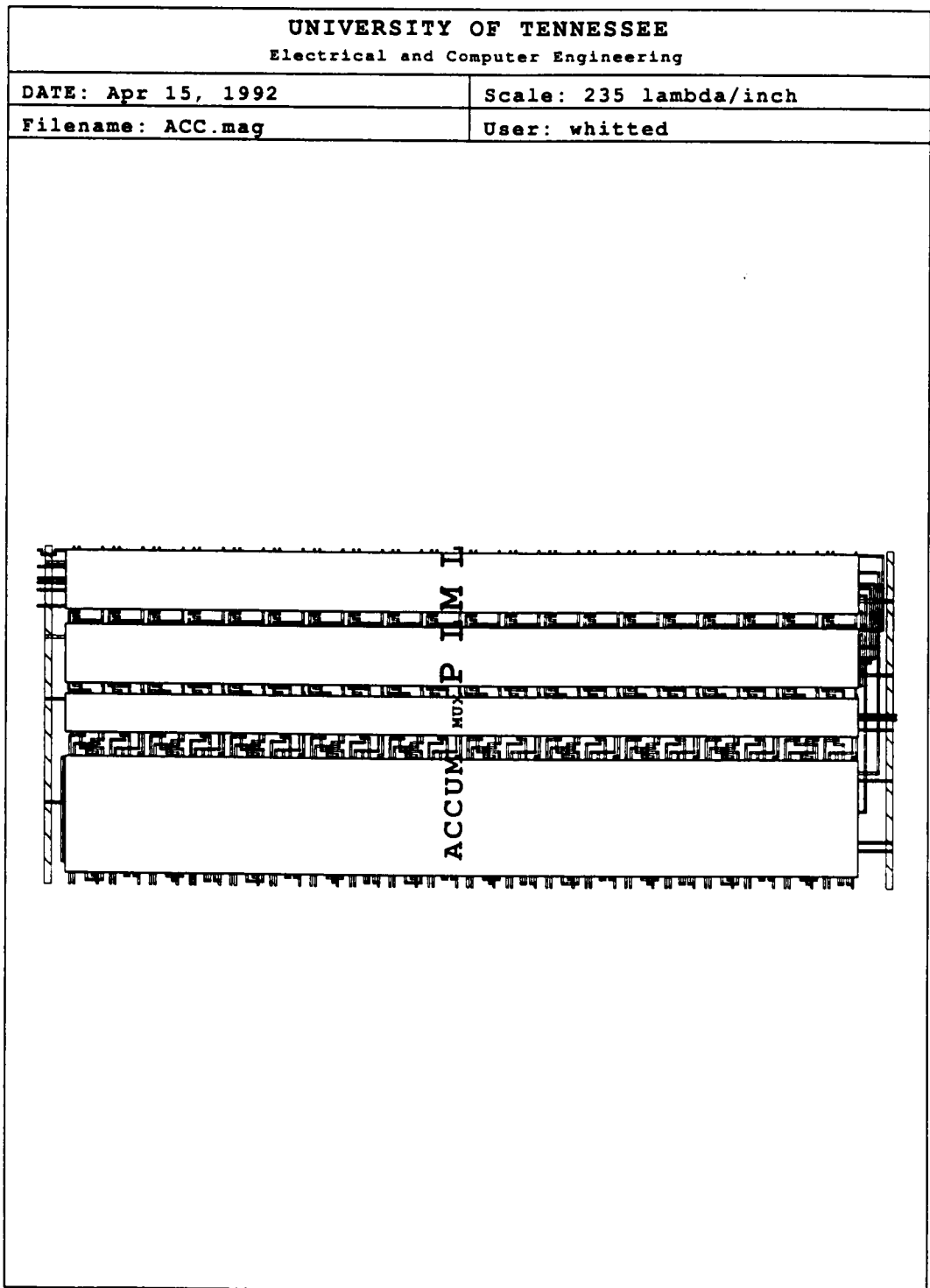
set ACCIN 00000000000000000000

V A1 010000000000000000

V A1INV 101111111111111111

analyzer PHI1 PHI2 A1 PIN ACCIN mlin mlout plout ACCOUT

R



ACC

Sun Apr 19 12:57:40 1992

PHI1	00	11	00	11	00	11	00	11	00	11	00	11	00	11	00	11	00	11	00	11	00
PHI2	11	00	11	00	11	00	11	00	11	00	11	00	11	00	11	00	11	00	11	00	11
A1																					
PIN	11111																				
ACCIN	00000																				
mlin	XXXXX	00000	11111	22222	33333	44444	55555	66666	77777	88888	99999	aaaaa	bbbbbb	ccccc	ddddd	eeeeee					
mlout	XXXXX	00000	11111	22222	33333	44444	55555	66666	77777	88888	99999	aaaaa	bbbbbb	ccccc	ddddd						
plout	XXXXX	11111																			
ACCOU	XXXXX	11111	22222	33333	44444	55555	66666	77777	88888	99999	aaaaa	bbbbbb	ccccc	ddddd	eeeeee						
time (ms)	0.0	250.0	500.0	750.0	1000.0	1250.0	1500.0	1750.0													

```
;ACLK.sdl -- clock tree for ACC
;Rodney Whitted, Mar. 1992
```

```
(parent-cell ACLK)
```

```
(layout-generator Flint)
```

```
(subcells
```

```
    (CBA    CB1    ((N 4)))
    (CBA    CB2    ((N 10)))
```

```
)
```

```
(instance CB1(
    (IN      clk1n1  (width 4))
```

```
    (Vdd     Vdd)
    (GND     GND)
```

```
))
```

```
(net clk1n1      ((parent PHI1      (net-base 0) (net-incr 1))))
```

```
(net clk1n1      ((parent PHI1INV (net-base 1) (net-incr 1))))
```

```
(net clk1n1      ((parent PHI2      (net-base 2) (net-incr 1))))
```

```
(net clk1n1      ((parent PHI2INV (net-base 3) (net-incr 1))))
```

```
(net clk1 (NETWIDTH 4) ((CB1 OUT)))
```

```
(instance CB2(
```

```
    (Vdd     Vdd)
    (GND     GND)
```

```
))
```

```
(net aclk ((parent A1)      (CB2 IN (term-base 0) (term-incr 1))))
```

```
(net aiclk ((parent A1INV) (CB2 IN (term-base 1) (term-incr 1))))
```

```
(net clk1 ((CB2 IN (width 2) (term-base 2) (term-incr 4) (net-base 0) (net-incr 0))))
```

```
(net clk1 ((CB2 IN (width 2) (term-base 3) (term-incr 4) (net-base 1) (net-incr 0))))
```

```
(net clk1 ((CB2 IN (width 2) (term-base 4) (term-incr 4) (net-base 2) (net-incr 0))))
```

```
(net clk1 ((CB2 IN (width 2) (term-base 5) (term-incr 4) (net-base 3) (net-incr 0))))
```

```
(net CLKOUT (NETWIDTH 10) ((CB2 OUT)))
```

```
(instance parent(
```

```
    ((terminal CLKOUT      (TERMTYPE CLOCK))      CLKOUT (width 10))
```

```
    ((terminal Vdd         (TERMTYPE SUPPLY))      Vdd)
```

```
    ((terminal GND        (TERMTYPE GROUND))      GND)
```

```
))
```

```
(end-sdl)
```

!ACLK.cmds -- commands to test ACC's clock tree
 !Rodney Whitted, Mar. 1992

h Vdd
 l GND

stepsize 5

clock PHI1 000000000000011111111111
 clock PHI1INV 11111111111111000000000000
 clock PHI2 01111111111110000000000000
 clock PHI2INV 10000000000001111111111111

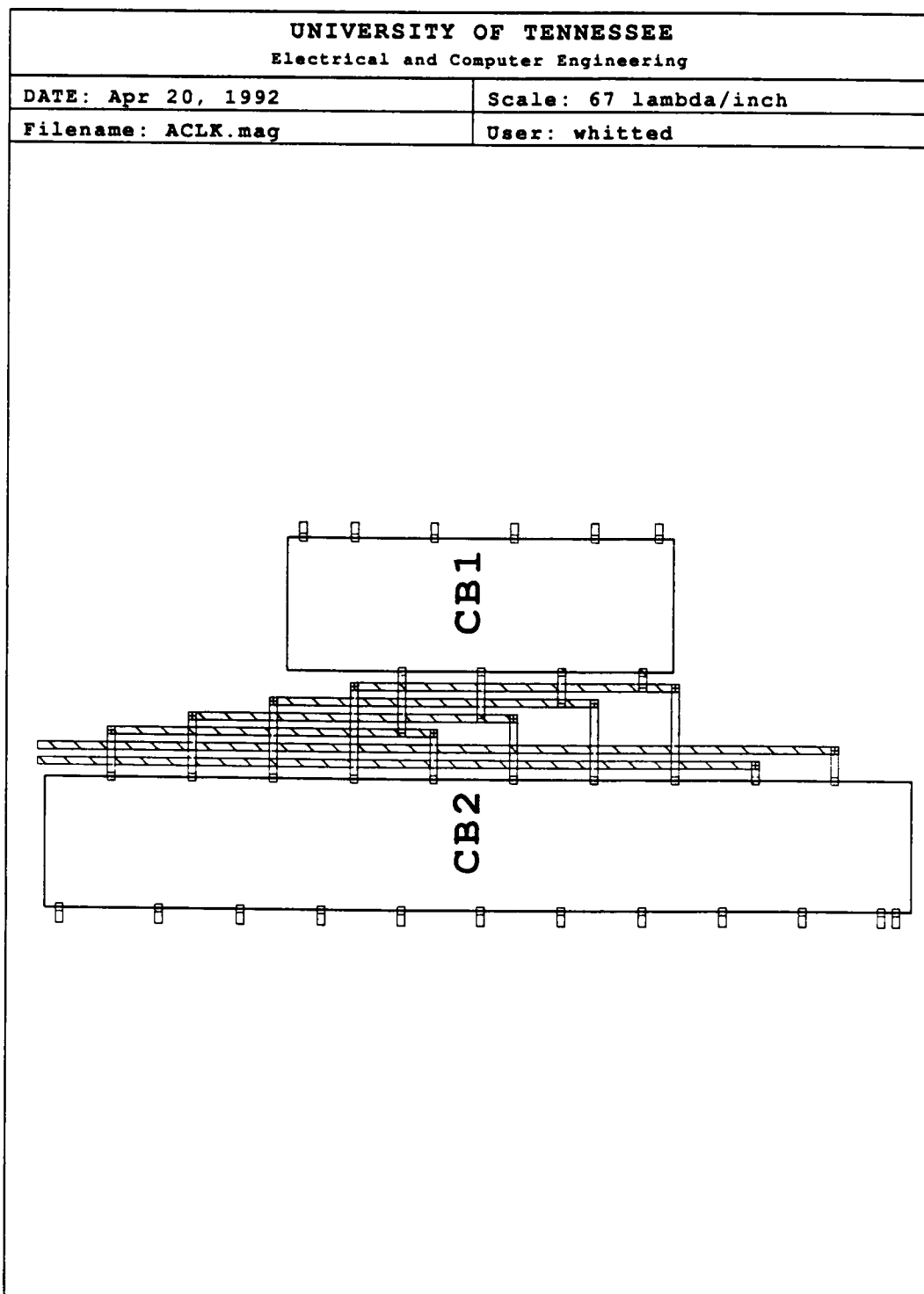
vector a CLKOUT[0]
 vector ainv CLKOUT[1]
 vector p1 CLKOUT[2]
 vector pli CLKOUT[3]
 vector p2 CLKOUT[4]
 vector p2i CLKOUT[5]

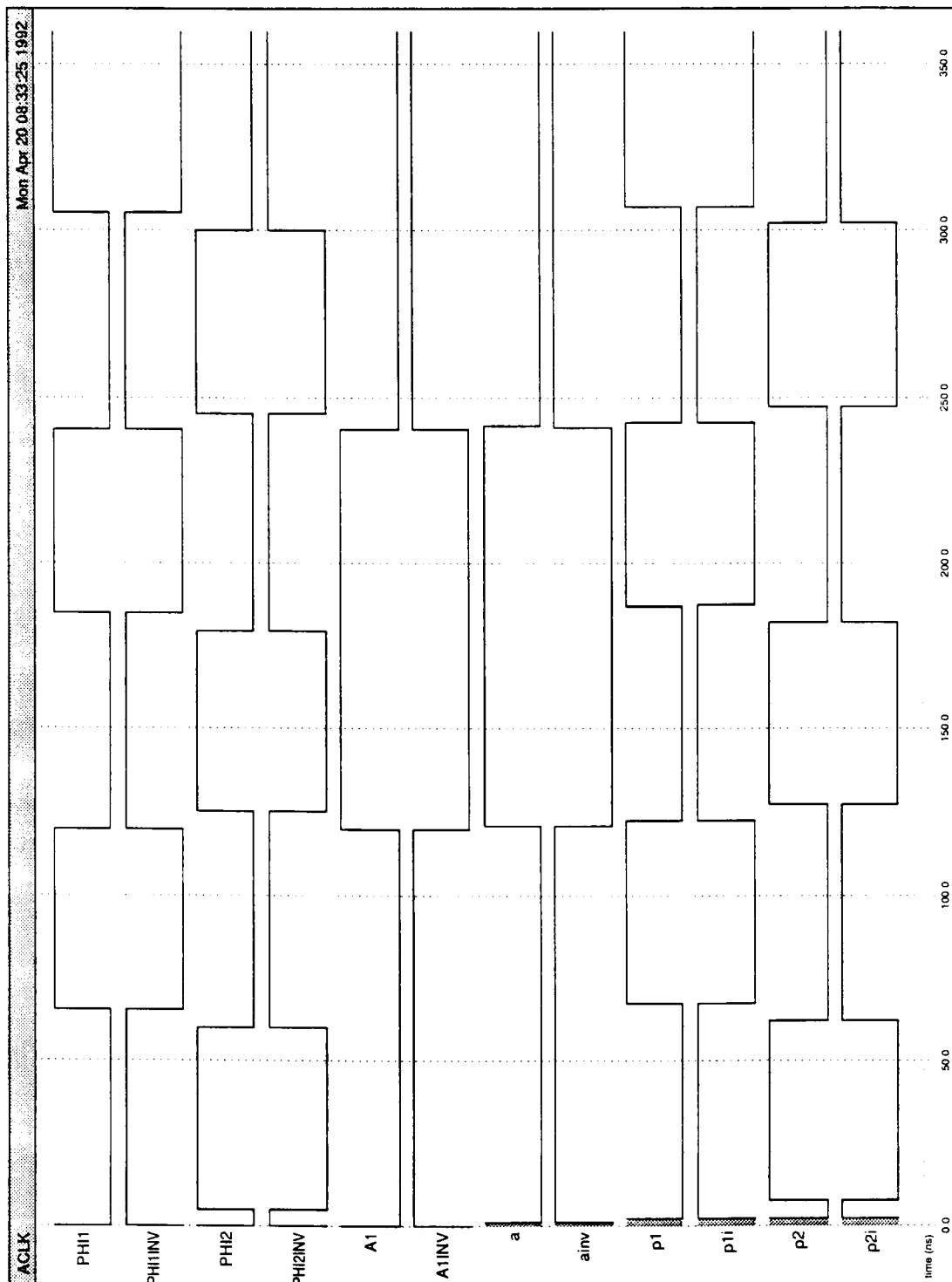
V A1 010

V A1INV 101

analyzer PHI1 PHI1INV PHI2 PHI2INV A1 A1INV a ainv p1 pli p2 p2i

R





```
;A_PIPE.sdl -- Arithmetic processing pipeline for deconvolution chip
;Rodney Whitted, Feb. 1992
```

```
(parent-cell A_PIPE)
(parameters (test 1))
(layout-generator Flint)
```

```
(subcells
  (GCLK          GCLK)
  (GSTATE        GSTATE)
  (MCLK          MCLK)
  (MATH          MATHPIPE)
  (mult          STAGE2 ((n 8) (m 8) (s 4) (csindex '(3 3 3 4))))
)
```

```
(instance GCLK(
  (S1          S1)
  (S1INV       S1INV)
  (S2          S2)
  (S2INV       S2INV)

  (CLKOUT      GCLKOUT      (width 16))

  (Vdd         Vdd)
  (GND         GND)
))
```

```
(instance GSTATE(
  (GIN         GIN      (width 8))
  (G           G        (width 8))
  (GOUT        GOUT     (width 8))

  (CNTL_VDD    Vdd)
  (CNTL_GND    GND)
  (Vdd         Vdd)
  (GND         GND)
))
```

```
(net GCLKOUT ((GSTATE S1 (width 2) (term-base 0) (term-incr 1) (net-base 0) (net-incr 0)))
(net GCLKOUT ((GSTATE S1INV (width 2) (term-base 0) (term-incr 1) (net-base 1) (net-incr 0)))
(net GCLKOUT ((GSTATE S2 (width 2) (term-base 0) (term-incr 1) (net-base 2) (net-incr 0)))
(net GCLKOUT ((GSTATE S2INV (width 2) (term-base 0) (term-incr 1) (net-base 3) (net-incr 0)))
```

```
(net GCLKOUT ((GSTATE S1 (width 2) (term-base 2) (term-incr 1) (net-base 4) (net-incr 0)))
(net GCLKOUT ((GSTATE S1INV (width 2) (term-base 2) (term-incr 1) (net-base 5) (net-incr 0)))
(net GCLKOUT ((GSTATE S2 (width 2) (term-base 2) (term-incr 1) (net-base 6) (net-incr 0)))
(net GCLKOUT ((GSTATE S2INV (width 2) (term-base 2) (term-incr 1) (net-base 7) (net-incr 0)))
```

```
(net GCLKOUT ((GSTATE S1 (width 2) (term-base 4) (term-incr 1) (net-base 8) (net-incr 0)))
(net GCLKOUT ((GSTATE S1INV (width 2) (term-base 4) (term-incr 1) (net-base 9) (net-incr 0)))
(net GCLKOUT ((GSTATE S2 (width 2) (term-base 4) (term-incr 1) (net-base 10) (net-incr 0)))
(net GCLKOUT ((GSTATE S2INV (width 2) (term-base 4) (term-incr 1) (net-base 11) (net-incr 0)))
```

```
(net GCLKOUT ((GSTATE S1 (width 1) (term-base 6) (term-incr 1) (net-base 12) (net-incr 0)))
(net GCLKOUT ((GSTATE S1INV (width 1) (term-base 6) (term-incr 1) (net-base 13) (net-incr 0)))
(net GCLKOUT ((GSTATE S2 (width 1) (term-base 6) (term-incr 1) (net-base 14) (net-incr 0)))
(net GCLKOUT ((GSTATE S2INV (width 1) (term-base 6) (term-incr 1) (net-base 15) (net-incr 0)))
```

```

(instance MCLK(
    (S1          S1)
    (S1INV       S1INV)
    (S2          S2)
    (S2INV       S2INV)

    (CLKOUT      MCLKOUT      (width 20))

    (Vdd         Vdd)
    (GND         GND)
))

```

```

(instance MATHPIPE(
    (XIN          XIN      (width 8))
    (XOUT         XOUT     (width 8))
    (GIN          G      (width 8))
    (CONIN        CONIN   (width 8))
    (DIFFOUT      DIFF    (width 8))
    (ADDR         ADDR    (width 8))
    (RAMIN        RAMIN   (width 8))
    (RAMOUT       RAMOUT   (width 8))

    (CNTL_VDD     Vdd)
    (CNTL_GND     GND)
    (Vdd          Vdd)
    (GND          GND)
))

```

```

(net CORR ((MATHPIPE PIN (width 8) (term-base 0) (term-incr 1) (net-base 5) (net-incr 1))))

```

```

(net MCLKOUT ((MATHPIPE S1 (width 2) (term-base 0) (term-incr 1) (net-base 0) (net-incr 0))))
(net MCLKOUT ((MATHPIPE S1INV (width 2) (term-base 0) (term-incr 1) (net-base 1) (net-incr 0))))
(net MCLKOUT ((MATHPIPE S2 (width 2) (term-base 0) (term-incr 1) (net-base 2) (net-incr 0))))
(net MCLKOUT ((MATHPIPE S2INV (width 2) (term-base 0) (term-incr 1) (net-base 3) (net-incr 0))))

```

```

(net MCLKOUT ((MATHPIPE S1 (width 2) (term-base 2) (term-incr 1) (net-base 4) (net-incr 0))))
(net MCLKOUT ((MATHPIPE S1INV (width 2) (term-base 2) (term-incr 1) (net-base 5) (net-incr 0))))
(net MCLKOUT ((MATHPIPE S2 (width 2) (term-base 2) (term-incr 1) (net-base 6) (net-incr 0))))
(net MCLKOUT ((MATHPIPE S2INV (width 2) (term-base 2) (term-incr 1) (net-base 7) (net-incr 0))))

```

```

(net MCLKOUT ((MATHPIPE S1 (width 2) (term-base 4) (term-incr 1) (net-base 8) (net-incr 0))))
(net MCLKOUT ((MATHPIPE S1INV (width 2) (term-base 4) (term-incr 1) (net-base 9) (net-incr 0))))
(net MCLKOUT ((MATHPIPE S2 (width 2) (term-base 4) (term-incr 1) (net-base 10) (net-incr 0))))
(net MCLKOUT ((MATHPIPE S2INV (width 2) (term-base 4) (term-incr 1) (net-base 11) (net-incr 0))))

```

```

(net MCLKOUT ((MATHPIPE S1 (width 2) (term-base 6) (term-incr 1) (net-base 12) (net-incr 0))))
(net MCLKOUT ((MATHPIPE S1INV (width 2) (term-base 6) (term-incr 1) (net-base 13) (net-incr 0))))
(net MCLKOUT ((MATHPIPE S2 (width 2) (term-base 6) (term-incr 1) (net-base 14) (net-incr 0))))
(net MCLKOUT ((MATHPIPE S2INV (width 2) (term-base 6) (term-incr 1) (net-base 15) (net-incr 0))))

```

```

(net MCLKOUT ((MATHPIPE S1 (width 1) (term-base 8) (term-incr 1) (net-base 16) (net-incr 0))))
(net MCLKOUT ((MATHPIPE S1INV (width 1) (term-base 8) (term-incr 1) (net-base 17) (net-incr 0))))
(net MCLKOUT ((MATHPIPE S2 (width 1) (term-base 8) (term-incr 1) (net-base 18) (net-incr 0))))
(net MCLKOUT ((MATHPIPE S2INV (width 1) (term-base 8) (term-incr 1) (net-base 19) (net-incr 0))))

```

```

(instance STAGE2(
  (X      DIFF      (width 8))
  (Y      RAMOUT    (width 8))
  (P      CORR      (width 15))

  (Vdd    Vdd)
  (GND    GND)
))

(instance parent(
  ((terminal GIN      (TERM_EDGE BOTTOM)) GIN      (width 8))
  ((terminal G      (TERM_EDGE BOTTOM)) G      (width 8))
  ((terminal GOUT    (TERM_EDGE BOTTOM)) GOUT    (width 8))
  ((terminal CONIN   (TERM_EDGE LEFT))  CONIN   (width 8))
  ((terminal XIN     (TERM_EDGE LEFT))  XIN     (width 8))
  ((terminal RAMIN   (TERM_EDGE RIGHT)) RAMIN   (width 8))
  ((terminal ADDR    (TERM_EDGE RIGHT)) ADDR    (width 8))
  ((terminal XOUT    (TERM_EDGE RIGHT)) XOUT    (width 8))
  ((terminal DIFF    (TERM_EDGE RIGHT)) DIFF    (width 8))
  ((terminal CORR    (TERM_EDGE RIGHT)) CORR    (width 15))

  ((terminal S1      (TERMTYPE CLOCK))  S1)
  ((terminal S1INV   (TERMTYPE CLOCK))  S1INV)
  ((terminal S2      (TERMTYPE CLOCK))  S2)
  ((terminal S2INV   (TERMTYPE CLOCK))  S2INV)

  ((terminal Vdd     (TERMTYPE SUPPLY))  Vdd)
  ((terminal GND     (TERMTYPE GROUND))  GND)
))

(end-sdl)

```

†Rodney Whitted, Mar. 1992

Correct operation produces the following outputs:

IXOUT = XX XX XX XX 00 00 00 10 13 16 1b 21 26 2c 32 32 30 2c 25 1d 16 0f 0a 06

$$|ADDR = XIN$$

1 GND

stepsize 25

[illegible]

```
clock      S1INV  11111111111111111111111111111111111111111111111111100000
```

[illegible][illegible]

```
vector  CONIN  CONIN[{7:0}]
```

vector	XIN	XIN[{7:0}]
--------	-----	------------

vector ADDR ADDR[{7:0}]

vector	GIN	GIN[{7:0}]
--------	-----	------------

vector	G	G[{7:0}]
--------	---	----------

```
vector RAMIN RAMIN[{7:0}]
```

vector XOUT XOUT[{7:0}]

vector GOUT GOUT[{7:0}]

```
VCONIN00000000 00000000 00000000 00010100 00010110 00011001 00011011 00011101 00011111
00100001 00100001 00100001 00100000 00011111 00011100 00011001 00010110 00010011 00010000
00001100 00001010 00000111 00000101
```

```
VGIN 00010001 00010100 00010111 00011011 00100000 00100100 00101000 00101011 00101011
00101010 00100111 00100010 00011100 00010110 00010000 00001011 00000111 00000101 00000011
00000010 00000001 00000000 00000000
```

```
V RAMIN 00000000 00000000 00000000 00000000 00001001 00001011 00001100 00001110 00010001
00010011 00010101 00010111 00010111 00010110 00010101 00010010 00001111 00001100 00001001
00000110 00000100 00000011 00000010
```

```
VXIN  00000000 00000000 00000000 00010001 00010100 00010111 00011011 00100000 00100100
00101000 00101011 00101011 00101010 00100111 00100010 00011100 00010110 00010000 00001011
00000111 00000101 00000011 00000010
```

```
analyzer S1 S2 GIN G GOUT CONIN XIN XOUT ADDR RAMIN
```

R

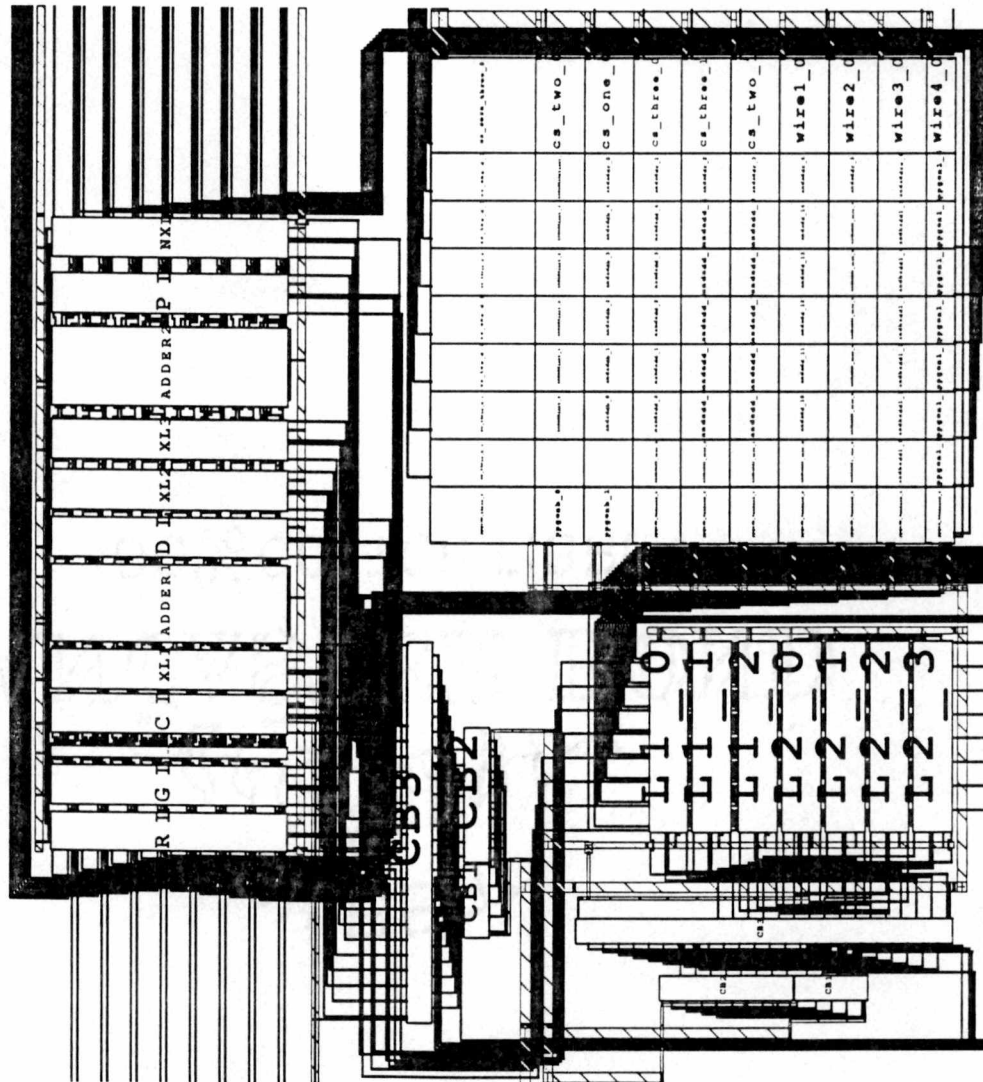
UNIVERSITY OF TENNESSEE
Electrical and Computer Engineering

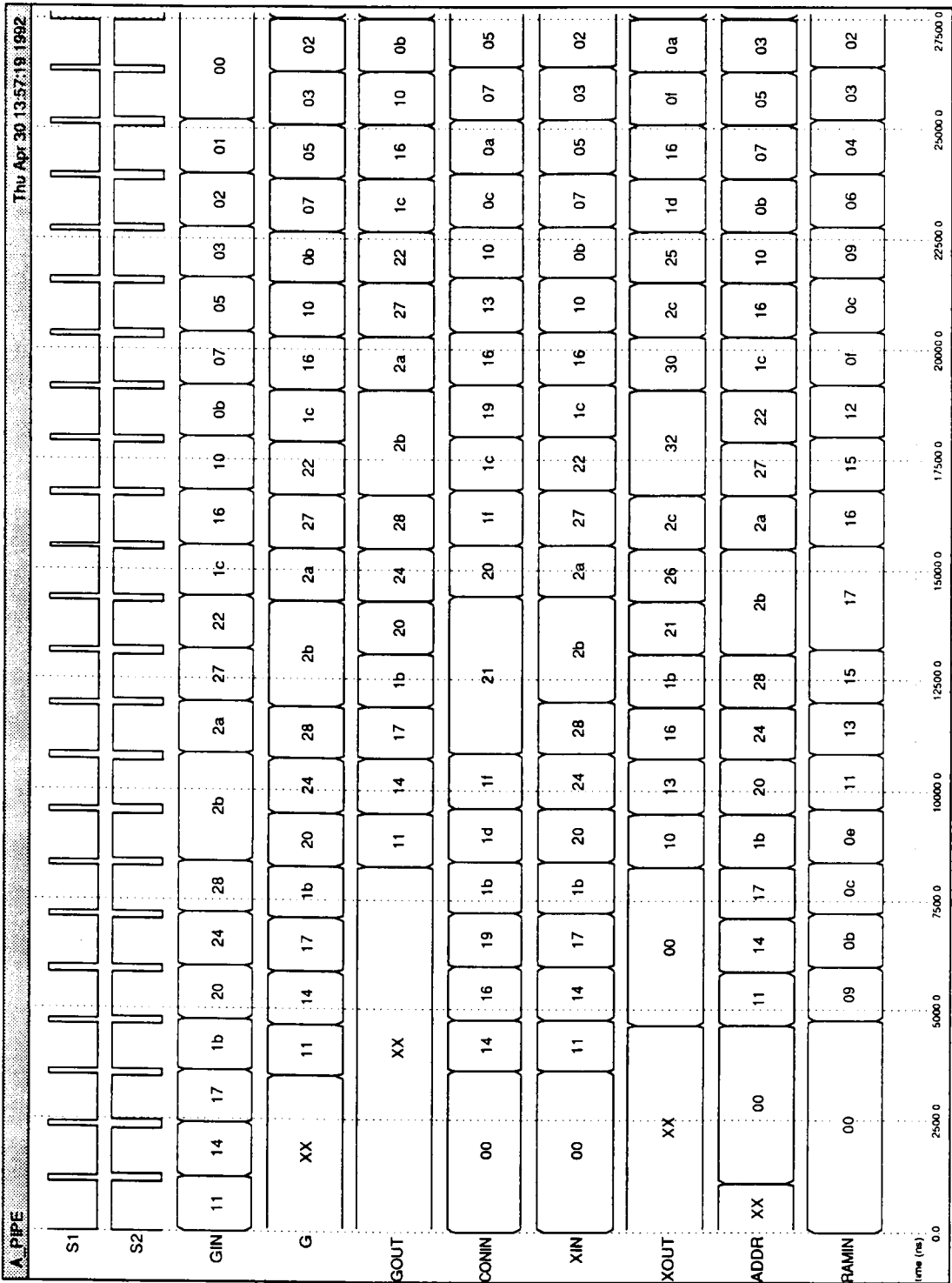
DATE: May 1, 1992

Scale: 399 lambda/inch

Filename: CORE MATH.mag

User: whitted





```
;CBA.sdl -- dpp buffer array for building clock trees
;Rodney Whitted, Mar. 1992
```

```
(parent-cell CBA)
```

```
(parameters N)
(structure-processor dpp)
(layout-generator Flint a)
```

```
(subcells
  (bufferhuge      CB      ((N N)))
)
```

```
(instance CB(
  (IN      IN)
  (OUT     OUT)

  (Vdd     Vdd)
  (GND     GND)
))
```

```
(instance parent(
  ((terminal IN      (TERM_EDGE LEFT))  IN)
  ((terminal OUT     (TERM_EDGE RIGHT)) OUT)

  ((terminal Vdd     (TERMTYPE SUPPLY))  Vdd)
  ((terminal GND     (TERMTYPE GROUND))  GND)
))
```

```
(end-sdl)
```

;CLOCK.sdl -- Standard cell 2-phase clock generator with ENABLE and GATE control
;Rodney Whitted, Mar. 1992

```
(parent-cell CLOCK (bag FLATTEN_TO (CELLCLASS LEAF)))
(parameters (chain 7))
(layout-generator Stdcell)
```

```
(dotimes (i chain)
  (subcells (invf101 I)

    (instance I(
      (A1      (inet (ALIAS (if (= i 0)
                            (format nil "clkln")
                            (format nil "ic[~D]" (- i 1))
                            )))
      (O      (onet (ALIAS (if (= i (- chain 1))
                            (format nil "ochain")
                            (format nil "ic[~D]" i)
                            )))
    )))
  )
```

```
(subcells
  (invf101 (INV1 INV2 P1IBUFF P2IBUFF))
  (norf201 (NOR1 NOR2))
  (norf211 OR)
  (swcf020 TG)
  (buff101 (P1IBUFF P2IBUFF))
)
```

```
(net Vdd (NETTYPE SUPPLY))
(net GND (NETTYPE GROUND))
```

```
(instance TG(
  (DATA1      CLK)
  (O          cin)
  (CTL2      ENABLE)
))
```

```
(instance OR(
  (A1      GATE)
  (B1      cin)
  (O1      clkln)
))
```

```
(instance INV1(
  (A1      ochain)
  (O       innor1)
))
```

```
(instance NOR1(
  (A1      innor1)
  (B1      clkln)
  (O       phi2net)
))
```

```
(instance P2IBUFF(
```

```

        (A1    phi2net)
        (O      PHI2)
    ))

(instance P2IBUFF(
    (A1    phi2net)
    (O      PHI2INV)
))

(instance INV2(
    (A1    clkin)
    (O      innor2)
))

(instance NOR2(
    (A1    innor2)
    (B1    ochain)
    (O      philnet)
))

(instance P1BUFF(
    (A1    philnet)
    (O      PHI1)
))

(instance P1IBUFF(
    (A1    philnet)
    (O      PHI1INV)
))

(instance parent(
    ((terminal CLK          (TERM_EDGE LEFT)) CLK)
    ((terminal ENABLE      (TERM_EDGE LEFT)) ENABLE)
    ((terminal GATE   (TERM_EDGE LEFT)) GATE)
    ((terminal PHI1      (TERM_EDGE RIGHT)) PHI1)
    ((terminal PHI1INV   (TERM_EDGE RIGHT)) PHI1INV)
    ((terminal PHI2      (TERM_EDGE RIGHT)) PHI2)
    ((terminal PHI2INV   (TERM_EDGE RIGHT)) PHI2INV)

    ((terminal Vdd       (TERMTYPE SUPPLY)) Vdd)
    ((terminal GND       (TERMTYPE GROUND)) GND)
))

(end-sdl)

```

| CLOCK.cmds -- IRSIM command file for testing CLOCK.sdl
| Rodney Whitted, Mar. 1992

h Vdd
l GND

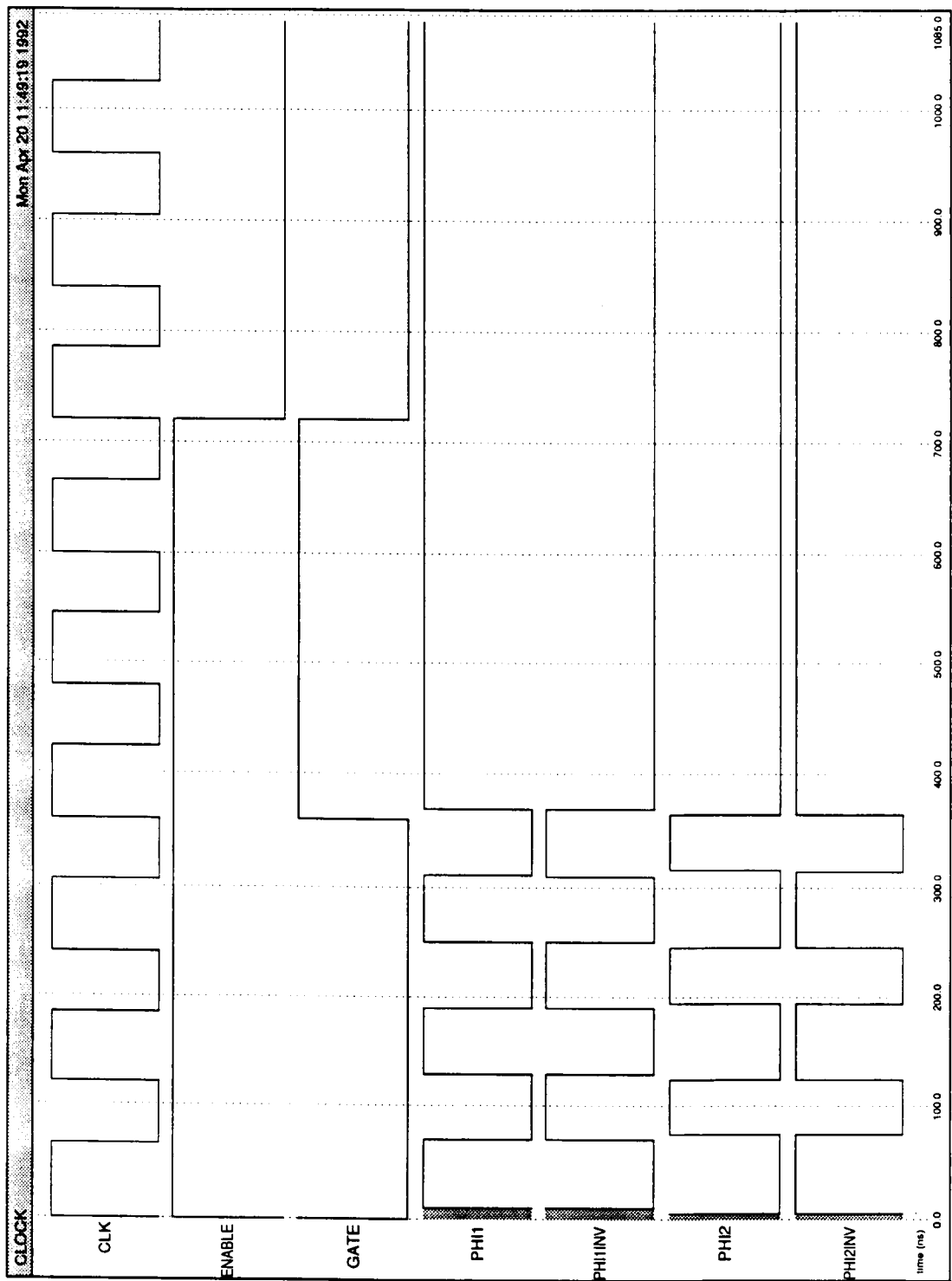
stepsize 5
clock CLK 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0

V ENABLE 1 1 1 1 1 1 0 0 0
V GATE 0 0 0 1 1 1 0 0 0

analyzer CLK ENABLE GATE PHI1 PHI1INV PHI2 PHI2INV

R

[illegible]



```
;CONTROL.sdl -- program cycle controller and clock generators for deconvolution chip
; Rodney B. Whitted, March 1992
```

```
(parent-cell CONTROL)
(layout-generator Flint)
```

```
(subcells
  (PROGCONTROL PC)
  (CLOCK (SCLKGEN ACLKGEN PCLKGEN))
)
```

```
(instance PC(
  (START START)
  (P/R P/R)
  (P/RINV P/RINV)
  (CLKENB CLKENB)
  (RST_B RST_B)
  (CLK_B PHI1INV)

  (CNTL_VDD Vdd)
  (CNTL_GND GND)
  (Vdd Vdd)
  (GND GND)
))
```

```
(instance SCLKGEN(
  (CLK SCLK)
  (ENABLE CLKENB)
  (GATE P/R)

  (PHI1 S1)
  (PHI1INV S1INV)
  (PHI2 S2)
  (PHI2INV S2INV)

  (Vdd Vdd)
  (GND GND)
))
```

```
(instance ACLKGEN(
  (CLK ACLK)
  (ENABLE CLKENB)
  (GATE P/R)

  (PHI1 A1)
  (PHI1INV A1INV)
  (PHI2 A2)
  (PHI2INV A2INV)

  (Vdd Vdd)
  (GND GND)
))
```

```
(instance PCLKGEN(
  (CLK PHICLK)
  (ENABLE CLKENB)
  (GATE GND)
))
```

```

(PHI1          PHI1)
(PHI1INV       PHI1INV)
(PHI2          PHI2)
(PHI2INV       PHI2INV)

(Vdd           Vdd)
(GND           GND)
))

(instance parent(
  ((terminal    START      (TERM_EDGE LEFT))  START)
  ((terminal    RST_B      (TERM_EDGE LEFT))  RST_B)
  ((terminal    PHICLK     (TERM_EDGE LEFT))  PHICLK)
  ((terminal    SCLK       (TERM_EDGE LEFT))  SCLK)
  ((terminal    ACLK       (TERM_EDGE LEFT))  ACLK)

  ((terminal    P/R        (TERM_EDGE RIGHT)) P/R)
  ((terminal    P/RINV     (TERM_EDGE RIGHT)) P/RINV)

  ((terminal    S1         (TERM_EDGE RIGHT)) S1)
  ((terminal    S1INV      (TERM_EDGE RIGHT)) S1INV)
  ((terminal    S2         (TERM_EDGE RIGHT)) S2)
  ((terminal    S2INV      (TERM_EDGE RIGHT)) S2INV)
  ((terminal    A1         (TERM_EDGE RIGHT)) A1)
  ((terminal    A1INV      (TERM_EDGE RIGHT)) A1INV)
  ((terminal    A2         (TERM_EDGE RIGHT)) A2)
  ((terminal    A2INV      (TERM_EDGE RIGHT)) A2INV)
  ((terminal    PHI1       (TERM_EDGE RIGHT)) PHI1)
  ((terminal    PHI1INV    (TERM_EDGE RIGHT)) PHI1INV)
  ((terminal    PHI2       (TERM_EDGE RIGHT)) PHI2)
  ((terminal    PHI2INV    (TERM_EDGE RIGHT)) PHI2INV)

  ((terminal    Vdd        (TERMTYPE SUPPLY)) Vdd)
  ((terminal    GND        (TERMTYPE GROUND)) GND)
))
(end-sdl)

```

| CONTROL.cmds -- IRSIM command file for testing CONTROL.sdl
| Rodney Whitted, Mar. 1992

h Vdd
l GND

stepsize 60

clock PHICLK 010101010101010101010101010101010101010101

clock SCLK 11000000000000000000000000000000000000000000

clock ACLK 00110000000000000000000000000000000000000000

V RST_B 0111100

V START 0010000

analyzer RST_B START P/R PHI1 PHI2 S1 S2 A1 A2

R

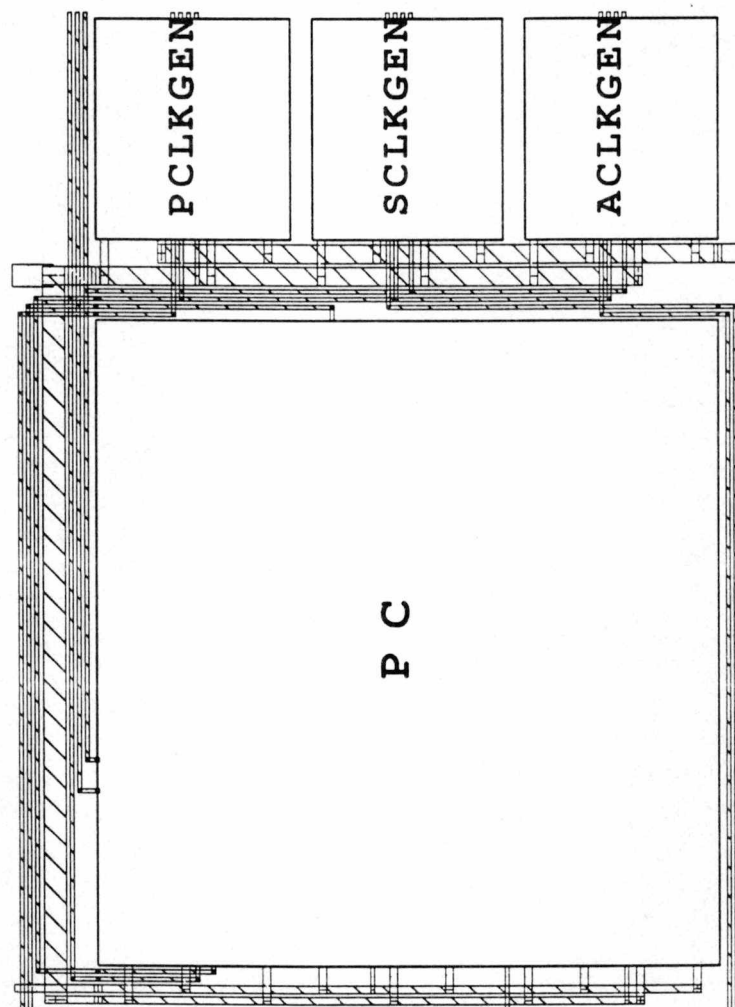
UNIVERSITY OF TENNESSEE
Electrical and Computer Engineering

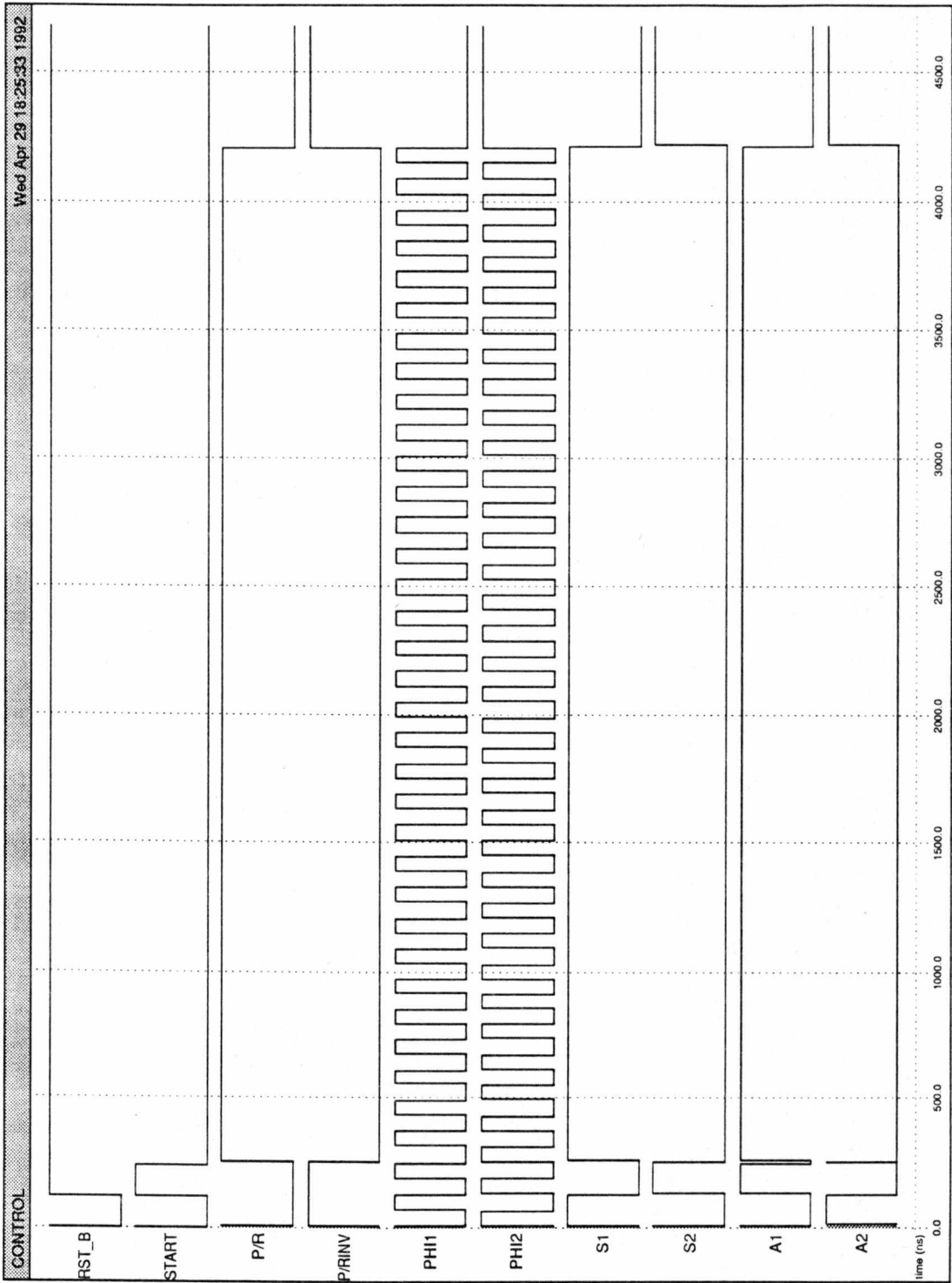
DATE: Apr 20, 1992

Scale: 150 lambda/inch

Filename: CONTROL.mag

User: whitted





```
;CORE.sdl -- Iterative Deconvolution Processor chip core
;Rodney Whitted, Mar. 1992
```

```
(parent-cell CORE)
(layout-generator Flint)
```

```
(subcells
  (IPARRAY CONVOLVER)
  (FORMAT FORMAT)
  (A_PIPE MATH)
  (CONTROL CONTROL)
)

(instance CONVOLVER(
  (HIN      HIN      (width 8))
  (HOUT     hout1    (width 8))
  (XIN      XIN      (width 8))
  (XCOUT    XCOUT    (width 8))
  (ACCOUT   ACCOUT   (width 20))

  (PHI1     PHI1)
  (PHI1INV  PHI1INV)
  (PHI2     PHI2)
  (PHI2INV  PHI2INV)
  (S1       S1)
  (S1INV    S1INV)
  (A1       A1)
  (A1INV    A1INV)
  (P/R      P/R)
  (P/RINV   P/RINV)

  (Vdd      Vdd)
  (GND      GND)
))
```

```
(instance FORMAT(
  (HIN      hout1    (width 8))
  (XIN      XCOUT    (width 8))
  (XOUT     fxout    (width 8))
  (ACCOUT   FORMATOUT (width 13))

  (PHI1     PHI1)
  (PHI1INV  PHI1INV)
  (PHI2     PHI2)
  (PHI2INV  PHI2INV)
  (S1       S1)
  (S1INV    S1INV)
  (A1       A1)
  (A1INV    A1INV)
  (P/R      P/R)
  (P/RINV   P/RINV)

  (Vdd      Vdd)
  (GND      GND)
))
```

```
(net ACCOUT ((FORMAT ACCIN (width 13) (term-base 0)(term-incr 1)(net-base 7)(net-incr 1))))
```

```

(instance MATH(
  (XIN      fxout      (width 8))
  (XOUT     XOUT       (width 8))
  (GIN      GIN        (width 8))
  (G        G          (width 8))
  (GOUT     GOUT       (width 8))
  (ADDR     ADDR       (width 8))
  (RAMIN    RAMIN      (width 8))
  (DIFF     DIFF       (width 8))
  (CORR     CORR       (width 15))

  (S1       S1)
  (S1INV    S1INV)
  (S2       S2)
  (S2INV    S2INV)

  (Vdd      Vdd)
  (GND      GND)
))

```

```

(net FORMATOUT ((MATH CONIN (width 8) (term-base 0) (term-incr 1) (net-base 0) (net-incr 1))))

```

```

(instance CONTROL(
  (START    START)
  (RST_B    RST_B)
  (P/R      P/R)
  (P/RINV   P/RINV)

  (SCLK     SCLK)
  (ACLK     ACLK)
  (PHICLK   PHICLK)

  (PHI1     PHI1)
  (PHI1INV  PHI1INV)
  (PHI2     PHI2)
  (PHI2INV  PHI2INV)
  (S1       S1)
  (S1INV    S1INV)
  (S2       S2)
  (S2INV    S2INV)
  (A1       A1)
  (A1INV    A1INV)

  (Vdd      Vdd)
  (GND      GND)
))

```

```

(instance parent(
  ((terminal START (TERMTYPE SIGNAL)) START)
  ((terminal P/R   (TERMTYPE SIGNAL)) P/R)
  ((terminal RST_B (TERMTYPE SIGNAL)) RST_B)

  ((terminal XIN   (TERMTYPE SIGNAL)) XIN (width 8))
  ((terminal GIN   (TERMTYPE SIGNAL)) GIN (width 8))
  ((terminal HIN   (TERMTYPE SIGNAL)) HIN (width 8))

```

```

((terminal ADDR (TERMTYPE SIGNAL)) ADDR (width 8))
((terminal RAMIN (TERMTYPE SIGNAL)) RAMIN (width 8))

((terminal XOUT (TERMTYPE SIGNAL)) XOUT (width 8))
((terminal GOUT (TERMTYPE SIGNAL)) GOUT (width 8))

((terminal PHCLK (TERMTYPE CLOCK)) PHCLK)
((terminal SCLK (TERMTYPE CLOCK)) SCLK)
((terminal ACLK (TERMTYPE CLOCK)) ACLK)

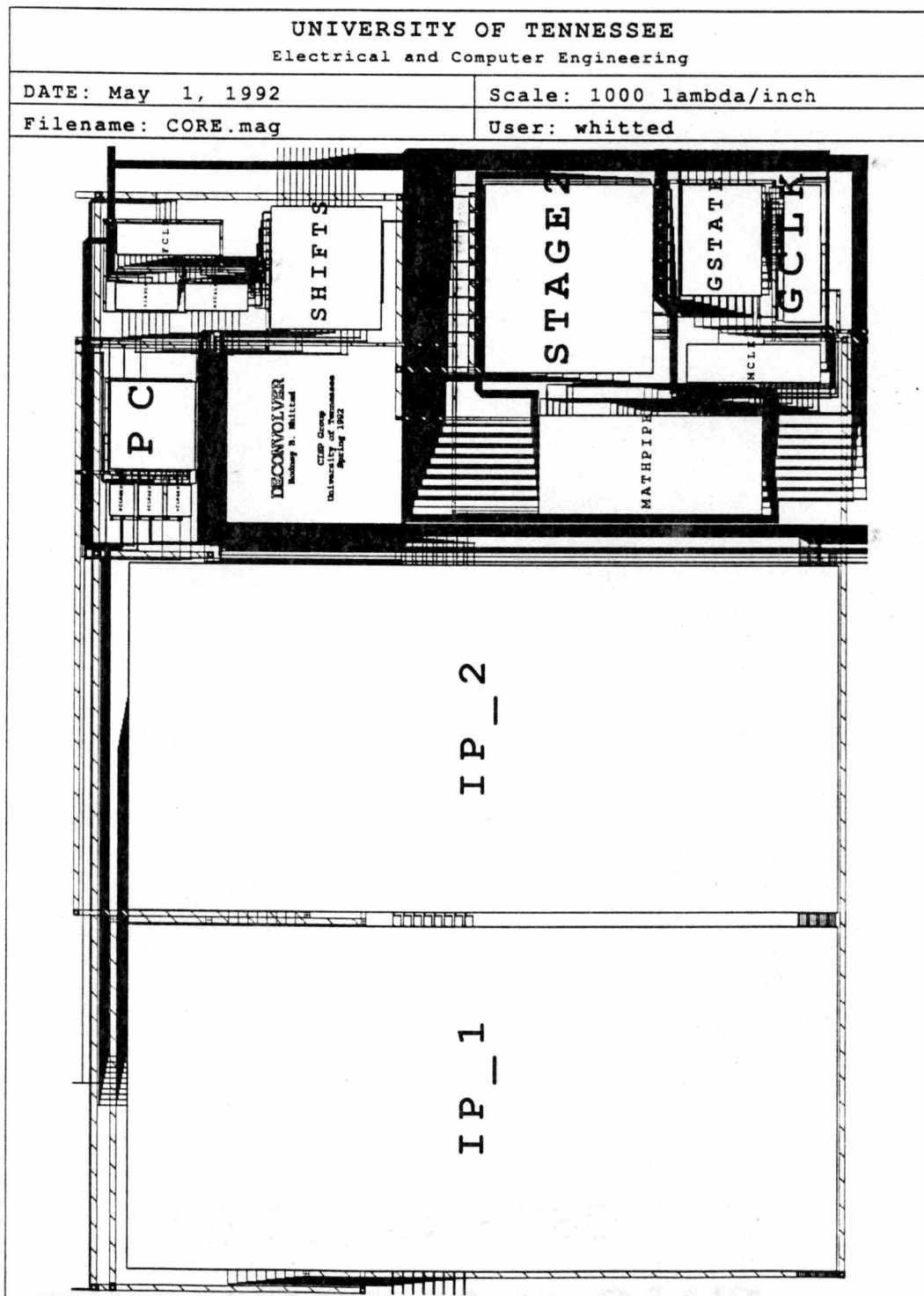
((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
((terminal GND (TERMTYPE GROUND)) GND)

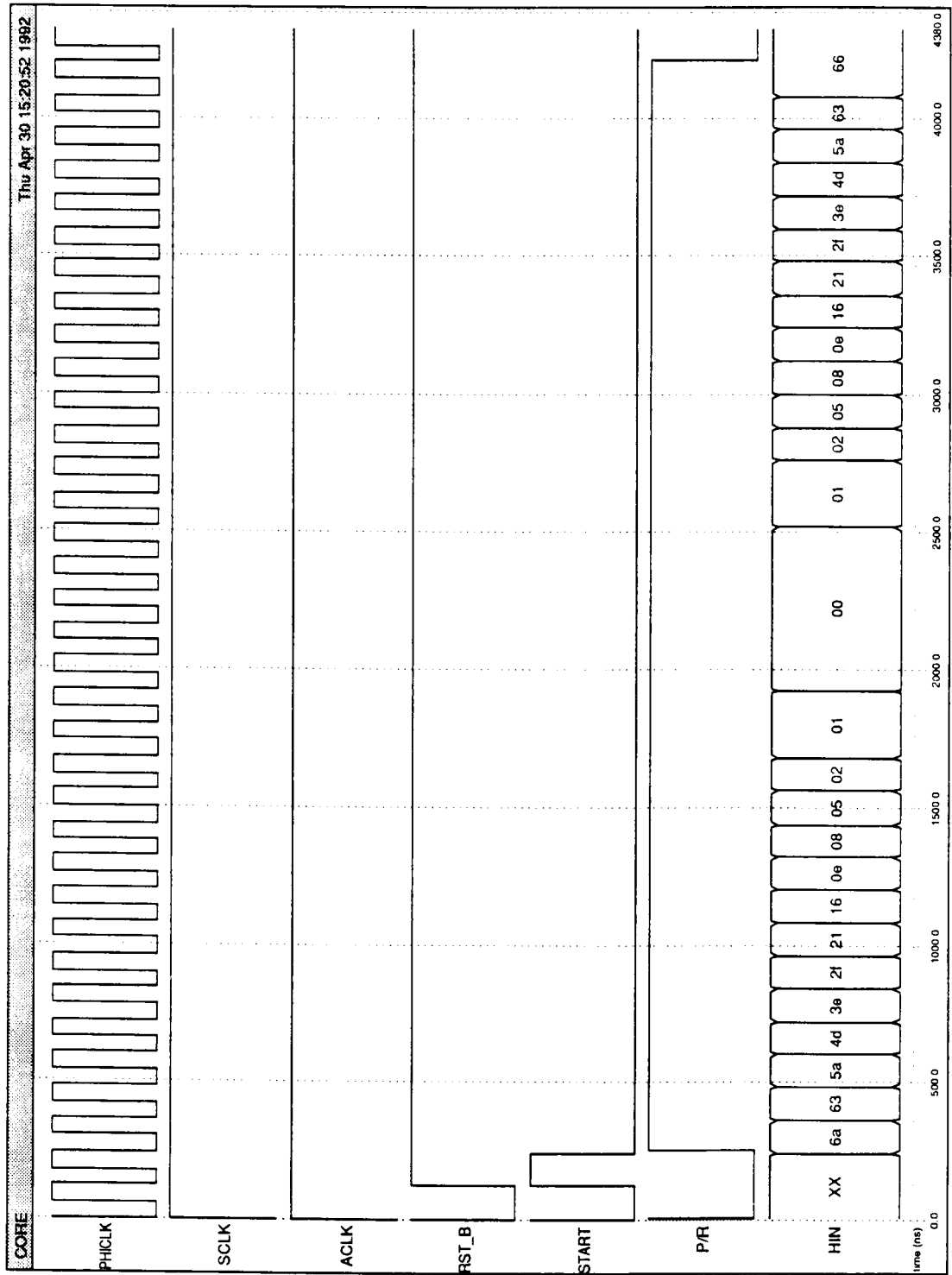
((terminal ACCOUT (TERMTYPE SIGNAL)) ACCOUT (width 20))
((terminal FORMATOUT (TERMTYPE SIGNAL)) FORMATOUT (width 13))
((terminal XCOUT (TERMTYPE SIGNAL)) XCOUT (width 8))
((terminal fxout (TERMTYPE SIGNAL)) fxout (width 8))
((terminal G (TERMTYPE SIGNAL)) G (width 8))
((terminal DIFF (TERMTYPE SIGNAL)) DIFF (width 8))
((terminal CORR (TERMTYPE SIGNAL)) CORR (width 15))

))

(end-sdl)

```





CORE		Thu Apr 30 16:52:34 1992															
SCLK																	
XIN	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f	10
ACCOUNT	00000																
FORMATOUT																	
ADDR	XX	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
RAMIN																	
DIFF																	
CORR																	
XOUT																	
GOUT																	
TIME (hr)		5:40:01.0	10:00:00.0	15:00:00.0	20:00:00.0	25:00:00.0	30:00:00.0	35:00:00.0	40:00:00.0	45:00:00.0	50:00:00.0	55:00:00.0	60:00:00.0	65:00:00.0	70:00:00.0	75:00:00.0	80:00:00.0

```
;FCLK.sdl -- clock tree for FORMAT
;Rodney Whitted, Mar. 1992
```

```
(parent-cell FCLK)
```

```
(layout-generator Flint)
```

```
(subcells
```

```
    (CBA    CB1    ((N 4)))
    (CBA    CB2    ((N 18)))
)
```

```
(instance CB1(
    (IN      clk1n1  (width 4))
```

```
    (Vdd     Vdd)
    (GND     GND)
```

```
))
(net clk1n1    ((parent PHI1    (net-base 0) (net-incr 1))))
(net clk1n1    ((parent PHI1INV (net-base 1) (net-incr 1))))
(net clk1n1    ((parent PHI2    (net-base 2) (net-incr 1))))
(net clk1n1    ((parent PHI2INV (net-base 3) (net-incr 1))))
```

```
(net clk1 (NETWIDTH 4)    ((CB1 OUT)))
```

```
(instance CB2(
    (Vdd     Vdd)
    (GND     GND)
```

```
))
(net aclk ((parent A1)      (CB2 IN (term-base 0) (term-incr 1))))
(net aiclk ((parent A1INV) (CB2 IN (term-base 1) (term-incr 1))))
(net prclk ((parent P/R)    (CB2 IN (term-base 2) (term-incr 1))))
(net priclk ((parent P/RINV) (CB2 IN (term-base 3) (term-incr 1))))
(net sclk ((parent S1)      (CB2 IN (term-base 4) (term-incr 1))))
(net siclk ((parent S1INV) (CB2 IN (term-base 5) (term-incr 1))))
```

```
(net clk1 ((CB2 IN (width 3) (term-base 6) (term-incr 4) (net-base 0) (net-incr 0))))
(net clk1 ((CB2 IN (width 3) (term-base 7) (term-incr 4) (net-base 1) (net-incr 0))))
(net clk1 ((CB2 IN (width 3) (term-base 8) (term-incr 4) (net-base 2) (net-incr 0))))
(net clk1 ((CB2 IN (width 3) (term-base 9) (term-incr 4) (net-base 3) (net-incr 0))))
```

```
(net CLKOUT (NETWIDTH 18)    ((CB2 OUT)))
```

```
(instance parent(
    ((terminal CLKOUT          (TERMTYPE CLOCK))    CLKOUT (width 18))
```

```
    ((terminal Vdd             (TERMTYPE SUPPLY))    Vdd)
    ((terminal GND             (TERMTYPE GROUND))    GND)
```

```
))
```

```
(end-sdl)
```

!FCLK.cmds -- commands to test FORMAT's clock tree
 !Rodney Whitted, Mar. 1992

h Vdd
 l GND

stepsize 5

clock PHI1 000000000000011111111111
 clock PHI1INV 111111111111110000000000
 clock PHI2 011111111111100000000000
 clock PHI2INV 100000000000011111111111

vector a CLKOUT[0]
 vector ainv CLKOUT[1]
 vector pr CLKOUT[2]
 vector pri CLKOUT[3]
 vector s CLKOUT[4]
 vector si CLKOUT[5]
 vector p1 CLKOUT[6]
 vector pli CLKOUT[7]
 vector p2 CLKOUT[8]
 vector p2i CLKOUT[9]

V A1 0 1 0

V A1INV 1 0 1

V P/R 0 1 0

V P/RINV 1 0 1

V S1 0 1 0

V S1INV 1 0 1

analyzer PHI1 PHI2 p1 p2 A1 P/R S1 a pr s

R

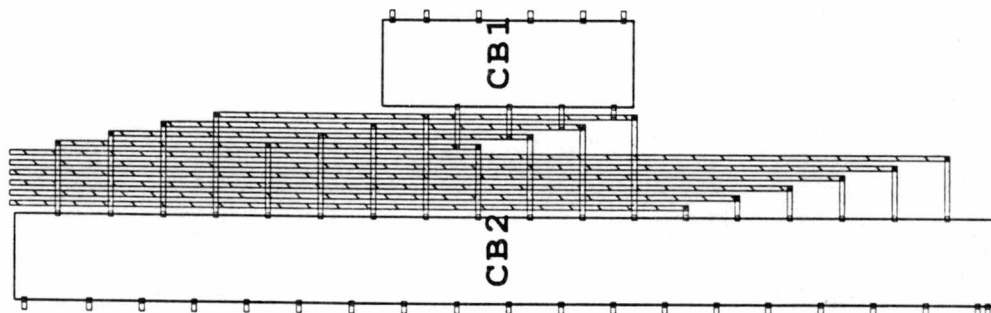
UNIVERSITY OF TENNESSEE
Electrical and Computer Engineering

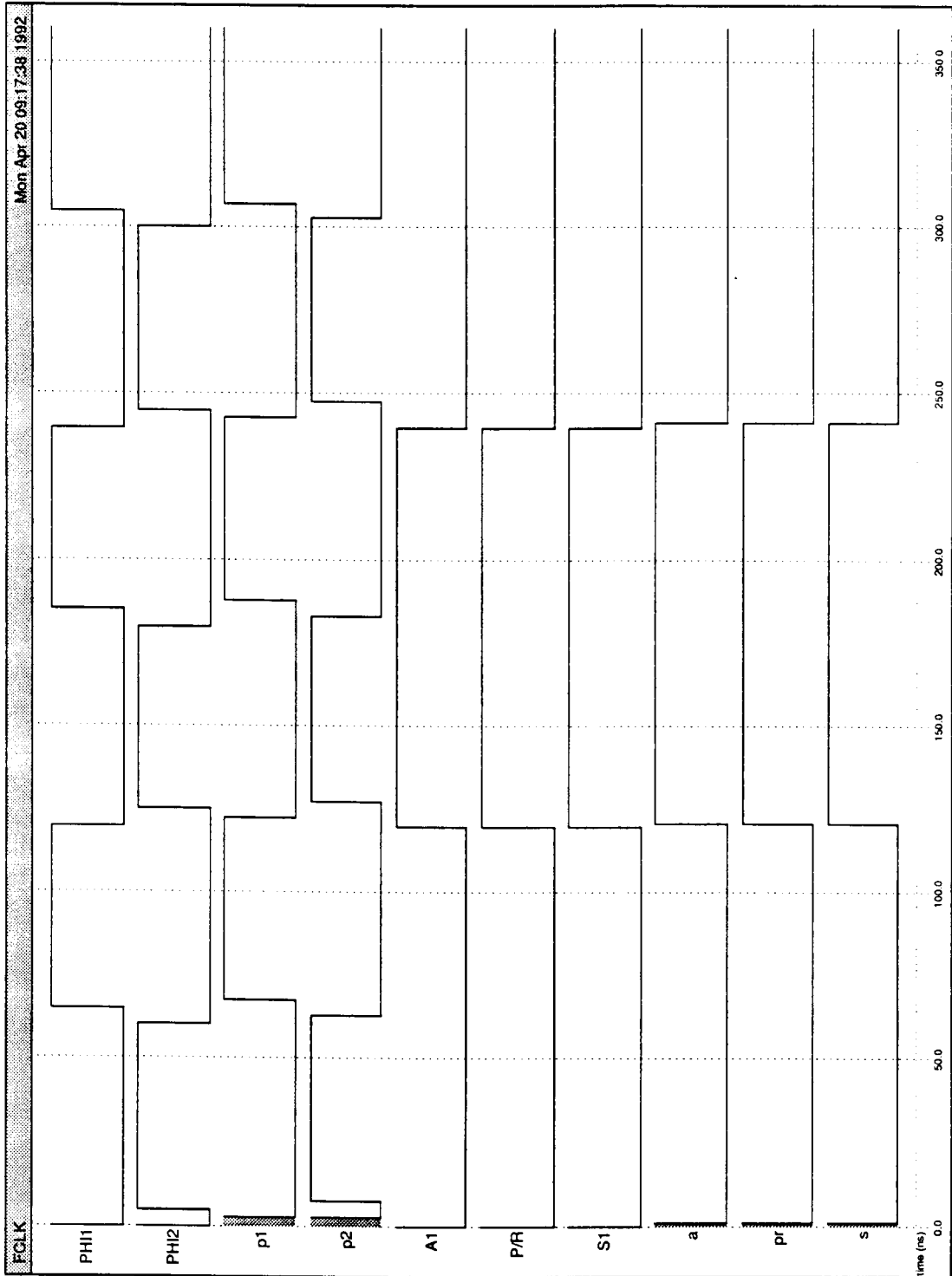
DATE: Apr 20, 1992

Scale: 117 lambda/inch

Filename: FCLK.mag

User: whitted





```
;FORMAT.sdl -- Output formatting block for IP output
;Rodney Whitted, Mar. 1992
```

```
(parent-cell FORMAT)
(layout-generator Flint)
```

```
(subcells
```

```
    (FCLK          FCLK)
    (XLATCH        XLATCH)
    (HSCALE        HSCALE)
    (SHIFTS        SHIFTS)
)
```

```
(instance FCLK(
```

```
    (S1            S1)
    (S1INV         S1INV)
    (A1            A1)
    (A1INV         A1INV)
    (P/R           P/R)
    (P/RINV        P/RINV)
```

```
    (PHI1          PHI1)
    (PHI1INV       PHI1INV)
    (PHI2          PHI2)
    (PHI2INV       PHI2INV)
```

```
    (CLKOUT        CLKOUT (width 18))
```

```
    (Vdd           Vdd)
    (GND           GND)
```

```
))
```

```
(instance XLATCH(
```

```
    (XIN           XIN    (width 8))
    (XOUT          XOUT   (width 8))
```

```
    (S1            CLKOUT[4])
    (S1INV         CLKOUT[5])
    (PHI2          CLKOUT[8])
    (PHI2INV       CLKOUT[9])
    (PHI1          CLKOUT[6])
    (PHI1INV       CLKOUT[7])
```

```
    (Vdd           Vdd)
    (GND           GND)
```

```
))
```

```
(instance HSCALE(
```

```
    (HIN           HIN    (width 8))
    (CONTROL       control (width 8))
```

```
    (P/R           CLKOUT[2])
    (P/RINV        CLKOUT[3])
    (PHI2          CLKOUT[12])
    (PHI2INV       CLKOUT[13])
    (PHI1          CLKOUT[10])
    (PHI1INV       CLKOUT[11])
```

```

        (Vdd          Vdd)
        (GND          GND)
    ))

(instance SHIFTS(
    (ACCIN            ACCIN (width 13))
    (ACCOUNT        ACCOUT (width 13))
    (SELNET           control (width 8))

    (A1               CLKOUT[0])
    (A1INV            CLKOUT[1])
    (PHI2             CLKOUT[16])
    (PHI2INV          CLKOUT[17])
    (PHI1             CLKOUT[14])
    (PHI1INV          CLKOUT[15])

    (Vdd              Vdd)
    (GND              GND)
))

(instance parent(
    ((terminal        ACCIN (TERM_EDGE LEFT))    ACCIN (width 13))
    ((terminal        ACCOUT (TERM_EDGE RIGHT))  ACCOUT (width 13))
    ((terminal        HIN (TERM_EDGE LEFT))      HIN (width 8))
    ((terminal        XIN (TERM_EDGE LEFT))      XIN (width 8))
    ((terminal        XOUT (TERM_EDGE RIGHT))    XOUT (width 8))

    ((terminal        P/R (TERM_EDGE TOP))       P/R)
    ((terminal        P/RINV (TERM_EDGE TOP))    P/RINV)
    ((terminal        S1 (TERM_EDGE TOP))        S1)
    ((terminal        S1INV (TERM_EDGE TOP))     S1INV)
    ((terminal        A1 (TERM_EDGE TOP))        A1)
    ((terminal        A1INV (TERM_EDGE TOP))     A1INV)
    ((terminal        PHI1 (TERM_EDGE TOP))      PHI1)
    ((terminal        PHI1INV (TERM_EDGE TOP))   PHI1INV)
    ((terminal        PHI2 (TERM_EDGE TOP))      PHI2)
    ((terminal        PHI2INV (TERM_EDGE TOP))   PHI2INV)

    ((terminal        Vdd (TERMTYPE SUPPLY))     Vdd)
    ((terminal        GND (TERMTYPE GROUND))     GND)
))

(end-sdl)

```

```
!FORMAT.cmds -- sim file for the output format block of the IP
!Rodney Whitted, Mar. 1992
```

```
h      Vdd
l      GND
```

```
stepsize 5
```

```
clock PHI1  0000000000000111111111111
clock PHI1INV 1111111111111000000000000
clock PHI2  0111111111111000000000000
clock PHI2INV 1000000000000111111111111
```

```
vector ACCIN ACCIN[{12:0}]
vector ACCOUT ACCOUT[{12:0}]
vector HIN    HIN[{7:0}]
vector XIN    XIN[{7:0}]
vector XOUT   XOUT[{7:0}]
```

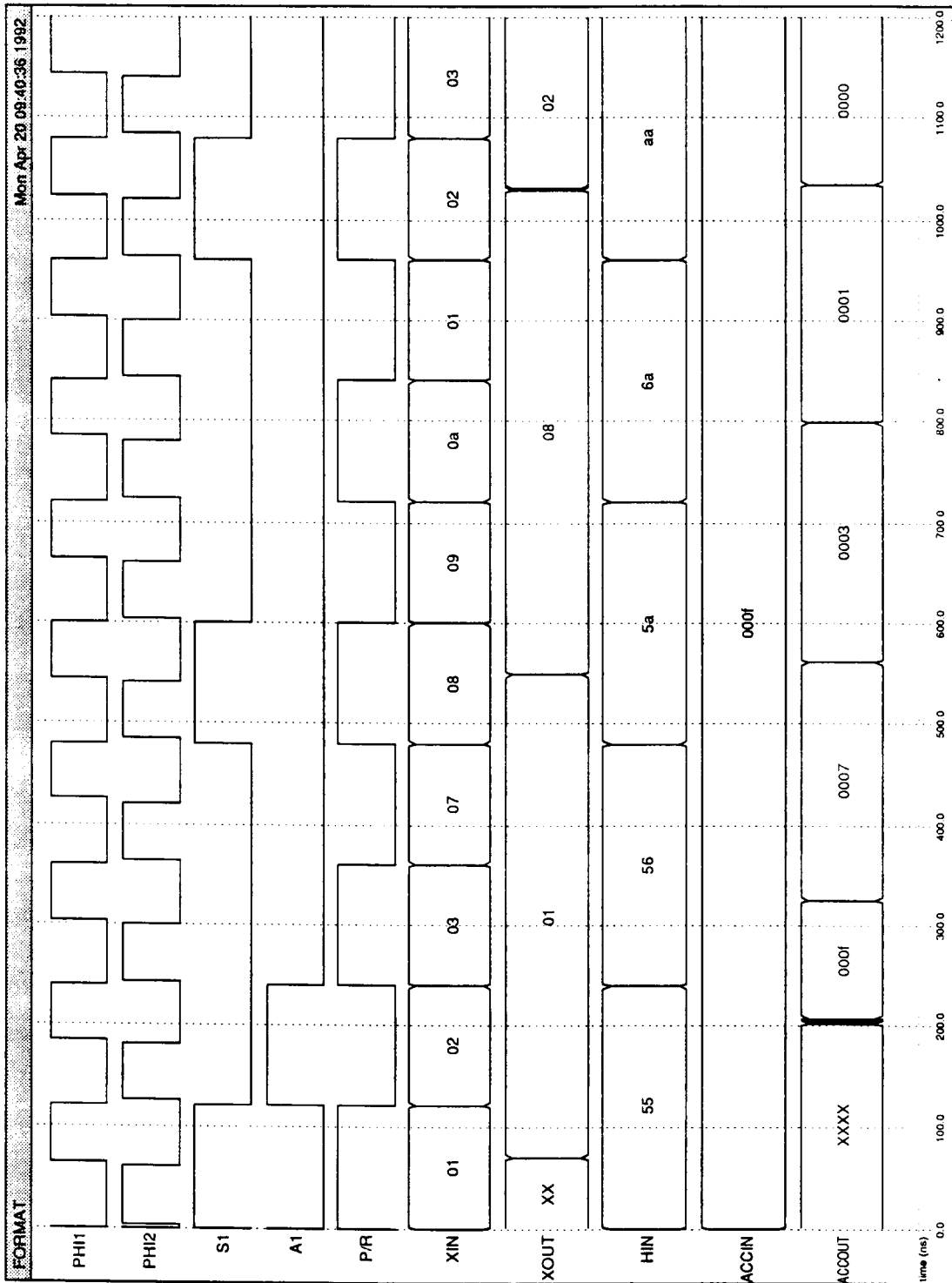
```
V      ACCIN  0000000001111
V      XIN    00000001 00000010 00000011 00000111 00001000 00001001 00001010
V      HIN    01010101 xxxxxxxx 01010110 xxxxxxxx 01011010 xxxxxxxx 01101010 xxxxxxxx
10101010 xxxxxxxx
```

```
V      P/R    1010101010
V      P/RINV 0101010101
V      S1     1000100010
V      S1INV  0111011101
V      A1     0100000000
V      A1INV  1011111111
```

```
analyzer PHI1 PHI2 S1 A1 P/R XIN XOUT HIN ACCIN ACCOUT
```

```
R
```

UNIVERSITY OF TENNESSEE	
Electrical and Computer Engineering	
DATE: May 1, 1992	Scale: 322 lambda/inch
Filename: CORE FORMAT.mag	User: whitted



```
;GCLK.sdl -- clock tree for driving GSTATE.sdl section of A_PIPE
;Rodney Whitted, Mar. 1992
```

```
(parent-cell GCLK)
```

```
(layout-generator Flint)
```

```
(subcells
```

```
  (CBA   CB1((N 4)))
```

```
  (CBA   CB2((N 8)))
```

```
  (CBA   CB3((N 24)))
```

```
)
```

```
(instance CB1(
```

```
  (IN clkin1  (width 4))
```

```
  (Vdd Vdd)
```

```
  (GND  GND)
```

```
))
```

```
(net clkin1  ((parent S1  (net-base 0) (net-incr 1))))
```

```
(net clkin1  ((parent S1INV (net-base 1) (net-incr 1))))
```

```
(net clkin1  ((parent S2  (net-base 2) (net-incr 1))))
```

```
(net clkin1  ((parent S2INV (net-base 3) (net-incr 1))))
```

```
(net clk1 (NETWIDTH 4)  ((CB1 OUT)))
```

```
(instance CB2(
```

```
  (Vdd Vdd)
```

```
  (GND  GND)
```

```
))
```

```
(net clk1 ((CB2 IN (width 2) (term-base 0) (term-incr 4) (net-base 0) (net-incr 0))))
```

```
(net clk1 ((CB2 IN (width 2) (term-base 1) (term-incr 4) (net-base 1) (net-incr 0))))
```

```
(net clk1 ((CB2 IN (width 2) (term-base 2) (term-incr 4) (net-base 2) (net-incr 0))))
```

```
(net clk1 ((CB2 IN (width 2) (term-base 3) (term-incr 4) (net-base 3) (net-incr 0))))
```

```
(net clk2 (NETWIDTH 8)  ((CB2 OUT)))
```

```
(instance CB3(
```

```
  (OUT  CLKOUT (width 16))
```

```
  (Vdd Vdd)
```

```
  (GND  GND)
```

```
))
```

```
(net clk2 ((CB3 IN (width 3) (term-base 0) (term-incr 8) (net-base 0) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 3) (term-base 1) (term-incr 8) (net-base 1) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 3) (term-base 2) (term-incr 8) (net-base 2) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 3) (term-base 3) (term-incr 8) (net-base 3) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 3) (term-base 4) (term-incr 8) (net-base 4) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 3) (term-base 5) (term-incr 8) (net-base 5) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 3) (term-base 6) (term-incr 8) (net-base 6) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 3) (term-base 7) (term-incr 8) (net-base 7) (net-incr 0))))
```

```
(instance parent(
```

```
  ((terminal CLKOUT  (TERMTYPE CLOCK)) CLKOUT (width 16))
```

```
  ((terminal Vdd      (TERMTYPE SUPPLY)) Vdd)
```

```
  ((terminal GND      (TERMTYPE GROUND))  GND)
```

```
))
```

```
(end-sdl)
```

```
;GSTATE.sdl -- g-input vector state memory
;Rodney Whitted, Feb. 1992
```

```
(parent-cell GSTATE)
(parameters (N 8) (delay1 3) (delay2 4))
(structure-processor dpp)
(layout-generator Flint a)
(dotimes (i delay1)
  (subcells
    (scanmslatch L1 ((N N)))
  )
  (instance L1(
    (IN (in1dum (ALIAS (if (= i 0)
      (format nil "GIN")
      (format nil "in1[~D]" i)
    )))
    (OUT (out1dum (ALIAS (if (= i (- delay1 1))
      (format nil "G")
      (format nil "in1[~D]" (+ i 1))
    )))
    (LOAD (lddum1 (ALIAS(format nil "S2[~D]" i))))
    (LOADINV (ldidum1 (ALIAS (format nil "S2INV[~D]" i))))
    (PHI1 (pdum1 (ALIAS (format nil "S1[~D]" i))))
    (PHI1INV (pidum1 (ALIAS (format nil "S1INV[~D]" i))))
    (SHIFT GND)
    (SHIFTINV Vdd)
    (Vdd Vdd)
    (GND GND)
  ))
)

(dotimes (i delay2)
  (subcells
    (scanmslatch L2 ((N N)))
  )
  (instance L2(
    (IN (in2dum (ALIAS (if (= i 0)
      (format nil "G")
      (format nil "in2[~D]" i)
    )))
    (OUT (out2dum (ALIAS (if (= i (- delay2 1))
      (format nil "GOUT")
      (format nil "in2[~D]" (+ i 1))
    )))
    (LOAD (lddum2 (ALIAS(format nil "S2[~D]" (+ i delay1)))))
    (LOADINV (ldidum2 (ALIAS (format nil "S2INV[~D]" (+ i delay1)))))
    (PHI1 (pdum2 (ALIAS (format nil "S1[~D]" (+ i delay1)))))
    (PHI1INV (pidum2 (ALIAS (format nil "S1INV[~D]" (+ i delay1)))))
    (SHIFT CNTL_GND)
    (SHIFTINV CNTL_VDD)
    (Vdd Vdd)
    (GND GND)
  ))
)

(instance parent(
  ((terminal GIN (TERM_EDGE LEFT)) GIN)
```

```

((terminal GOUT (TERM_EDGE RIGHT)) GOUT)
((terminal G (TERM_EDGE LEFT)) G)

((terminal S1 (TERM_EDGE TOP)) S1 (width (+ delay1 delay2)))
((terminal S1INV (TERM_EDGE TOP)) S1INV (width (+ delay1 delay2)))
((terminal S2 (TERM_EDGE TOP)) S2 (width (+ delay1 delay2)))
((terminal S2INV (TERM_EDGE TOP)) S2INV (width (+ delay1 delay2)))

((terminal CNTL_VDD (TERMTYPE SUPPLY)) CNTL_VDD)
((terminal CNTL_GND (TERMTYPE GROUND)) CNTL_GND)
((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
((terminal GND (TERMTYPE GROUND)) GND)
))
(end-sdl)

```

|GSTATE.cmds -- IRSIM command file for testing GSTATE
| Rodney B. Whitted. February 1992

h Vdd
l GND

stepsize 5

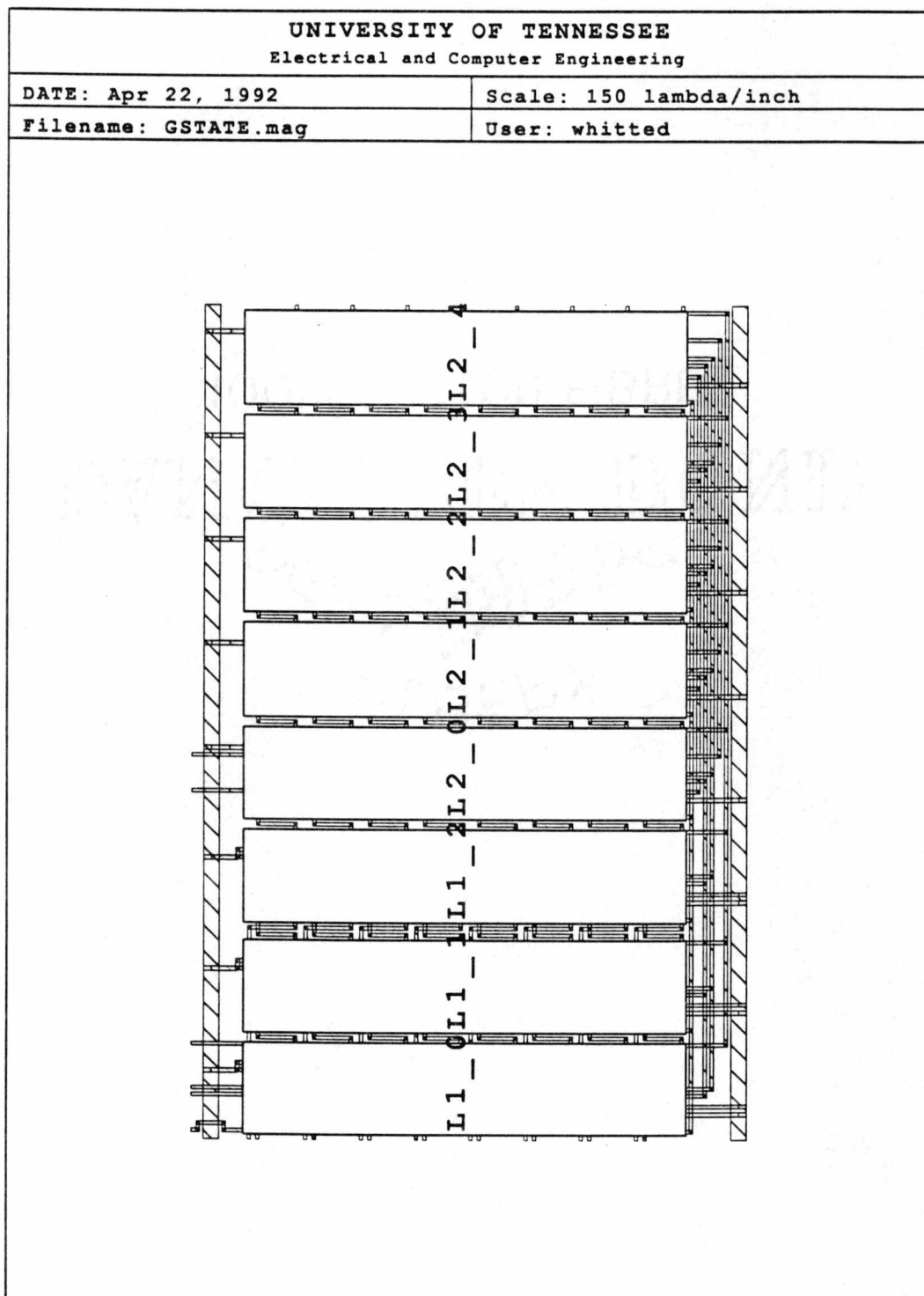
clock S1 0000000000001111111111
clock S1INV 1111111111110000000000
clock S2 0111111111110000000000
clock S2INV 1000000000011111111111

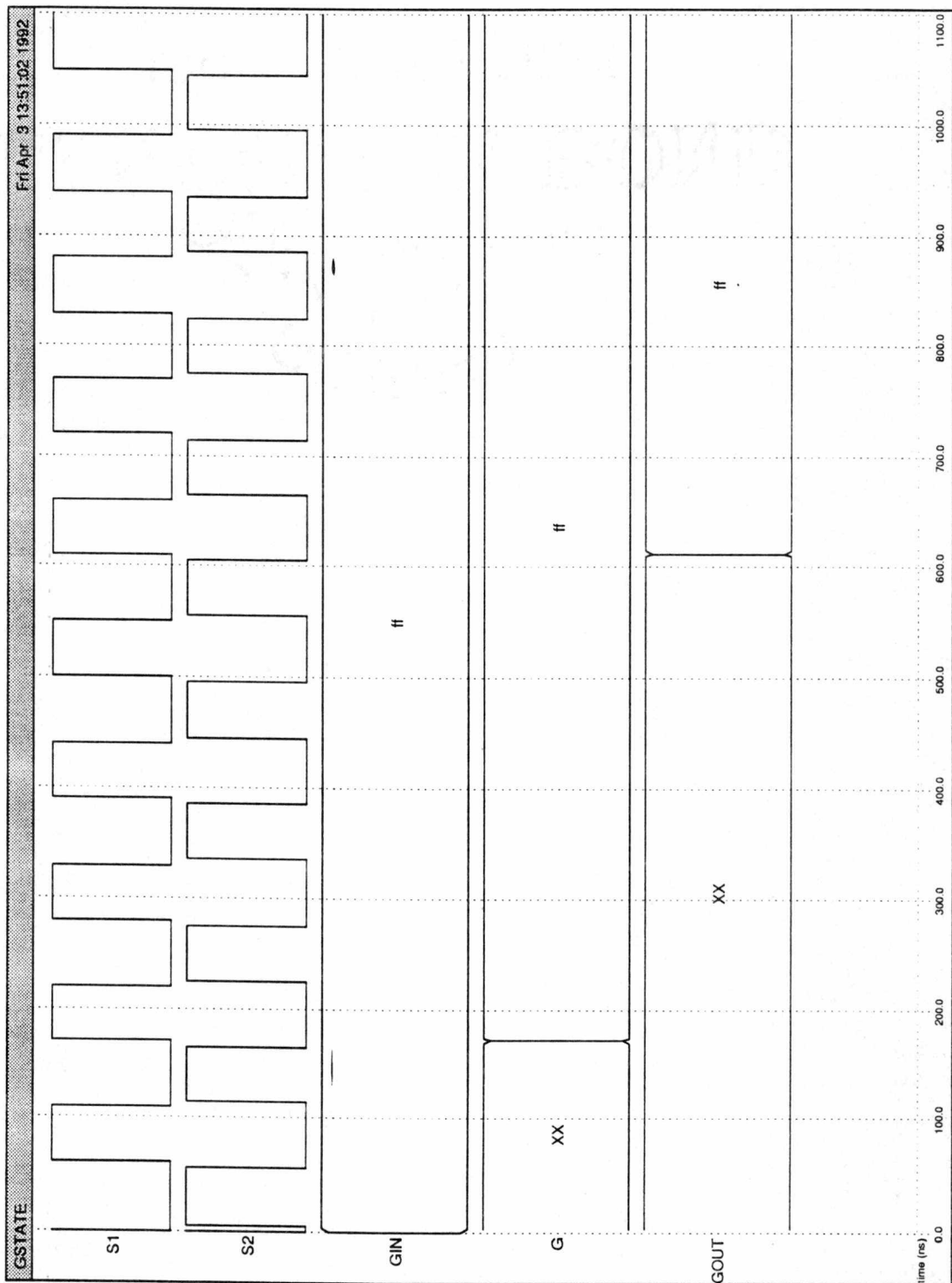
vector GIN GIN[{7:0}]
vector G G[{7:0}]
vector GOUT GOUT[{7:0}]

VGIN 11111111

analyzer S1 S2 GIN G GOUT

R 10





```
;H2CLK.sdl -- clock tree for HSTATE2
;Rodney Whitted, Mar. 1992
```

```
(parent-cell H2CLK)
```

```
(layout-generator Flint)
```

```
(subcells
```

```
    (CBA    CB1    ((N 5)))
    (CBA    CB2    ((N 20)))
    (CBA    CB3    ((N 68)))

```

```
)
```

```
(instance CB1(
```

```
    (IN      clk1n1  (width 4))
```

```
    (Vdd     Vdd)
```

```
    (GND     GND)
```

```
))
```

```
(net clk1n1      ((parent PHI1      (net-base 0) (net-incr 1))))
```

```
(net clk1n1      ((parent PHI1INV (net-base 1) (net-incr 1))))
```

```
(net clk1n1      ((parent PHI2      (net-base 2) (net-incr 1))))
```

```
(net clk1n1      ((parent PHI2INV (net-base 3) (net-incr 1))))
```

```
(net clk1 (NETWIDTH 4)  ((CB1 OUT)))
```

```
(instance CB2(
```

```
    (Vdd     Vdd)
```

```
    (GND     GND)
```

```
))
```

```
(net clk1 ((CB2 IN (width 5) (term-base 0) (term-incr 4) (net-base 0) (net-incr 0))))
```

```
(net clk1 ((CB2 IN (width 5) (term-base 1) (term-incr 4) (net-base 1) (net-incr 0))))
```

```
(net clk1 ((CB2 IN (width 5) (term-base 2) (term-incr 4) (net-base 2) (net-incr 0))))
```

```
(net clk1 ((CB2 IN (width 5) (term-base 3) (term-incr 4) (net-base 3) (net-incr 0))))
```

```
(net clk2 (NETWIDTH 20) ((CB2 OUT)))
```

```
(instance CB3(
```

```
    (OUT      CLKOUT (width 68))
```

```
    (Vdd     Vdd)
```

```
    (GND     GND)
```

```
))
```

```
(net clk2 ((CB3 IN (width 4) (term-base 0) (term-incr 16) (net-base 0) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 4) (term-base 1) (term-incr 16) (net-base 1) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 4) (term-base 2) (term-incr 16) (net-base 2) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 4) (term-base 3) (term-incr 16) (net-base 3) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 4) (term-base 4) (term-incr 16) (net-base 4) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 4) (term-base 5) (term-incr 16) (net-base 5) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 4) (term-base 6) (term-incr 16) (net-base 6) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 4) (term-base 7) (term-incr 16) (net-base 7) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 4) (term-base 8) (term-incr 16) (net-base 8) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 4) (term-base 9) (term-incr 16) (net-base 9) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 4) (term-base 10) (term-incr 16) (net-base 10) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 4) (term-base 11) (term-incr 16) (net-base 11) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 4) (term-base 12) (term-incr 16) (net-base 12) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 4) (term-base 13) (term-incr 16) (net-base 13) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 4) (term-base 14) (term-incr 16) (net-base 14) (net-incr 0))))
```

```

(net clk2 ((CB3 IN (width 4) (term-base 15) (term-incr 16) (net-base 15) (net-incr 0))))
(net clk2 ((CB3 IN (width 1) (term-base 64) (term-incr 1) (net-base 16) (net-incr 0))))
(net clk2 ((CB3 IN (width 1) (term-base 65) (term-incr 1) (net-base 17) (net-incr 0))))
(net clk2 ((CB3 IN (width 1) (term-base 66) (term-incr 1) (net-base 18) (net-incr 0))))
(net clk2 ((CB3 IN (width 1) (term-base 67) (term-incr 1) (net-base 19) (net-incr 0))))

(instance parent(
  ((terminal CLKOUT      (TERMTYPE CLOCK))      CLKOUT (width 68))

  ((terminal Vdd         (TERMTYPE SUPPLY))      Vdd)
  ((terminal GND         (TERMTYPE GROUND))      GND)
))

(end-sdl)

```

!HCLK.cmds -- commands to test HSTATE's clock tree
!Rodney Whitted, Mar. 1992

h Vdd
l GND

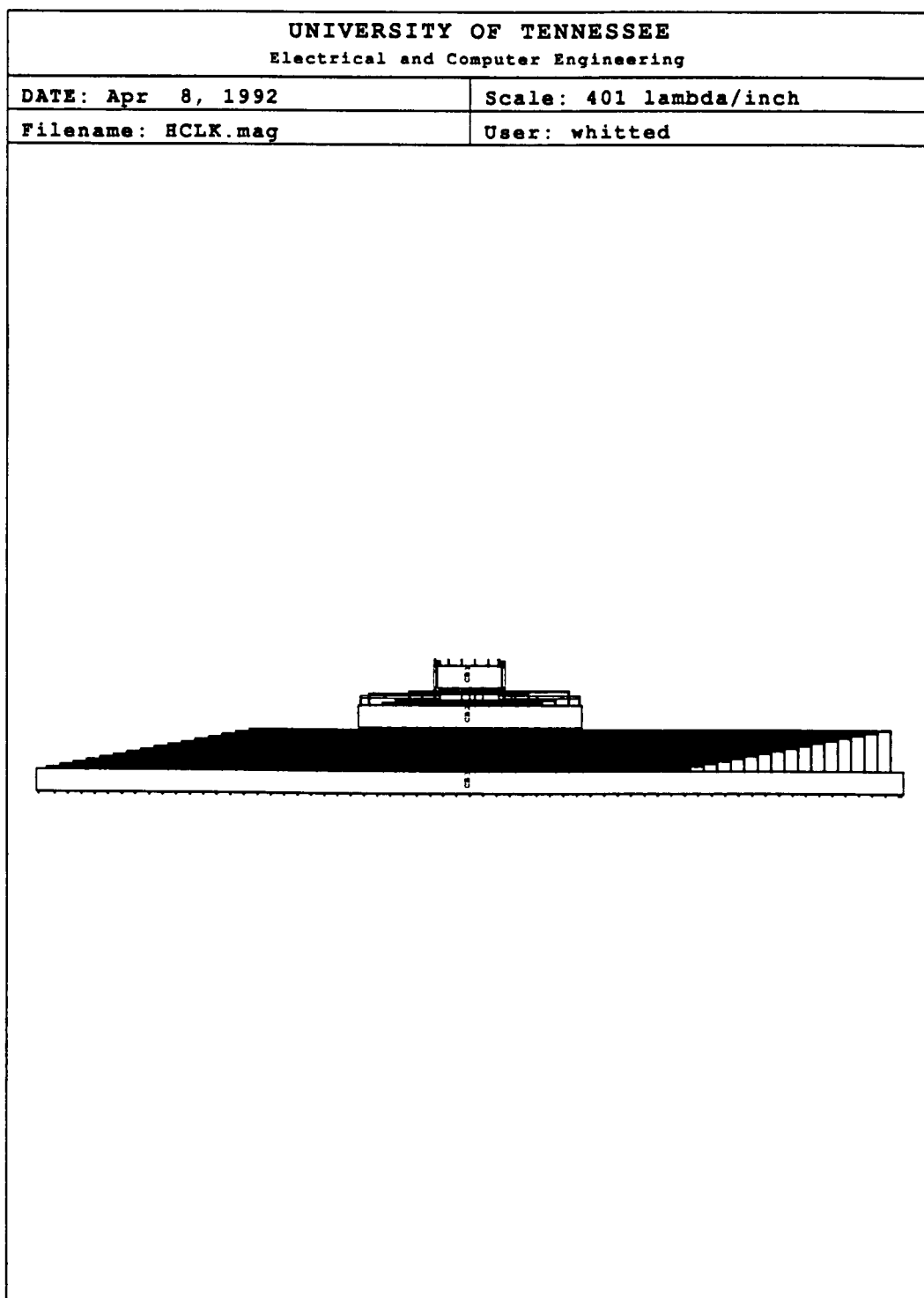
stepsize 5

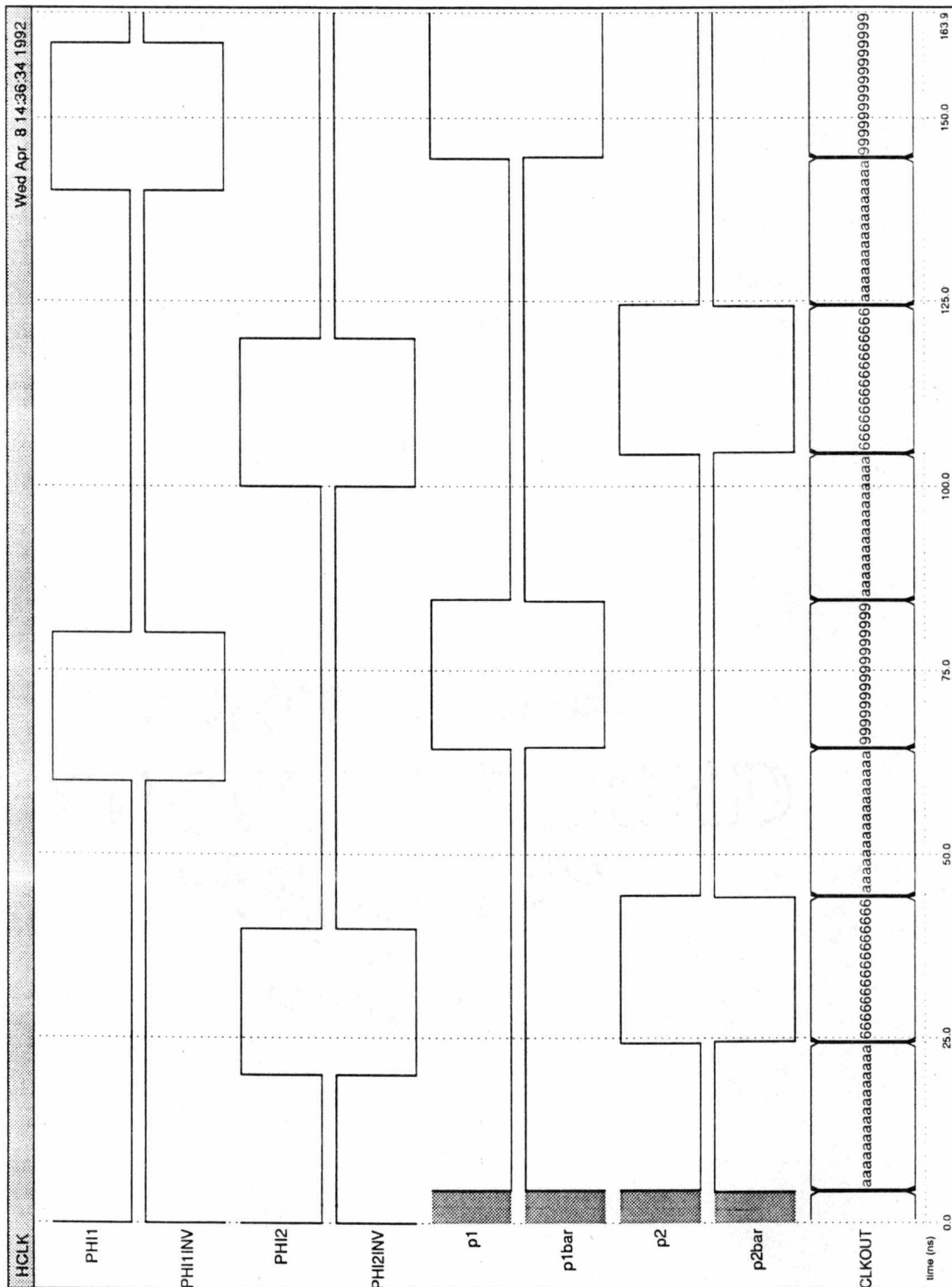
clock PHI1 000000000000011111111111
clock PHI1INV 11111111111111000000000000
clock PHI2 0111111111111100000000000000
clock PHI2INV 10000000000001111111111111

vector CLKOUT CLKOUT[{63:0}]
vector p1 CLKOUT[0]
vector p1bar CLKOUT[1]
vector p2 CLKOUT[2]
vector p2bar CLKOUT[3]

analyzer PHI1 PHI1INV PHI2 PHI2INV p1 p1bar p2 p2bar CLKOUT

c 5





```
;HSCALE.sdl -- h scale factor memory for use in FORMAT.sdl
;Rodney Whitted, Mar. 1992
```

```
(parent-cell HSCALE)
(parameters (N 8))
(structure-processor dpp)
(layout-generator Flint a)
(subcells
  (mux2to1      MUX ((N N)))
  (scanmslatch HL  ((N N)))
)
(instance MUX(
  (IN1      HIN)
  (IN2      CONTROL)
  (OUT      hlin)

  (SEL1     P/R)
  (SEL2     P/RINV)

  (Vdd      Vdd)
  (GND      GND)
))
(instance HL(
  (IN      hlin)
  (OUT     CONTROL)

  (LOAD     PHI2)
  (LOADINV  PHI2INV)
  (PHI1     PHI1)
  (PHI1INV  PHI1INV)

  (SHIFT   GND)
  (SHIFTINV Vdd)
  (Vdd     Vdd)
  (GND     GND)
))
(instance parent(
  ((terminal HIN (TERM_EDGE LEFT)) HIN)
  ((terminal CONTROL (TERM_EDGE RIGHT)) CONTROL)

  ((terminal P/R (TERM_EDGE TOP)) P/R)
  ((terminal P/RINV (TERM_EDGE TOP)) P/RINV)

  ((terminal PHI1 (TERM_EDGE TOP)) PHI1)
  ((terminal PHI1INV (TERM_EDGE TOP)) PHI1INV)
  ((terminal PHI2 (TERM_EDGE TOP)) PHI2)
  ((terminal PHI2INV (TERM_EDGE TOP)) PHI2INV)

  ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
  ((terminal GND (TERMTYPE GROUND)) GND)
))
(end-sdl)
```

HSCALE.cmds -- simulation command file for HSCALE.sdl
Rodney Whitted, Mar. 1992

h Vdd
l GND

!** 8.333 MHz simulation **
stepsize 5

clock PHI1 000000000000011111111111
clock PHI1INV 111111111111100000000000
clock PHI2 111111111110000000000000
clock PHI2INV 000000000001111111111111

vector HIN HIN[{7:0}]
vector CONTROL CONTROL[{7:0}]

V HIN 00000001 00000010 00000011 00000100 00000101 00000110 00000111 00001000
00001001

V P/R 101000100

V P/RINV 010111011

analyzer PHI1 PHI2 P/R HIN CONTROL

R

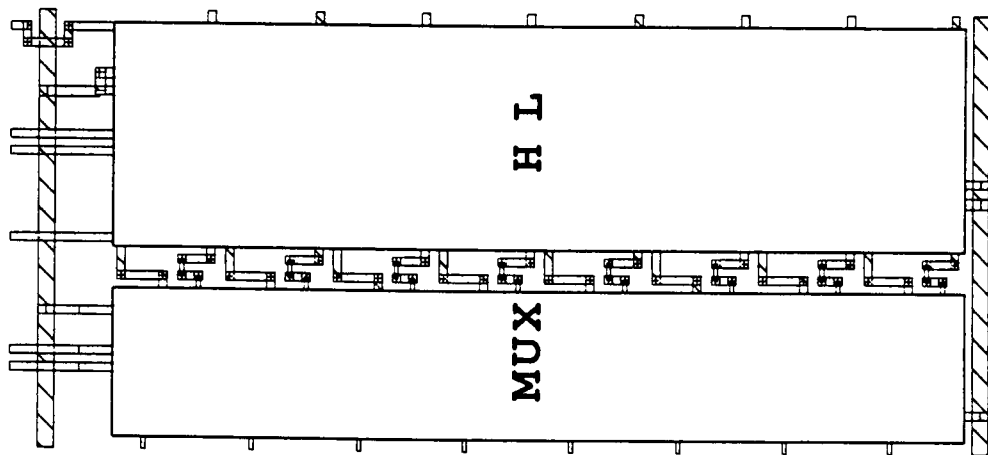
UNIVERSITY OF TENNESSEE
Electrical and Computer Engineering

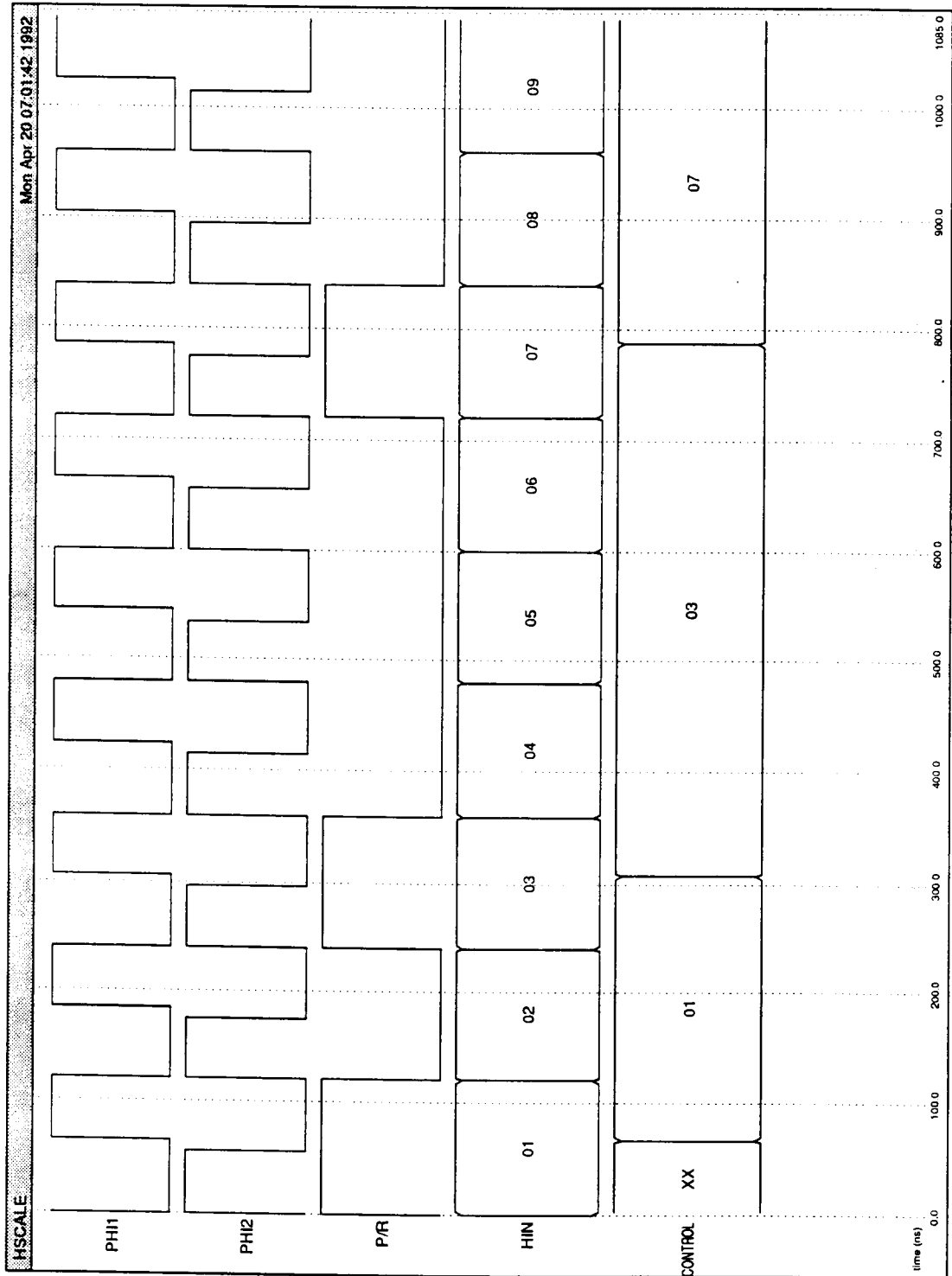
DATE: Apr 20, 1992

Scale: 71 lambda/inch

Filename: HSCALE.mag

User: whitted





```
;HSTATE2.sdl -- coefficient vector state memory used in IP
;Rodney Whitted, Feb. 1992
```

```
(parent-cell HSTATE2)
(parameters (N 8) (delay 16))
(structure-processor dpp)
(layout-generator Flint a)
(subcells
  (mux2to1      (HMUX XMUX) ((N N)))
  (scanmslatch (L_IN XL)    ((N N)))
)

(instance HMUX(
  (IN1      HIN)
  (IN2      HOUT)
  (OUT      muxout)
  (SEL1     P/R)
  (SEL2     P/RINV)
  (Vdd      Vdd)
  (GND      GND)
))

(instance L_IN(
  (IN      muxout)
  (OUT     H)
  (LOAD     PHI2[0])
  (LOADINV  PHI2INV[0])
  (PHI1     PHI1[0])
  (PHI1INV  PHI1INV[0])
  (SHIFT    CNTL_GND)
  (SHIFTINV CNTL_VDD)
  (Vdd      Vdd)
  (GND      GND)
))

(dotimes (i (- delay 1))
  (subcells
    (scanmslatch L ((N N)))
  )
  (instance L(
    (IN (indum (ALIAS (if (= i 0)
                      (format nil "H")
                      (format nil "in[~D]" i)
                    )))
    (OUT (outdum (ALIAS (if (= i (- delay 2))
                          (format nil "HOUT")
                          (format nil "in[~D]" (+ i 1))
                        )))
    (LOAD (ld (ALIAS (format nil "PHI2[~D]" (+ i 1)))))
    (LOADINV (ldi (ALIAS (format nil "PHI2INV[~D]" (+ i 1)))))
    (PHI1 (p1 (ALIAS (format nil "PHI1[~D]" (+ i 1)))))
    (PHI1INV (pli (ALIAS (format nil "PHI1INV[~D]" (+ i 1)))))
    (SHIFT CNTL_GND)
    (SHIFTINV CNTL_VDD)
    (Vdd Vdd)
    (GND GND)
  ))
)
```

```

)

(instance XMUX(
    (IN1          XCHAININ)
    (IN2          XCHAINOUT)
    (OUT          chainmuxout)
    (SEL1         S1)
    (SEL2         S1INV)
    (Vdd          Vdd)
    (GND          GND)
))

(instance XL(
    (IN          chainmuxout)
    (OUT         XCHAINOUT)
    (LOAD        (ld2 (ALIAS (format nil "PHI2[~D]" delay))))
    (LOADINV     (ldi2 (ALIAS (format nil "PHI2INV[~D]" delay))))
    (PHI1        (p12 (ALIAS (format nil "PHI1[~D]" delay))))
    (PHI1INV     (pli2 (ALIAS (format nil "PHI1INV[~D]" delay))))
    (SHIFT       CNTL_GND)
    (SHIFTINV    CNTL_VDD)
    (Vdd         Vdd)
    (GND         GND)
))

(instance parent(
    ((terminal    HIN      (TERM_EDGE LEFT))    HIN)
    ((terminal    HOUT    (TERM_EDGE RIGHT))    HOUT)
    ((terminal    H       (TERM_EDGE LEFT))    H)

    ((terminal    XCHAININ (TERM_EDGE LEFT))    XCHAININ)
    ((terminal    XCHAINOUT (TERM_EDGE RIGHT))    XCHAINOUT)

    ((terminal    P/R     (TERM_EDGE TOP))    P/R)
    ((terminal    P/RINV  (TERM_EDGE TOP))    P/RINV)

    ((terminal    S1      (TERM_EDGE TOP))    S1)
    ((terminal    S1INV   (TERM_EDGE TOP))    S1INV)

    ((terminal    PHI1    (TERM_EDGE TOP))    PHI1 (width (+ delay 1)))
    ((terminal    PHI1INV (TERM_EDGE TOP))    PHI1INV (width (+ delay 1)))
    ((terminal    PHI2    (TERM_EDGE TOP))    PHI2 (width (+ delay 1)))
    ((terminal    PHI2INV (TERM_EDGE TOP))    PHI2INV (width (+ delay 1)))

    ((terminal    CNTL_VDD (TERMTYPE SUPPLY))    CNTL_VDD)
    ((terminal    CNTL_GND (TERMTYPE GROUND))    CNTL_GND)
    ((terminal    Vdd     (TERMTYPE SUPPLY))    Vdd)
    ((terminal    GND     (TERMTYPE GROUND))    GND)
))
(end-sdl)

```

| HSTATE2.cmds -- IRSIM command file for testing HSTATE2
 | Rodney B. Whitted, March 1992

h Vdd CNTL_VDD
 l GND CNTL_GND

! ** 8.333MHz simulation **
 stepsize 5

vector PHI1 PHI1[{16:0}]
 vector PHI1INV PHI1INV[{16:0}]
 vector PHI2 PHI2[{16:0}]
 vector PHI2INV PHI2INV[{16:0}]

clock PHI1 0000000000000000 0000000000000000 0000000000000000 0000000000000000
 0000000000000000 0000000000000000 0000000000000000 0000000000000000
 0000000000000000 0000000000000000 0000000000000000 0000000000000000
 0000000000000000 1111111111111111 1111111111111111 1111111111111111
 1111111111111111 1111111111111111 1111111111111111 1111111111111111
 1111111111111111 1111111111111111 1111111111111111 1111111111111111

clock PHI1INV 1111111111111111 1111111111111111 1111111111111111 1111111111111111
 1111111111111111 1111111111111111 1111111111111111 1111111111111111
 1111111111111111 1111111111111111 1111111111111111 1111111111111111
 1111111111111111 0000000000000000 0000000000000000 0000000000000000
 0000000000000000 0000000000000000 0000000000000000 0000000000000000
 0000000000000000 0000000000000000 0000000000000000 0000000000000000

clock PHI2 0000000000000000 1111111111111111 1111111111111111 1111111111111111
 1111111111111111 1111111111111111 1111111111111111 1111111111111111
 1111111111111111 1111111111111111 1111111111111111 1111111111111111
 0000000000000000 0000000000000000 0000000000000000 0000000000000000
 0000000000000000 0000000000000000 0000000000000000 0000000000000000
 0000000000000000 0000000000000000 0000000000000000 0000000000000000

clock PHI2INV 1111111111111111 0000000000000000 0000000000000000 0000000000000000
 0000000000000000 0000000000000000 0000000000000000 0000000000000000
 0000000000000000 0000000000000000 0000000000000000 0000000000000000
 1111111111111111 1111111111111111 1111111111111111 1111111111111111
 1111111111111111 1111111111111111 1111111111111111 1111111111111111
 1111111111111111 1111111111111111 1111111111111111 1111111111111111

vector HIN HIN[{7:0}]
 vector H H[{7:0}]
 vector HOUT HOUT[{7:0}]

vector XCHAININ XCHAININ[{7:0}]
 vector XCHAINOUT XCHAINOUT[{7:0}]

VHIN 11111111

VXCHAININ 00000001 00000010 00000011 00000100 00000101 00000110 00000111 00001000
 00001001 00001010 00001011 00001100 00001101 00001110 00001111 00010000 00010001 00010010
 00010011 00010100

VP/R 11111111111111110000

V P/RINV 000000000000000001111

V S1 10001000100010001000

V S1INV 01110111011101110111

analyzer PHI1 PHI2 P/R S1 HIN H HOUT XCHAININ XCHAINOUT

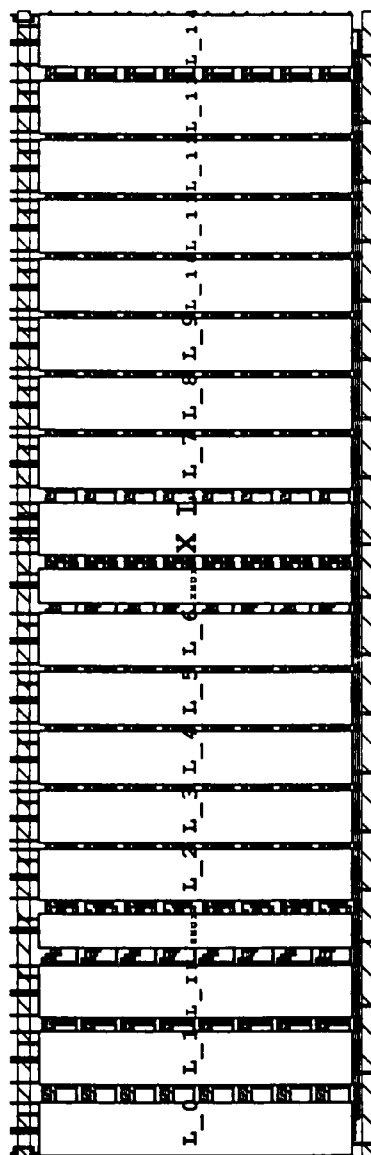
R

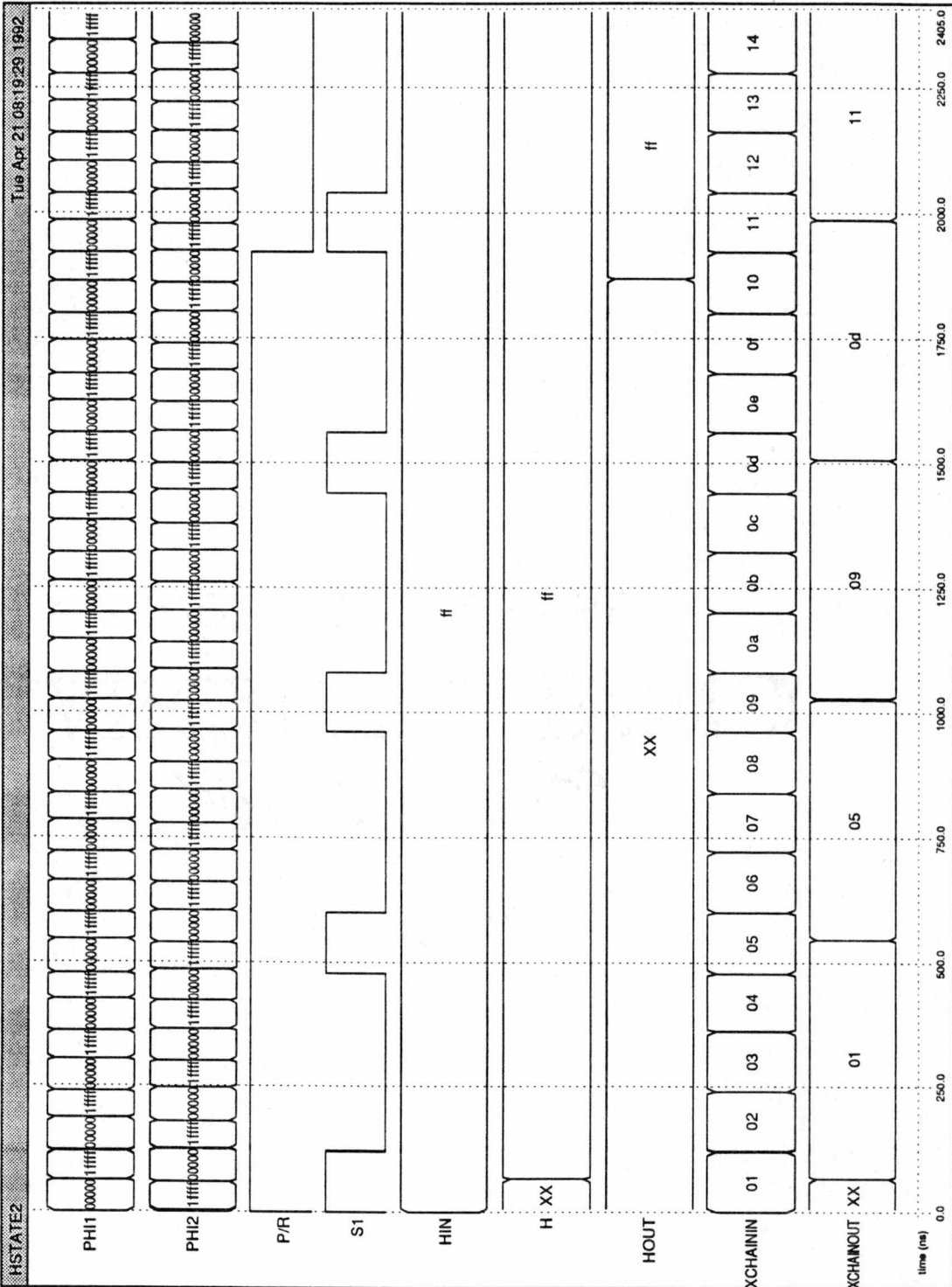
UNIVERSITY OF TENNESSEE
Electrical and Computer Engineering

DATE: Apr 22, 1992	Scale: 300 lambda/inch
Filename: ESTATE2.mag	User: whitted

Scale: 300 lambda/inch

User: whitted





```
;IP2.sdl -- Iteration Processor for FIR filter array
;Rodney Whitted, Feb. 1992
```

```
(parent-cell IP2)
(layout-generator Flint)
(subcells
```

```
    (XCLK      XCLK)
    (XSTATE    XSTATE)
    (H2CLK     H2CLK)
    (HSTATE2   HSTATE2)
    (MAC       MAC)
    (CBA       CB ((N 10)))
```

```
)
```

```
(instance CB(
  (IN      clk_in  (width 10))
  (OUT     clk_out (width 10))

  (Vdd     Vdd)
  (GND     GND)
))
```

```
(net clk_in ((parent A1      (net-base 0) (net-incr 1)))
(net clk_in ((parent A1INV   (net-base 1) (net-incr 1)))
(net clk_in ((parent S1     (net-base 2) (net-incr 1)))
(net clk_in ((parent S1INV  (net-base 3) (net-incr 1)))
(net clk_in ((parent P/R    (net-base 4) (net-incr 1)))
(net clk_in ((parent P/RINV (net-base 5) (net-incr 1)))
(net clk_in ((parent PHI1   (net-base 6) (net-incr 1)))
(net clk_in ((parent PHI1INV (net-base 7) (net-incr 1)))
(net clk_in ((parent PHI2   (net-base 8) (net-incr 1)))
(net clk_in ((parent PHI2INV (net-base 9) (net-incr 1)))
```

```
(instance XCLK(
  (PHI1      clkout[6])
  (PHI1INV   clkout[7])
  (PHI2      clkout[8])
  (PHI2INV   clkout[9])

  (Vdd       Vdd)
  (GND       GND)
))
```

```
(net xphi1 (NETWIDTH 17) ((XCLK CLKOUT (term-base 0) (term-incr 4)))
(net xphi1inv (NETWIDTH 17) ((XCLK CLKOUT (term-base 1) (term-incr 4)))
(net xphi2 (NETWIDTH 17) ((XCLK CLKOUT (term-base 2) (term-incr 4)))
(net xphi2inv (NETWIDTH 17) ((XCLK CLKOUT (term-base 3) (term-incr 4)))
```

```
(instance XSTATE(
  (XIN      XIN      (width 8))
  (X        x        (width 8))
  (XOUT     XOUT     (width 8))

  (PHI2     xphi2    (width 17))
  (PHI2INV  xphi2inv (width 17))
  (PHI1     xphi1    (width 17))
  (PHI1INV  xphi1inv (width 17))
```

```

        (S1          clkout[2])
        (S1INV       clkout[3])

        (CNTL_VDD    Vdd)
        (CNTL_GND    GND)
        (Vdd         Vdd)
        (GND         GND)
    ))

(instance H2CLK(
    (PHI1          clkout[6])
    (PHI1INV       clkout[7])
    (PHI2          clkout[8])
    (PHI2INV       clkout[9])

    (Vdd          Vdd)
    (GND          GND)
))

(net hphi1 (NETWIDTH 17) ((H2CLK CLKOUT (term-base 0) (term-incr 4))))
(net hphi1inv (NETWIDTH 17) ((H2CLK CLKOUT (term-base 1) (term-incr 4))))
(net hphi2 (NETWIDTH 17) ((H2CLK CLKOUT (term-base 2) (term-incr 4))))
(net hphi2inv (NETWIDTH 17) ((H2CLK CLKOUT (term-base 3) (term-incr 4))))

(instance HSTATE2(
    (HIN          HIN      (width 8))
    (H            h        (width 8))
    (HOUT         HOUT     (width 8))

    (XCHAININ     XCHAININ (width 8))
    (XCHAINOUT    XCHAINOUT (width 8))

    (PHI2         hphi2   (width 17))
    (PHI2INV      hphi2inv (width 17))
    (PHI1         hphi1   (width 17))
    (PHI1INV      hphi1inv (width 17))
    (S1          clkout[2])
    (S1INV       clkout[3])
    (P/R         clkout[4])
    (P/RINV      clkout[5])

    (CNTL_VDD    Vdd)
    (CNTL_GND    GND)
    (Vdd         Vdd)
    (GND         GND)
))

(instance MAC(
    (ACCIN        ACCIN   (width 20))
    (ACCOUT       ACCOUT  (width 20))

    (PHI1          clkout[6])
    (PHI1INV       clkout[7])
    (PHI2          clkout[8])
    (PHI2INV       clkout[9])
    (A1            clkout[0])
    (A1INV         clkout[1])

```

```

        (Vdd      Vdd)
        (GND      GND)
    ))

    (net h ((MAC H (width 8) (term-base 0) (term-incr 1) (net-base 0) (net-incr 1))))
    (net x ((MAC XEST (width 8) (term-base 0) (term-incr 1) (net-base 0) (net-incr 1))))
    (net h ((parent H (width 8))))
    (net x ((parent XEST (width 8))))

    (instance parent(
        ((terminal      XIN      (TERM_EDGE LEFT))      XIN      (width 8))
        ((terminal      XCHAININ  (TERM_EDGE LEFT))      XCHAININ  (width 8))
        ((terminal      HIN      (TERM_EDGE LEFT))      HIN      (width 8))
        ((terminal      ACCIN  (TERM_EDGE LEFT))      ACCIN  (width 20))

        ((terminal      XOUT  (TERM_EDGE RIGHT))      XOUT  (width 8))
        ((terminal      XCHAINOUT  (TERM_EDGE RIGHT))      XCHAINOUT  (width 8))
        ((terminal      HOUT  (TERM_EDGE RIGHT))      HOUT  (width 8))
        ((terminal      ACCOUT (TERM_EDGE RIGHT))      ACCOUT (width 20))

        ((terminal      Vdd      (TERMTYPE SUPPLY))      Vdd)
        ((terminal      GND      (TERMTYPE GROUND))      GND)
    ))

    (end-sdl)

```

||Pprogram.cmds -- this command file programs the IP's memory.
 |Rodney Whitted, Mar. 1992

h Vdd
 l GND

stepsize 5

clock PHI1 000000000000011111111111
 clock PHI1INV 111111111111100000000000
 clock PHI2 011111111111000000000000
 clock PHI2INV 100000000000011111111111

vector XIN XIN[{7:0}]
 vector XOUT XOUT[{7:0}]
 vector XEST XEST[{7:0}]
 vector HIN HIN[{7:0}]
 vector HOUT HOUT[{7:0}]
 vector H H[{7:0}]
 vector ACCIN ACCIN[{19:0}]
 vector ACCOUT ACCOUT[{19:0}]

V ACCIN 00000000000000000000
 V XIN 00000000

V HIN 00000001 00000010 00000011 00000100 00000101 00000110 00000111 00001000
 00001001 00001010 00001011 00001100 00001101 00001110 00001111 00010000

V P/R 1111111111111111
 V P/RINV 0000000000000000

V S1 1111111111111111
 V S1INV 0000000000000000

V A1 1111111111111111
 V A1INV 0000000000000000

analyzer PHI1 PHI2 S1 A1 HOUT H XOUT XEST ACCIN ACCOUT

R

> IPready

IPrun1.cmds -- This command file simulates the IP after it has been programmed
by running IPprogram.cmds.

Rodney Whitted, Mar. 1992

< IPready

h Vdd
l GND

stepsize 5

clock PHI1 000000000000011111111111
clock PHI1INV 111111111111100000000000
clock PHI2 011111111111000000000000
clock PHI2INV 100000000000011111111111

vector XIN XIN[{7:0}]
vector XOUT XOUT[{7:0}]
vector XEST XEST[{7:0}]
vector HIN HIN[{7:0}]
vector HOUT HOUT[{7:0}]
vector H H[{7:0}]
vector ACCIN ACCIN[{19:0}]
vector ACCOUT ACCOUT[{19:0}]

V ACCIN 00000000000000000000
V HIN XXXXXXXX
V P/R 0
V P/RINV 1

V XIN 00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000001
00000001 00000001 00000001 00000001 00000001 00000001 00000001 00000010 00000010
00000010 00000010 00000010 00000010 00000010 00000010 00000010 00000010 00000010 00000010
00000010 00000010 00000010 00000010 00000011 00000011 00000011 00000011 00000011 00000011
00000011 00000011 00000011 00000011 00000011 00000011 00000011 00000011 00000011 00000011
00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100 00000100
00000100 00000100 00000100 00000100 00000100 00000100

V S1 10000000000000000100000000000000010000000000000000100
0000000000000000

V S1INV 0111111111111111011111111111111011111111111111011
1111111111111111

V A1 010000000000000001000000000000000100000000000000010
0000000000000000

V A1INV 101111111111111101111111111111101111111111111101
1111111111111111

analyzer PHI1 PHI2 S1 A1 HOUT H XOUT XEST ACCIN ACCOUT

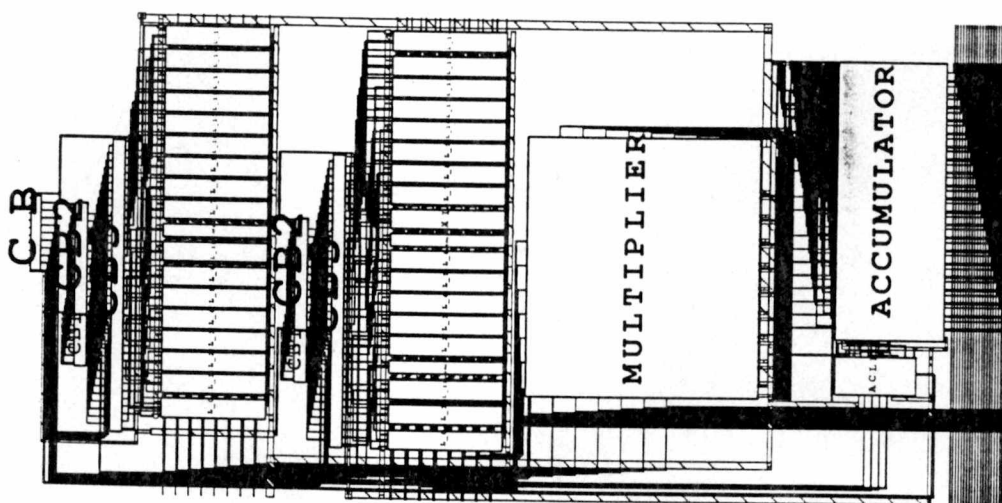
R

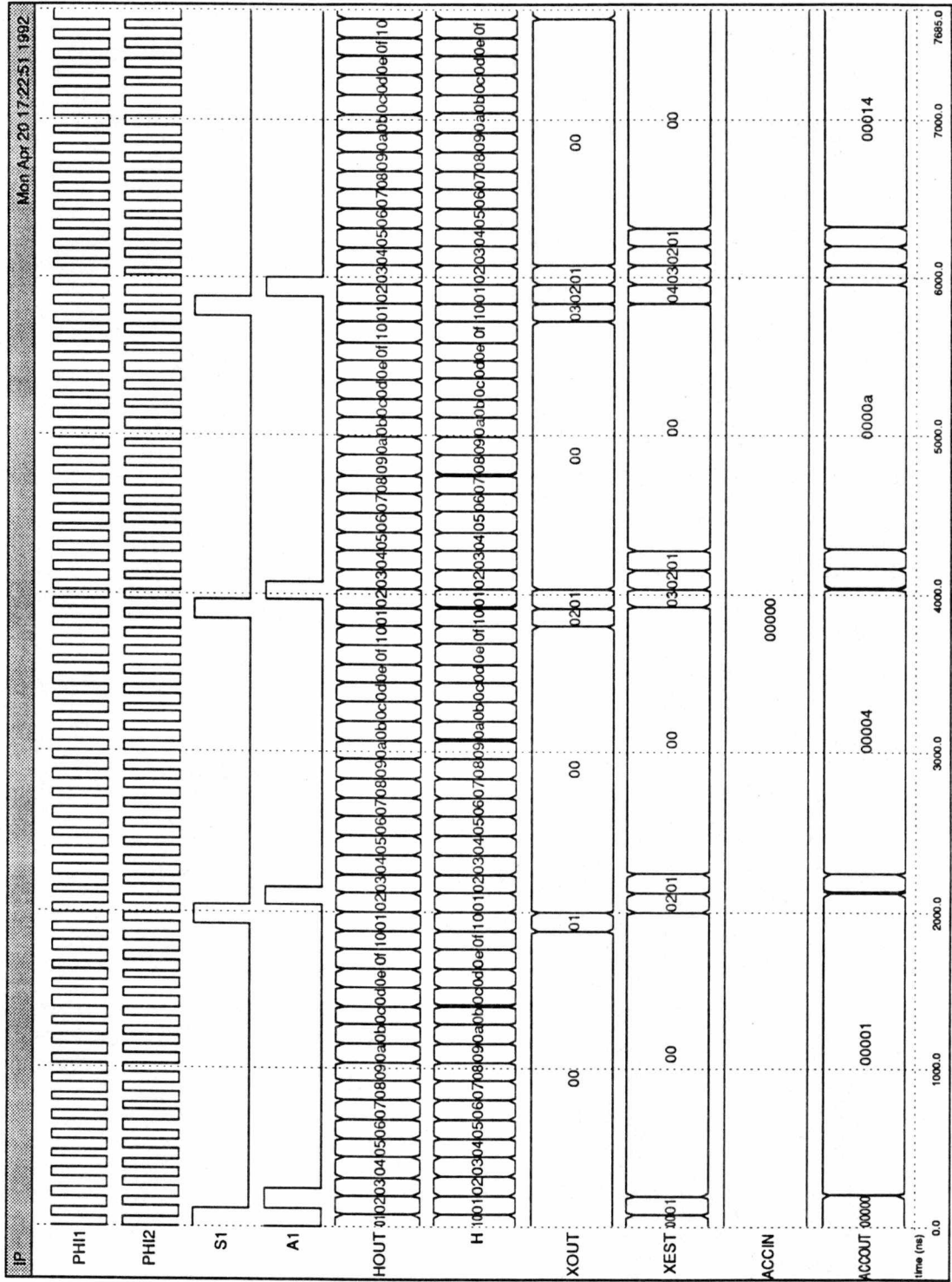
UNIVERSITY OF TENNESSEE
Electrical and Computer Engineering

DATE: May 1, 1992

Scale: 820 lambda/inch

Filename: CORE CONVOLVER IP 1.mg User: whitted





```
;IPARRAY.sdl -- IP array for a 32-point FIR filter (convolver/correlator)
;Rodney Whitted, Mar. 1992
```

```
(parent-cell IPARRAY)
(layout-generator Flint)
(subcells
```

```
    (IP2          (IP_1 IP_2))
    (FORMAT      FORMAT)
)
```

```
(instance IP_1(
    (HIN          HIN    (width 8))
    (HOUT         hout1  (width 8))
    (XIN          XIN    (width 8))
    (XOUT         xout1  (width 8))
    (XCHAININ     XIN    (width 8))
    (XCHAINOUT    xcout1 (width 8))
    (ACCIN        GND    (merge 0 19))
    (ACCOUT       accout1 (width 20))
```

```

    (PHI1         PHI1)
    (PHI1INV      PHI1INV)
    (PHI2         PHI2)
    (PHI2INV      PHI2INV)
    (S1           S1)
    (S1INV        S1INV)
    (A1           A1)
    (A1INV        A1INV)
    (P/R          P/R)
    (P/RINV       P/RINV)

```

```

    (Vdd          Vdd)
    (GND          GND)

```

```
))
```

```
(instance IP_2(
    (HIN          hout1  (width 8))
    (HOUT         hout2  (width 8))
    (XIN          xout1  (width 8))
    (XCHAININ     xcout1 (width 8))
    (XCHAINOUT    xcout2 (width 8))
    (ACCIN        accout1 (width 20))
    (ACCOUT       accout2 (width 20))
```

```

    (PHI1         PHI1)
    (PHI1INV      PHI1INV)
    (PHI2         PHI2)
    (PHI2INV      PHI2INV)
    (S1           S1)
    (S1INV        S1INV)
    (A1           A1)
    (A1INV        A1INV)
    (P/R          P/R)
    (P/RINV       P/RINV)

```

```

    (Vdd          Vdd)
    (GND          GND)

```

```
))
```

```
(instance FORMAT(
  (HIN      hout2 (width 8))
  (XIN      xcout2 (width 8))
  (XOUT     XOUT  (width 8))
  (ACCOUT   ACCOUT(width 13))
```

```
  (PHI1     PHI1)
  (PHI1INV  PHI1INV)
  (PHI2     PHI2)
  (PHI2INV  PHI2INV)
  (S1       S1)
  (S1INV    S1INV)
  (A1       A1)
  (A1INV    A1INV)
  (P/R      P/R)
  (P/RINV   P/RINV)
```

```
  (Vdd      Vdd)
  (GND      GND)
```

```
)
```

```
(net account2 ((FORMAT ACCIN (width 13) (net-base 7) (net-incr 1))))
(net account2 ((parent NC (width 7) (net-base 0) (net-incr 1))))
```

```
(instance parent(
  ((terminal      HIN      (TERM_EDGE LEFT))      HIN      (width 8))
  ((terminal      XIN      (TERM_EDGE LEFT))      XIN      (width 8))
  ((terminal      XOUT     (TERM_EDGE RIGHT))     XOUT     (width 8))
  ((terminal      ACCOUT   (TERM_EDGE RIGHT))     ACCOUT   (width 13))

  ((terminal      PHI1     (TERM_EDGE TOP))      PHI1)
  ((terminal      PHI1INV  (TERM_EDGE TOP))      PHI1INV)
  ((terminal      PHI2     (TERM_EDGE TOP))      PHI2)
  ((terminal      PHI2INV  (TERM_EDGE TOP))      PHI2INV)
  ((terminal      S1       (TERM_EDGE TOP))      S1)
  ((terminal      S1INV    (TERM_EDGE TOP))      S1INV)
  ((terminal      A1       (TERM_EDGE TOP))      A1)
  ((terminal      A1INV    (TERM_EDGE TOP))      A1INV)

  ((terminal      P/R      (TERM_EDGE TOP))      P/R)
  ((terminal      P/RINV   (TERM_EDGE TOP))      P/RINV)

  ((terminal      Vdd      (TERMTYPE SUPPLY))     Vdd)
  ((terminal      GND      (TERMTYPE GROUND))     GND)
```

```
)
```

```
(end-sdl)
```

IPARRAYprogram.cmds -- this command file programs the IPARRAYs vector memories.
|Rodney Whitted, Mar. 1992

h Vdd
l GND CNTL_GND

stepsize 60

clock PHI 0 1

vector XIN XIN[{7:0}]
vector XOUT XOUT[{7:0}]
vector HIN HIN[{7:0}]
vector hout2 hout2[{7:0}]
vector ACCOUT ACCOUT[{12:0}]
vector accout2 accout2[{19:0}]

V XIN 00000000

V HIN 01010101 01100011 01011010 01001101 00111110 00101111 00100001 00010110
00001110 00001000 00000101 00000010 00000001 00000001 00000000 00000000 00000000
00000000 00000001 00000001 00000010 00000101 00001000 00001110 00010110 00100001 00101111
00111110 01001101 01011010 01100011 01100110

V P/R 1
V P/RINV 0
V S 1
V A 1

analyzer PHI1 PHI2 S1 A1 P/R HIN XIN XOUT ACCOUT
R

> IPAprogrammed

IPARRAYrun1.cmds -- This command file simulates the IPARRAY after it has
been programmed by running IPARRAYprogram.cmds.

Rodney Whitted, Mar. 1992

< IPArrayprogrammed

stepsize 60

```
clock  PHI    0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
clock  S      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
clock  A      0 0 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
```

```
vector XIN     XIN[{7:0}]
vector XOUT    XOUT[{7:0}]
vector HIN     HIN[{7:0}]
vector ACCOUT  ACCOUT[{12:0}]
```

```
vector xout1   xout1[{7:0}]
vector xcout1  xcout1[{7:0}]
vector xcout2  xcout2[{7:0}]
vector x1      x1[{7:0}]
vector x2      x2[{7:0}]
vector hout1   hout1[{7:0}]
vector h1      h1[{7:0}]
vector hout2   hout2[{7:0}]
vector h2      h2[{7:0}]
vector accout1 accout1[{19:0}]
vector accout2 accout2[{19:0}]
```

```
V      HIN     XXXXXXXX
V      P/R     0
V      P/RINV  1
```

```
V      XIN     00000001 00000001 00000010 00000011 00000100 00000110 00000111 00001001
00001010
```

```
100001100 00001110 00001111 00010001 00010100 00010111 00011011 00100000 00100100 00101000
00101011 00101011 00101010 00100111 00100010 00011100 00010110 00010000 00001011 00000111
00000101 00000011 00000010
```

analyzer PHI1 PHI2 S1 XIN xcout1 xcout2 XOUT ACCOUT

R

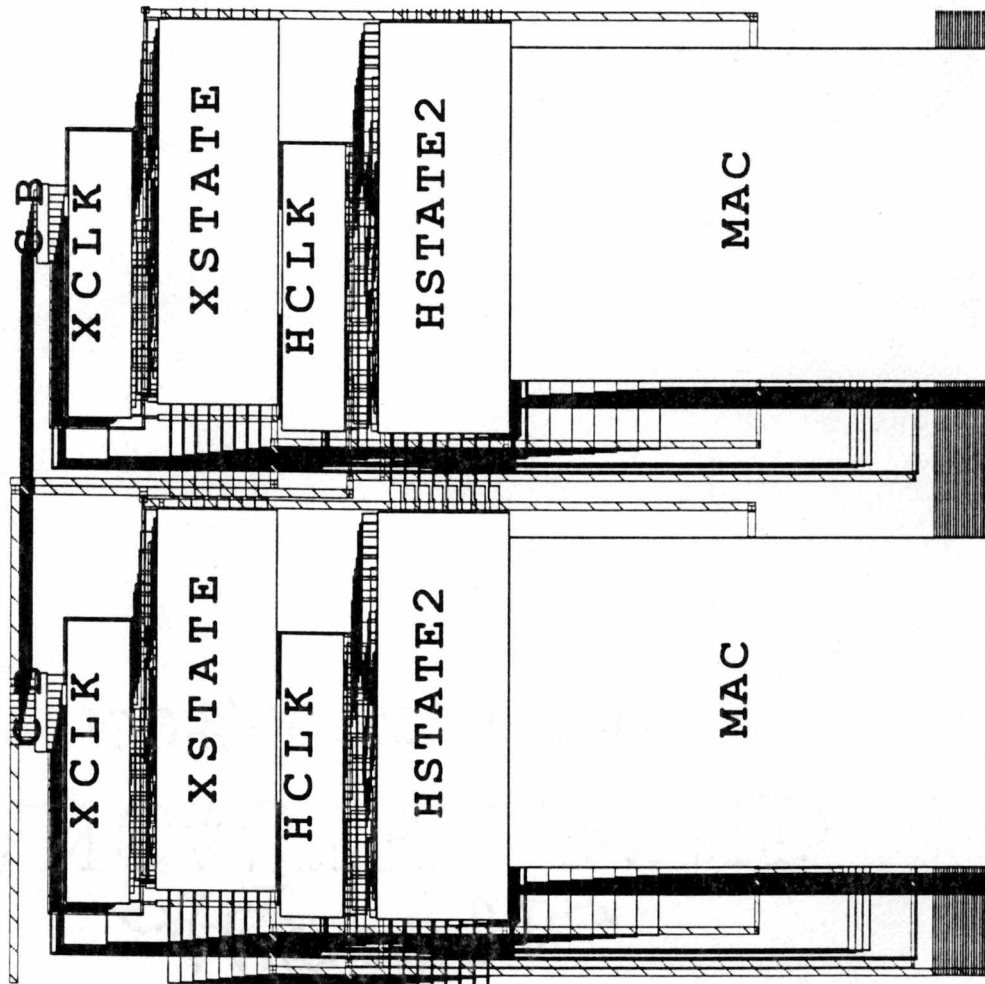
UNIVERSITY OF TENNESSEE
Electrical and Computer Engineering

DATE: May 1, 1992

Scale: 842 lambda/inch

Filename: CORE CONVOLVER.mag

User: whitted



```
;MAC.sdl -- multiply / accumulate structure for deconvolution IP processor
;Rodney Whitted, Feb. 1992
```

```
(parent-cell MAC)
(layout-generator Flint)
```

```
(subcells
  (mult    MULTIPLIER    ((n 8) (m 8) (s 4) (csindex '(3 3 3 4))))
  (ACC     ACCUMULATOR  ((N 20) (test 1)))
  (ACLK    ACLK)
)
```

```
(instance MULTIPLIER(
  (X      XEST    (width 8))
  (Y      H       (width 8))
  (P      product (width 15))

  (Vdd    Vdd)
  (GND    GND)
))
```

```
(instance ACLK(
  (A1      A1)
  (A1INV   A1INV)
  (PHI1    PHI1)
  (PHI1INV PHI1INV)
  (PHI2    PHI2)
  (PHI2INV PHI2INV)

  (CLKOUT   CLKOUT (width 10))

  (Vdd    Vdd)
  (GND    GND)
))
```

```
(net CLKOUT ((ACCUMULATOR A1 (net-base 0) (net-incr 1))))
(net CLKOUT ((ACCUMULATOR A1INV (net-base 1) (net-incr 1))))
```

```
(net CLKOUT ((ACCUMULATOR PHI1 (width 2) (net-base 2) (net-incr 4))))
(net CLKOUT ((ACCUMULATOR PHI1INV (width 2) (net-base 3) (net-incr 4))))
(net CLKOUT ((ACCUMULATOR PHI2 (width 2) (net-base 4) (net-incr 4))))
(net CLKOUT ((ACCUMULATOR PHI2INV (width 2) (net-base 5) (net-incr 4))))
```

```
(instance ACCUMULATOR(
  (ACCIN    ACCIN    (width 20))
  (ACCOUT   ACCOUT   (width 20))
  (plout    plout    (width 20))
  (mlout    mlout    (width 20))
  (mlin     mlin     (width 20))

  (CNTL_VDD Vdd)
  (CNTL_GND GND)
  (Vdd      Vdd)
  (GND      GND)
))
```

```
(net product ((ACCUMULATOR PIN (width 14) (term-base 0) (term-incr 1) (net-base 0) (net-incr 1))))
(net product ((ACCUMULATOR PIN (width 6) (term-base 14) (term-incr 1) (net-base 14) (net-incr 0))))
```

```
(net product ((parent P (width 15))))
```

```
(instance parent(  
  ((terminal ACCIN (TERM_EDGE LEFT)) ACCIN (width 20))  
  ((terminal ACCOUT (TERM_EDGE RIGHT)) ACCOUT (width 20))  
  ((terminal XEST (TERM_EDGE LEFT)) XEST (width 8))  
  ((terminal H (TERM_EDGE LEFT)) H (width 8))  
  
  ((terminal plout (TERM_EDGE RIGHT)) plout (width 20))  
  ((terminal mlout (TERM_EDGE RIGHT)) mlout (width 20))  
  ((terminal mlin (TERM_EDGE RIGHT)) mlin (width 20))  
  
  ((terminal PHI1 (TERMTYPE CLOCK)) PHI1)  
  ((terminal PHI1INV (TERMTYPE CLOCK)) PHI1INV)  
  ((terminal PHI2 (TERMTYPE CLOCK)) PHI2)  
  ((terminal PHI2INV (TERMTYPE CLOCK)) PHI2INV)  
  ((terminal A1 (TERMTYPE CLOCK)) A1)  
  ((terminal A1INV (TERMTYPE CLOCK)) A1INV)  
  
  ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)  
  ((terminal GND (TERMTYPE GROUND)) GND)  
  
))  
(end-sdl)
```

MAC.cmds -- simulation command file for MAC portion of IP
Rodney Whitted, Mar. 1992

Correct operation results in the following vector at ACCOUT:
XX 55 aa ff 154 1a9 1fe 253 2a8 sfd 352 3a7 3fc 451 4a6 4fb

h Vdd
l GND

** 8.333 MHz simulation **
stepsize 5

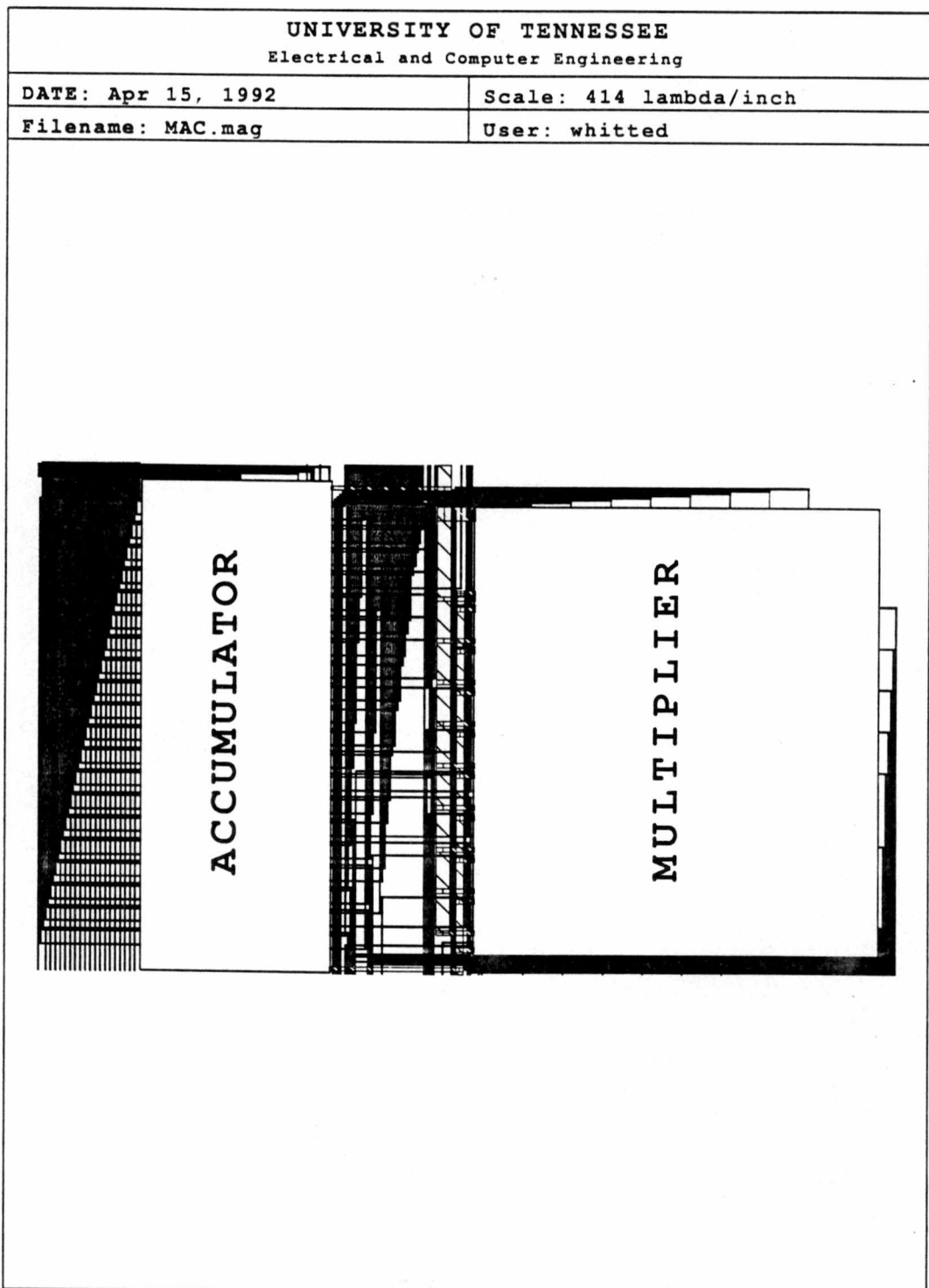
clock PHI1 000000000000011111111111
clock PHI1INV 111111111111100000000000
clock PHI2 111111111110000000000000
clock PHI2INV 000000000000111111111111

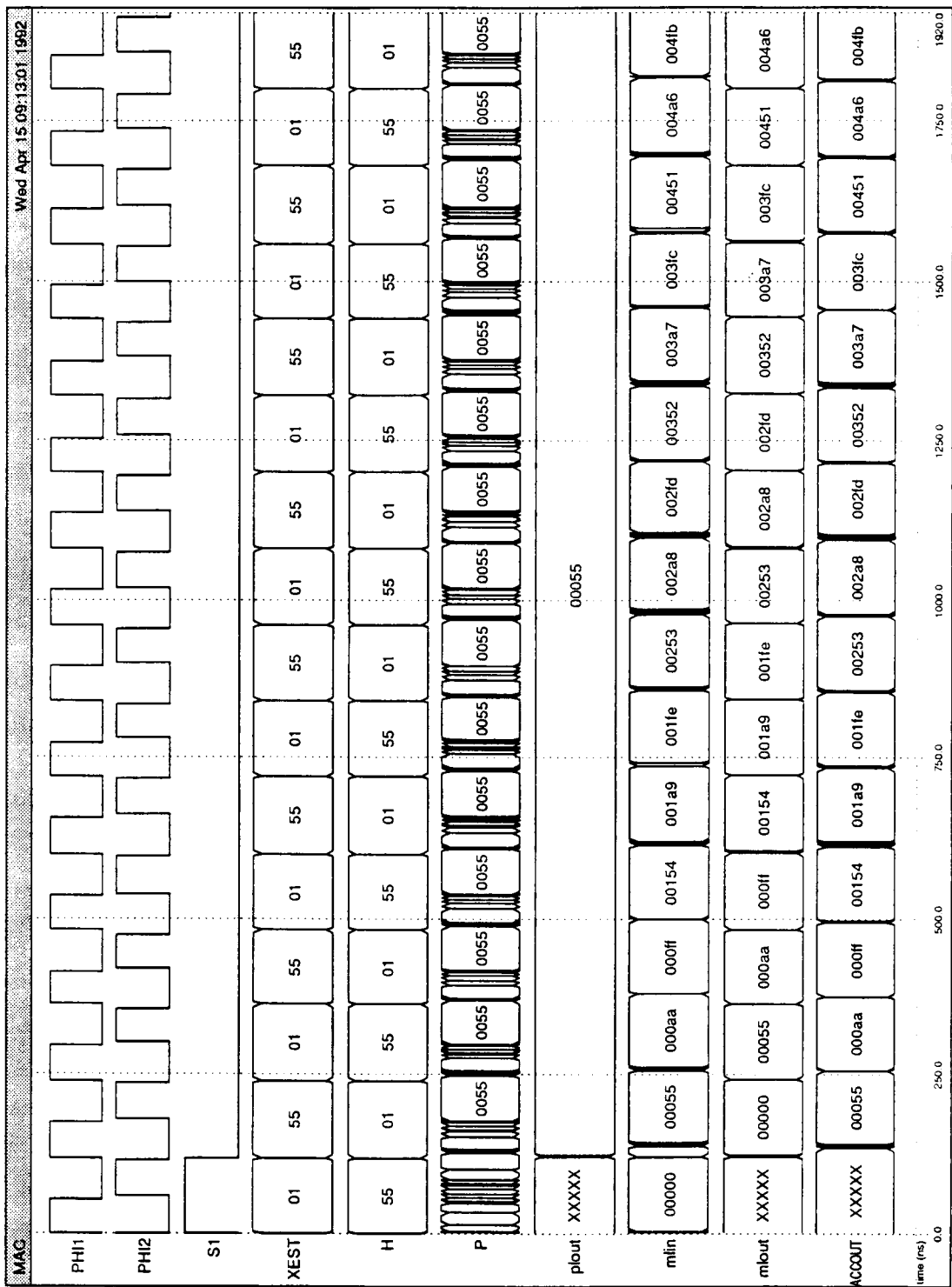
vector XEST XEST[{7:0}]
vector H H[{7:0}]
vector ACCIN ACCIN[{19:0}]
vector ACCOUT ACCOUT[{19:0}]
vector plout plout[{19:0}]
vector mlout mlout[{19:0}]
vector mlin mlin[{19:0}]
vector P P[{14:0}]

V XEST 00000001 01010101
V H 01010101 00000001
V ACCIN 00000000000000000000
V S1 10000000000000000000
V S1INV 01111111111111111111

analyzer PHI1 PHI2 S1 XEST H P plout mlin mlout ACCOUT

R





```
;MATH.sdl -- Deconvolution arithmetic pipeline.
;Rodney Whitted, Mar. 1992
```

```
(parent-cell MATH)
(parameters (N 8))
(structure-processor dpp)
(layout-generator Flint a)
```

```
(subcells
  (inverter INV ((N 8)))
  (adder      (ADDER1 ADDER2) ((N 8)))
  (scanmslatch (XL1 XL2 XL3 GL CL DL RL PL NXL) ((N 8)))
)
```

```
(instance XL1(
  (IN      XIN)
  (OUT      ADDR)

  (LOAD      S2[0])
  (LOADINV    S2INV[0])
  (PHI1      S1[0])
  (PHI1INV    S1INV[0])
  (SHIFT      CNTL_GND)
  (SHIFTINV    CNTL_VDD)

  (Vdd      Vdd)
  (GND      GND)
))
```

```
(instance XL2(
  (IN      ADDR)
  (OUT      x12out)

  (LOAD      S2[1])
  (LOADINV    S2INV[1])
  (PHI1      S1[1])
  (PHI1INV    S1INV[1])
  (SHIFT      CNTL_GND)
  (SHIFTINV    CNTL_VDD)

  (Vdd      Vdd)
  (GND      GND)
))
```

```
(instance XL3(
  (IN      x12out)
  (OUT      x13out)

  (LOAD      S2[2])
  (LOADINV    S2INV[2])
  (PHI1      S1[2])
  (PHI1INV    S1INV[2])
  (SHIFT      CNTL_GND)
  (SHIFTINV    CNTL_VDD)

  (Vdd      Vdd)
  (GND      GND)
))
```

```

))

(instance GL(
  (IN      GIN)
  (OUT      glout)

  (LOAD      S2[3])
  (LOADINV   S2INV[3])
  (PHI1      S1[3])
  (PHI1INV   S1INV[3])
  (SHIFT     CNTL_GND)
  (SHIFTINV  CNTL_VDD)

  (Vdd      Vdd)
  (GND      GND)
))

(instance CL(
  (IN      CONIN)
  (OUT      conlout)

  (LOAD      S2[4])
  (LOADINV   S2INV[4])
  (PHI1      S1[4])
  (PHI1INV   S1INV[4])
  (SHIFT     CNTL_GND)
  (SHIFTINV  CNTL_VDD)

  (Vdd      Vdd)
  (GND      GND)
))

(instance INV(
  (IN      conlout)
  (OUTINV  invout)

  (Vdd      Vdd)
  (GND      GND)
))

(instance ADDER1(
  (IN1      invout)
  (IN2      glout)
  (OUT      diff)
  (CIN      CNTL_VDD)
  (CININV   CNTL_GND)

  (Vdd      Vdd)
  (GND      GND)
))

(instance DL(
  (IN      diff)
  (OUT      DIFFOUT)

  (LOAD      S2[5])
  (LOADINV   S2INV[5])

```

```

(PHI1      S1[5])
(PHI1INV   S1INV[5])
(SHIFT     CNTL_GND)
(SHIFTINV  CNTL_VDD)

(Vdd      Vdd)
(GND      GND)
))

(instance RL(
  (IN      RAMIN)
  (OUT     RAMOUT)

  (LOAD     S2[6])
  (LOADINV  S2INV[6])
  (PHI1     S1[6])
  (PHI1INV  S1INV[6])
  (SHIFT    CNTL_GND)
  (SHIFTINV CNTL_VDD)

  (Vdd      Vdd)
  (GND      GND)
))

(instance PL(
  (IN      PIN)
  (OUT     plout)

  (LOAD     S2[7])
  (LOADINV  S2INV[7])
  (PHI1     S1[7])
  (PHI1INV  S1INV[7])
  (SHIFT    CNTL_GND)
  (SHIFTINV CNTL_VDD)

  (Vdd      Vdd)
  (GND      GND)
))

(instance ADDER2(
  (IN1     x13out)
  (IN2     plout)
  (OUT     sum)
  (CIN     CNTL_GND)
  (CININV  CNTL_VDD)

  (Vdd      Vdd)
  (GND      GND)
))

(instance NXL(
  (IN      sum)
  (OUT     XOUT)

  (LOAD     S2[8])
  (LOADINV  S2INV[8])
  (PHI1     S1[8])

```

```

(PHI1INV S1INV[8])
(SHIFT CNTL_GND)
(SHIFTINV CNTL_VDD)

(Vdd Vdd)
(GND GND)
))

(instance parent(
  ((terminal PIN (TERM_EDGE RIGHT)) PIN)
  ((terminal GIN (TERM_EDGE LEFT)) GIN)
  ((terminal CONIN (TERM_EDGE LEFT)) CONIN)
  ((terminal XIN (TERM_EDGE LEFT)) XIN)
  ((terminal XOUT (TERM_EDGE RIGHT)) XOUT)
  ((terminal ADDR (TERM_EDGE RIGHT)) ADDR)
  ((terminal RAMIN (TERM_EDGE LEFT)) RAMIN)
  ((terminal RAMOUT (TERM_EDGE LEFT)) RAMOUT)
  ((terminal DIFFOUT (TERM_EDGE RIGHT)) DIFFOUT)

  ((terminal S1 (TERMTYPE CLOCK)) S1 (width 9))
  ((terminal S1INV (TERMTYPE CLOCK)) S1INV (width 9))
  ((terminal S2 (TERMTYPE CLOCK)) S2 (width 9))
  ((terminal S2INV (TERMTYPE CLOCK)) S2INV (width 9))
  ((terminal CNTL_VDD (TERMTYPE SIGNAL)) CNTL_VDD)
  ((terminal CNTL_GND (TERMTYPE SIGNAL)) CNTL_GND)

  ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
  ((terminal GND (TERMTYPE GROUND)) GND)
))

(end-sdl)

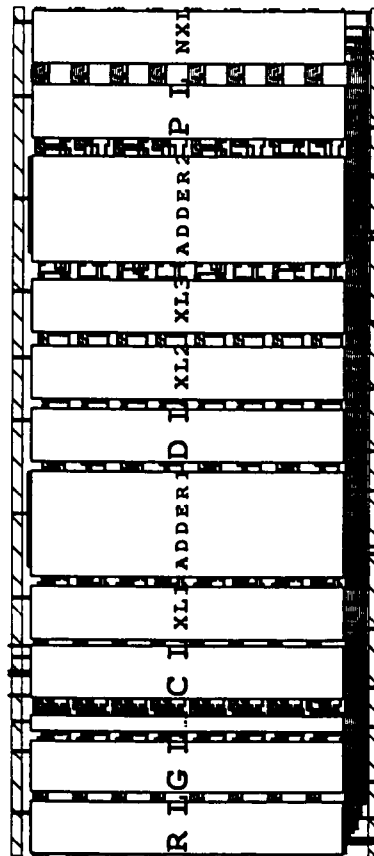
```

UNIVERSITY OF TENNESSEE	
Electrical and Computer Engineering	
DATE: Apr 22, 1992	Scale: 300 lambda/inch
Filename: MATH.mag	User: whitted

The micrograph displays a circuit board with several components labeled vertically. From top to bottom, the labels are: RIG ID, C I, XL, ADDRESS, P I, and NXT. The components are arranged in a vertical column, with each label corresponding to a specific component. The labels are oriented vertically, reading from top to bottom. The components are connected to a network of lines and pads, typical of a printed circuit board (PCB) layout. The labels are: RIG ID, C I, XL, ADDRESS, P I, and NXT.

Scale: 300 lambda/inch

User: whitted



```
;MCLK.sdl -- clock tree for driving MATH.sdl section of A_PIPE
;Rodney Whitted, Mar. 1992
```

```
(parent-cell MCLK)
```

```
(layout-generator Flint)
```

```
(subcells
```

```
  (CBA    CB1((N 4)))
```

```
  (CBA    CB2((N 8)))
```

```
  (CBA    CB3((N 24)))
```

```
)
```

```
(instance CB1(
```

```
  (IN  clk1  (width 4))
```

```
  (Vdd Vdd)
```

```
  (GND  GND)
```

```
))
```

```
(net clk1  ((parent S1  (net-base 0) (net-incr 1))))
```

```
(net clk1  ((parent S1INV (net-base 1) (net-incr 1))))
```

```
(net clk1  ((parent S2  (net-base 2) (net-incr 1))))
```

```
(net clk1  ((parent S2INV (net-base 3) (net-incr 1))))
```

```
(net clk1 (NETWIDTH 4)  ((CB1 OUT)))
```

```
(instance CB2(
```

```
  (Vdd Vdd)
```

```
  (GND  GND)
```

```
))
```

```
(net clk1 ((CB2 IN (width 2) (term-base 0) (term-incr 4) (net-base 0) (net-incr 0))))
```

```
(net clk1 ((CB2 IN (width 2) (term-base 1) (term-incr 4) (net-base 1) (net-incr 0))))
```

```
(net clk1 ((CB2 IN (width 2) (term-base 2) (term-incr 4) (net-base 2) (net-incr 0))))
```

```
(net clk1 ((CB2 IN (width 2) (term-base 3) (term-incr 4) (net-base 3) (net-incr 0))))
```

```
(net clk2 (NETWIDTH 8)  ((CB2 OUT)))
```

```
(instance CB3(
```

```
  (OUT  CLKOUT (width 20))
```

```
  (Vdd Vdd)
```

```
  (GND  GND)
```

```
))
```

```
(net clk2 ((CB3 IN (width 3) (term-base 0) (term-incr 8) (net-base 0) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 3) (term-base 1) (term-incr 8) (net-base 1) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 3) (term-base 2) (term-incr 8) (net-base 2) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 3) (term-base 3) (term-incr 8) (net-base 3) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 3) (term-base 4) (term-incr 8) (net-base 4) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 3) (term-base 5) (term-incr 8) (net-base 5) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 3) (term-base 6) (term-incr 8) (net-base 6) (net-incr 0))))
```

```
(net clk2 ((CB3 IN (width 3) (term-base 7) (term-incr 8) (net-base 7) (net-incr 0))))
```

```
(instance parent(
```

```
  ((terminal CLKOUT  (TERMTYPE CLOCK)) CLKOUT (width 20))
```

```
  ((terminal Vdd      (TERMTYPE SUPPLY)) Vdd)
```

```
  ((terminal GND      (TERMTYPE GROUND))  GND)
```

```
))
```

(end-sdl)

```
;PROGCONTROL.sdl -- program cycle controller
;Rodney B. Whitted, March 1992
```

```
(parent-cell PROGCONTROL      (bag FLATTEN_TO (CELLCLASS LEAF)))
(layout-generator Stdcell)
```

```
(subcells
  (SC_ffarray      FF ((N 2)))
  (proglogic       LOGIC)
  (SC_counter      COUNTER ((N 6)))
  (buff101 BUFF)
  (invf100 INVBUFF)
)
```

```
(instance FF(
  (DATA1          nstate (width 2))
  (Q              pstate (width 2))
  (CLK2           CLK_B)
  (RST3           RST_B)
))
```

```
(instance LOGIC(
  (nextstate      nstate (width 2))
  (presentstate   pstate (width 2))
  (start          START)
  (progenbprogenb)
  (clkenb         CLKENB)
  (load           load)
  (zero           zero)
))
```

```
(instance COUNTER(
  (in             in      (width 6))
  (out            OUT     (width 6))
  (dec            P/R)
  (load           load)
  (zero           zero)
  (clk_b          CLK_B)
  (reset_b       RST_B)
))
(net CNTL_GND ((COUNTER in (term-base 0) (term-incr 1))))
(net CNTL_GND ((COUNTER in (term-base 1) (term-incr 1))))
(net CNTL_GND ((COUNTER in (term-base 2) (term-incr 1))))
(net CNTL_GND ((COUNTER in (term-base 3) (term-incr 1))))
(net CNTL_GND ((COUNTER in (term-base 4) (term-incr 1))))

(net CNTL_VDD ((COUNTER in (term-base 5) (term-incr 1))))
```

```
(instance BUFF(
  (A1            progenb)
  (O             P/R)
))
```

```
(instance INVBUFF(
  (A1            progenb)
  (O             P/RINV)
))
```

```

(instance parent(
  (START      START)
  (RST_B      RST_B)
  (CLK_B      CLK_B)
  (CLKENB     CLKENB)
  (P/R        P/R)
  (P/RINV     P/RINV)
  (CNTL_VDD   CNTL_VDD)
  (CNTL_GND   CNTL_GND)
  (Vdd        Vdd)
  (GND        GND)
))

(end-sdl)

```

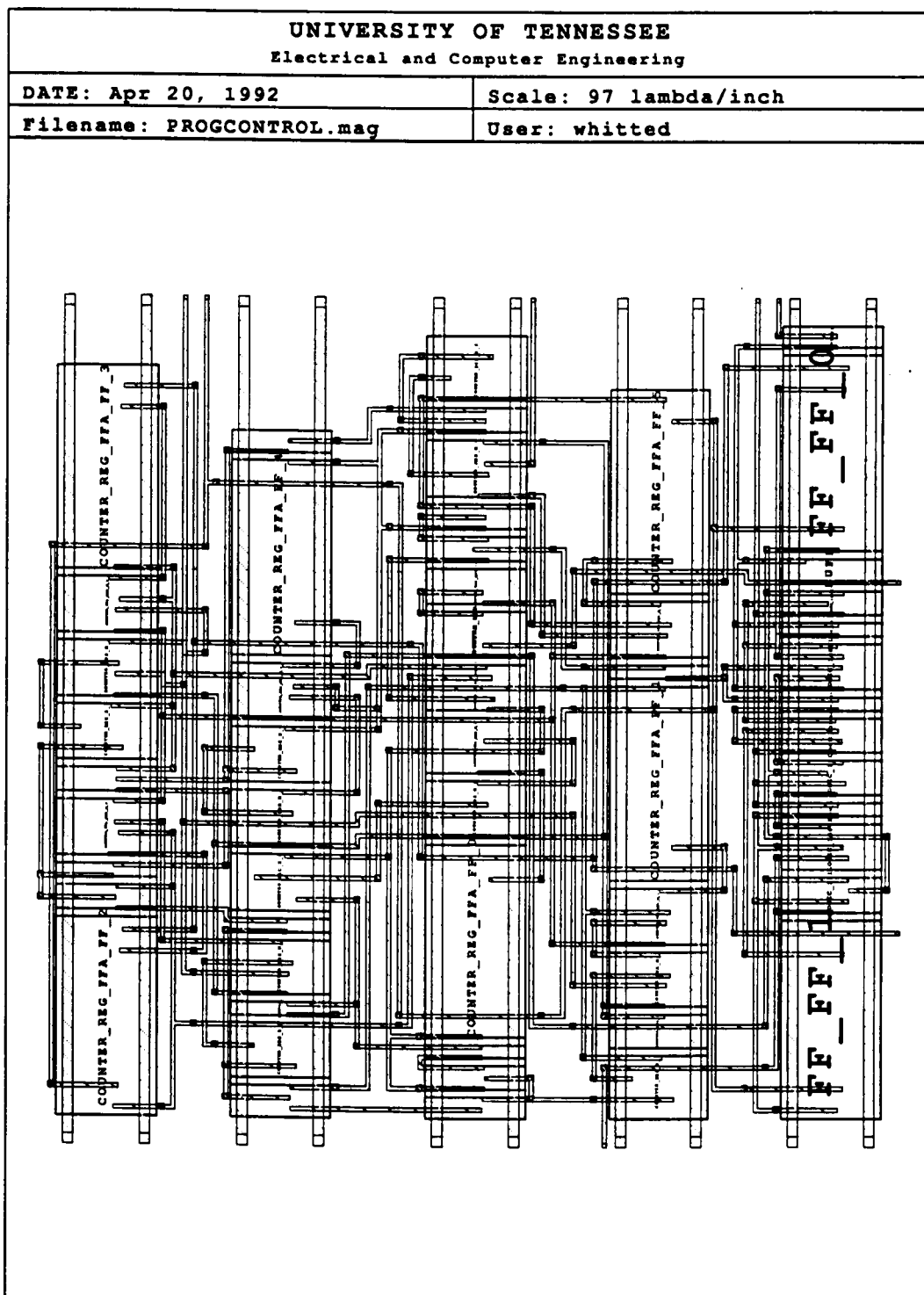
| PROGCONTROL.cmds -- IRSIM command file for testing PROGCONTROL.sdl
| Rodney Whitted, Mar. 1992

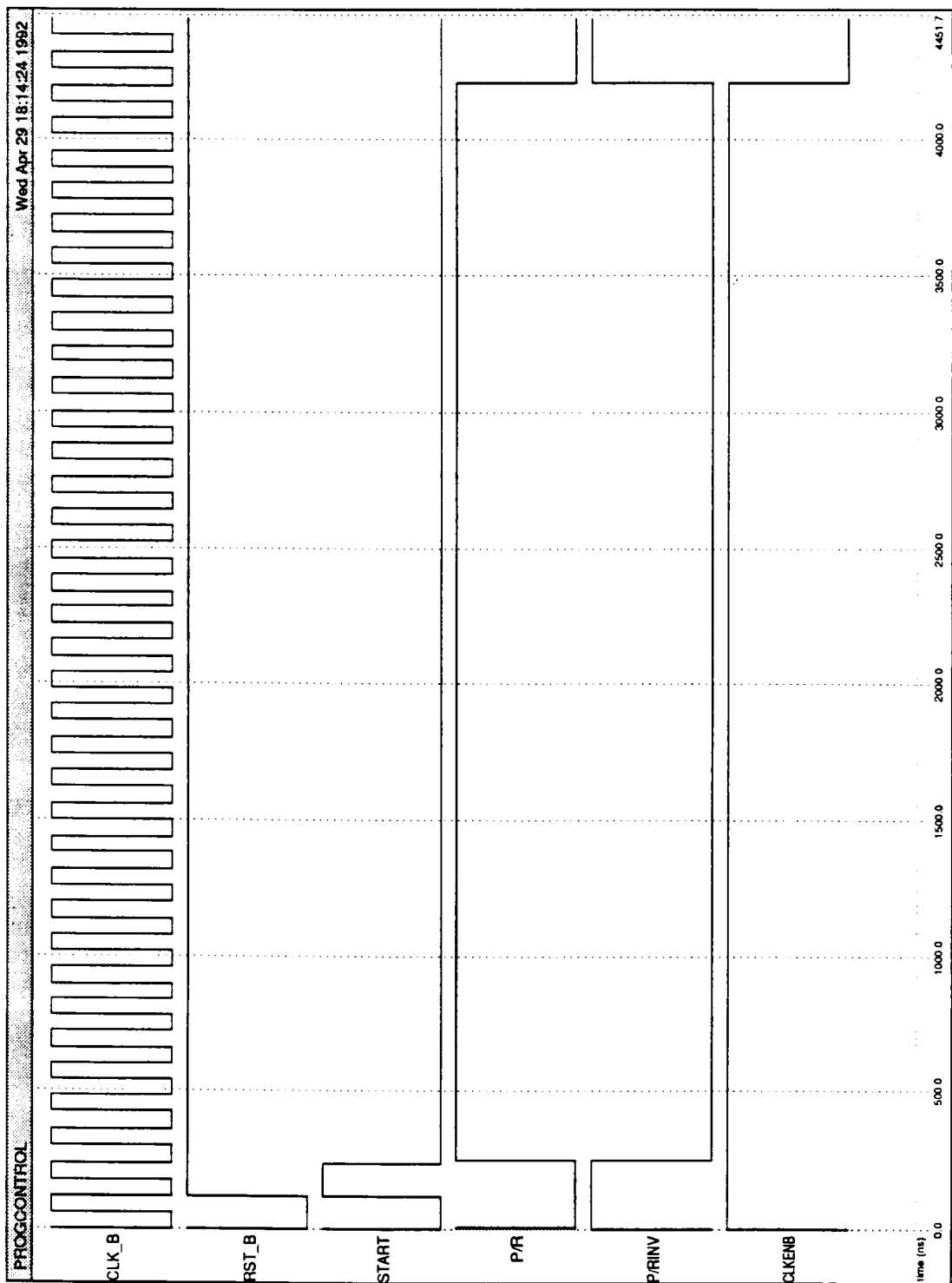
h Vdd CNTL_VDD
l GND CNTL_GND

stepsize 5
clock CLK_B 111111111111000000000000

V RST_B 011111111111111111111111111111111111
V START 00100000000000000000000000000000000000

analyzer CLK_B RST_B START P/R P/RINV CLKENB
R





```
! proglogic.bds -- logic for control FSM in deconvolution chip
! Rodney B. Whitted, March 1992
```

```
MODEL proglogic
```

```
! outputs
```

```
nextstate<1:0>,      ! input to state flip-flops
progenb,             ! programming in progress signal
load,                ! load counter signal
clkenb               ! system clocks enabled signal
=
```

```
!inputs
```

```
presentstate<1:0>,   ! output of state flip-flops
start,               ! start programming signal
zero;                ! address counter at zero signal
```

```
CONSTANT S0=0, S1=1, S2=2;      ! state encodings
```

```
ROUTINE main;
```

```
    nextstate=presentstate;
```

```
    SELECT presentstate FROM
```

```
        [S0]: ! NORMAL OPERATING STATE
```

```
        BEGIN
```

```
            progenb=0;
```

```
            load=1;
```

```
            clkenb=1;
```

```
            IF start THEN
```

```
                BEGIN
```

```
                    load=0;
```

```
                    nextstate=S1;
```

```
                END;
```

```
        END;
```

```
        [S1]: ! PROGRAMMING IN PROGRESS STATE
```

```
        BEGIN
```

```
            progenb=1;
```

```
            clkenb=1;
```

```
            IF zero THEN
```

```
                BEGIN
```

```
                    nextstate=S2;
```

```
                END;
```

```
        END;
```

```
        [S2]: ! DISABLE SYSTEM CLOCKS AND AWAIT A RESET SIGNAL
```

```
        BEGIN
```

```
            clkenb=0;
```

```
        END;
```

```
    ENDSELECT;
```

```
ENDROUTINE;
```

```
ENDMODEL;
```

```

; proglogic.sdl -- SDL interface for proglogic.bds
; Rodney B. Whitted, March 1992

(parent-cell proglogic
  (DEPENDS-ON "proglogic.bds")
)
(parameters (bdsyn "proglogic.bds"))
(structure-processor Bds2stdcell)
(layout-generator Stdcell)

(instance parent(
  (nextstate      nextstate(width 2))
  (progenbprogenb)
  (load           load)
  (clkenb         clkenb)
  (presentstate   presentstate      (width 2))
  (start          start)
  (zero           zero)
))

(end-sdl)

```

```
;SHIFTS.sdl -- Output shifters for FORMAT.sdl
;Rodney Whitted, Mar. 1992
```

```
(parent-cell SHIFTS)
(layout-generator Flint)
```

```
(subcells
  (LATCH L_IN ((N 13)))
  (MUX (LMUX MUX1 MUX2 MUX3 MUX4) ((N 13)))
)
```

```
(instance LMUX(
  (IN1      ACCIN  (width 13))
  (IN2      accnet (width 13))
  (OUT      lin    (width 13))

  (SEL1     A1)
  (SEL2     A1INV)

  (Vdd      Vdd)
  (GND      GND)
))
```

```
(instance L_IN(
  (IN      lin    (width 13))
  (OUT     accnet (width 13))

  (LOAD     PHI2)
  (LOADINV  PHI2INV)
  (PHI1     PHI1)
  (PHI1INV  PHI1INV)

  (Vdd      Vdd)
  (GND      GND)
))
```

```
(instance MUX1(
  (SEL1     SELNET[0])
  (SEL2     SELNET[1])

  (Vdd      Vdd)
  (GND      GND)
))
```

```
(net accnet ((MUX1 IN1 (width 13) (term-base 0) (term-incr 1) (net-base 0) (net-incr 1))))
(net accnet ((MUX1 IN2 (width 12) (term-base 0) (term-incr 1) (net-base 1) (net-incr 1))))
(net GND ((MUX1 IN2 (term-base 12) (term-incr 1))))
```

```
(net mux1out ((MUX1 OUT (width 13))))
```

```
(instance MUX2(
  (SEL1     SELNET[2])
  (SEL2     SELNET[3])

  (Vdd      Vdd)
  (GND      GND)
))
```

```

(net mux1out ((MUX2 IN1 (width 13) (term-base 0) (term-incr 1) (net-base 0) (net-incr 1))))
(net mux1out ((MUX2 IN2 (width 12) (term-base 0) (term-incr 1) (net-base 1) (net-incr 1))))
(net GND ((MUX2 IN2 (term-base 12) (term-incr 1))))

(net mux2out ((MUX2 OUT (width 13))))

(instance MUX3(
    (SEL1      SELNET[4])
    (SEL2      SELNET[5])

    (Vdd       Vdd)
    (GND       GND)
))
(net mux2out ((MUX3 IN1 (width 13) (term-base 0) (term-incr 1) (net-base 0) (net-incr 1))))
(net mux2out ((MUX3 IN2 (width 12) (term-base 0) (term-incr 1) (net-base 1) (net-incr 1))))
(net GND ((MUX3 IN2 (term-base 12) (term-incr 1))))

(net mux3out ((MUX3 OUT (width 13))))

(instance MUX4(
    (SEL1      SELNET[6])
    (SEL2      SELNET[7])

    (Vdd       Vdd)
    (GND       GND)
))
(net mux3out ((MUX4 IN1 (width 13) (term-base 0) (term-incr 1) (net-base 0) (net-incr 1))))
(net mux3out ((MUX4 IN2 (width 12) (term-base 0) (term-incr 1) (net-base 1) (net-incr 1))))
(net GND ((MUX4 IN2 (term-base 12) (term-incr 1))))

(net ACCOUT ((MUX4 OUT (width 13))))

(instance parent(
    ((terminal  ACCIN (TERM_EDGE TOP))      ACCIN (width 13))
    ((terminal  ACCOUT (TERM_EDGE BOTTOM))   ACCOUT (width 13))
    ((terminal  SELNET (TERM_EDGE RIGHT))    SELNET (width 8))

    ((terminal  A1      (TERM_EDGE RIGHT))   A1)
    ((terminal  A1INV   (TERM_EDGE RIGHT))   A1INV)
    ((terminal  PHI1    (TERM_EDGE RIGHT))   PHI1)
    ((terminal  PHI1INV (TERM_EDGE RIGHT))   PHI1INV)
    ((terminal  PHI2    (TERM_EDGE RIGHT))   PHI2)
    ((terminal  PHI2INV (TERM_EDGE RIGHT))   PHI2INV)

    ((terminal  Vdd     (TERMTYPE SUPPLY))    Vdd)
    ((terminal  GND     (TERMTYPE GROUND))    GND)
))

(end-sdl)

```

|SHIFTS.cmds -- sim file for testing the format shifter multiplexers
|Rodney Whitted, Mar. 1992

h Vdd
l GND

stepsize 5

clock PHI1 00000000000000111111111111
clock PHI1INV 11111111111111000000000000
clock PHI2 0111111111111100000000000000
clock PHI2INV 10000000000000111111111111

vector ACCIN ACCIN[{12:0}]
vector ACCOUT ACCOUT[{12:0}]
vector SELNET SELNET[{7:0}]

V ACCIN 0000000001111

V SELNET 01010101 01010101 01010101 01010110 01010110 01010110 01011010 01011010
01011010 01101010 01101010 01101010 10101010 10101010 10101010

V A1 100100100100100

V A1INV 011011011011011

analyzer PHI1 PHI2 A1 SELNET ACCIN ACCOUT

R

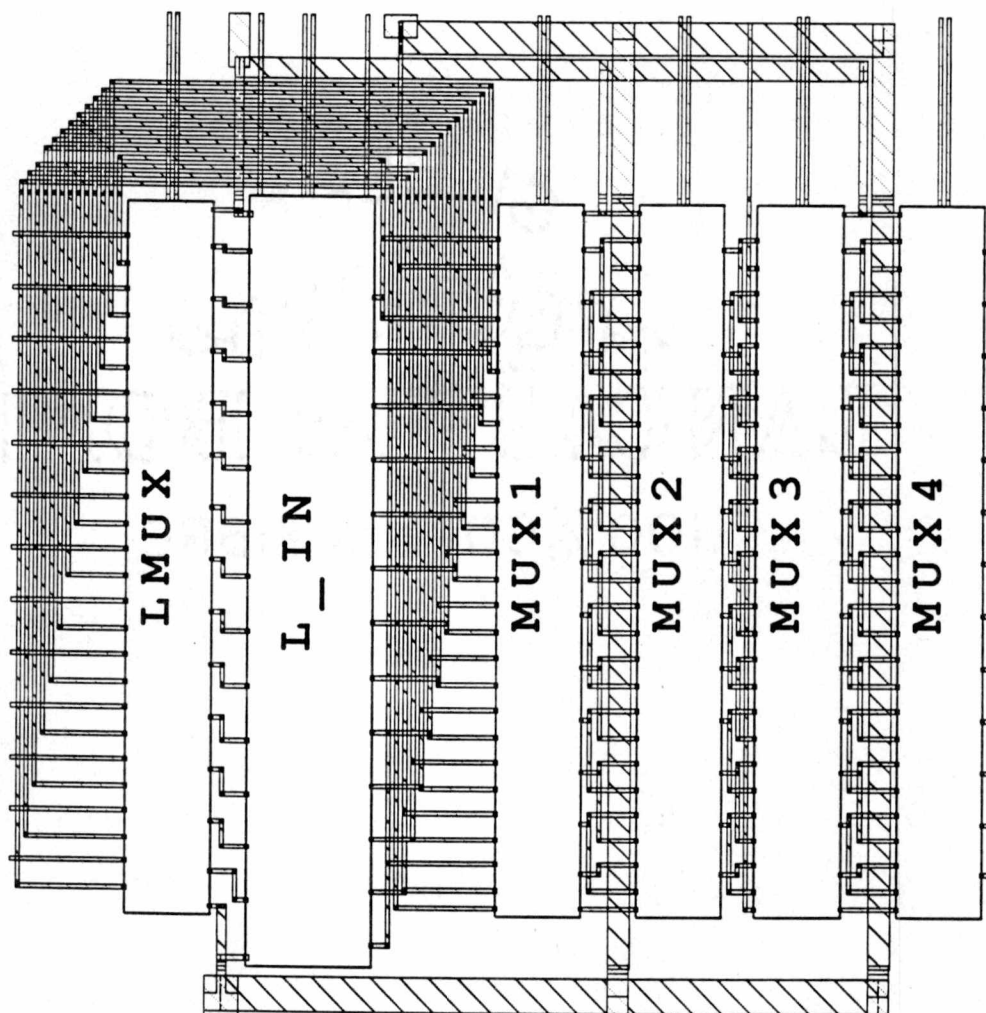
UNIVERSITY OF TENNESSEE
Electrical and Computer Engineering

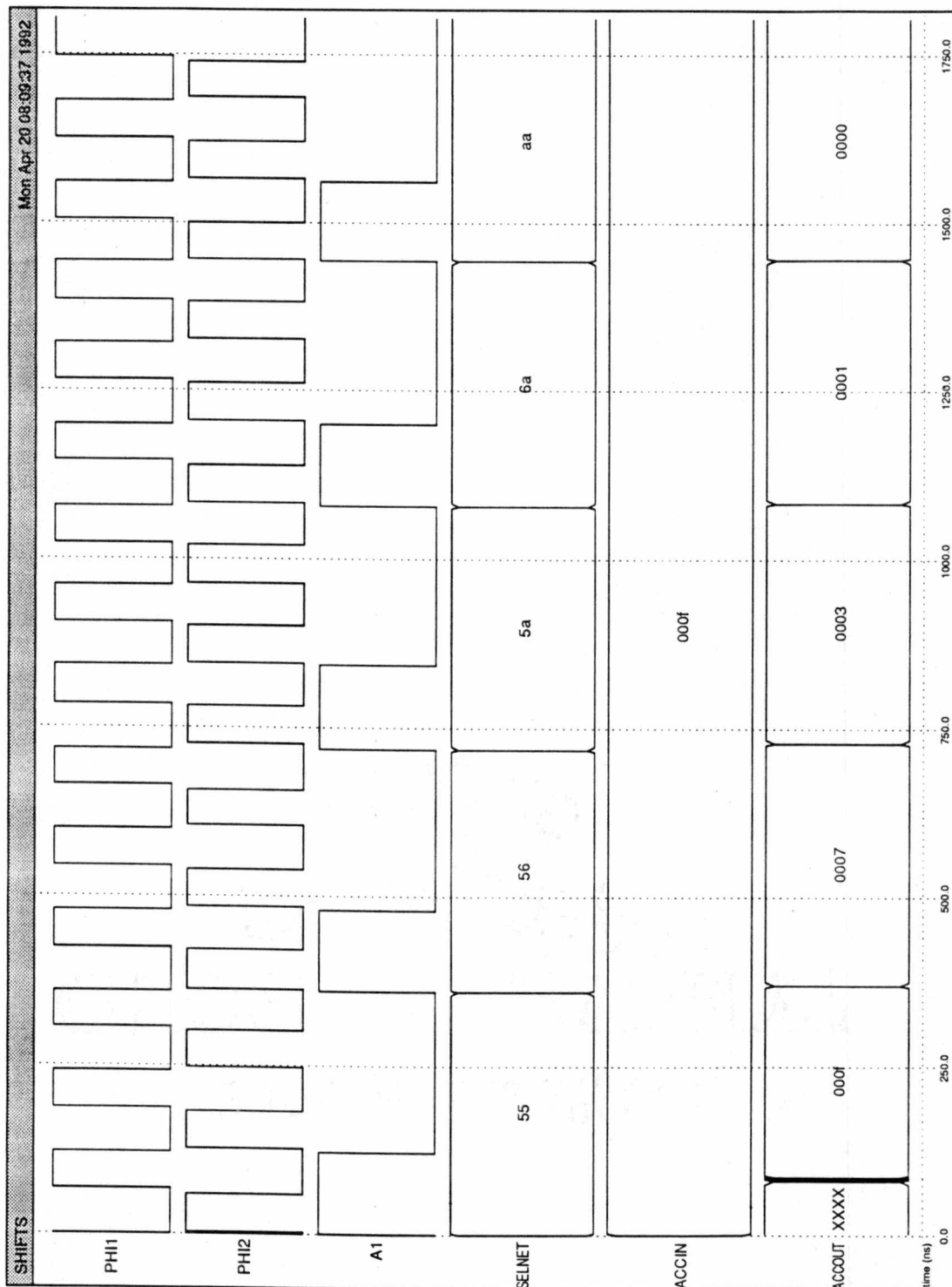
DATE: Apr 20, 1992

Scale: 141 lambda/inch

Filename: SHIFTS.mag

User: whitted





```
;XCLK.sdl -- clock tree for XSTATE
;Rodney Whitted, Mar. 1992
```

```
(parent-cell XCLK)
```

```
(layout-generator Flint)
```

```
(subcells
```

```
    (CBA    CB1    ((N 5)))
    (CBA    CB2    ((N 20)))
    (CBA    CB3    ((N 68)))
)
```

```
(instance CB1(
    (IN      clk1    (width 4))
```

```
    (Vdd     Vdd)
    (GND     GND)
```

```
)
(net clk1      ((parent PHI1      (net-base 0) (net-incr 1))))
(net clk1      ((parent PHI1INV   (net-base 1) (net-incr 1))))
(net clk1      ((parent PHI2      (net-base 2) (net-incr 1))))
(net clk1      ((parent PHI2INV   (net-base 3) (net-incr 1))))
```

```
(net clk1 (NETWIDTH 4)  ((CB1 OUT)))
```

```
(instance CB2(
    (Vdd     Vdd)
    (GND     GND)
```

```
)
(net clk1 ((CB2 IN (width 5) (term-base 0) (term-incr 4) (net-base 0) (net-incr 0))))
(net clk1 ((CB2 IN (width 5) (term-base 1) (term-incr 4) (net-base 1) (net-incr 0))))
(net clk1 ((CB2 IN (width 5) (term-base 2) (term-incr 4) (net-base 2) (net-incr 0))))
(net clk1 ((CB2 IN (width 5) (term-base 3) (term-incr 4) (net-base 3) (net-incr 0))))
```

```
(net clk2 (NETWIDTH 20) ((CB2 OUT)))
```

```
(instance CB3(
    (OUT      CLKOUT (width 68))
```

```
    (Vdd     Vdd)
    (GND     GND)
```

```
)
(net clk2 ((CB3 IN (width 4) (term-base 0) (term-incr 16) (net-base 0) (net-incr 0))))
(net clk2 ((CB3 IN (width 4) (term-base 1) (term-incr 16) (net-base 1) (net-incr 0))))
(net clk2 ((CB3 IN (width 4) (term-base 2) (term-incr 16) (net-base 2) (net-incr 0))))
(net clk2 ((CB3 IN (width 4) (term-base 3) (term-incr 16) (net-base 3) (net-incr 0))))
(net clk2 ((CB3 IN (width 4) (term-base 4) (term-incr 16) (net-base 4) (net-incr 0))))
(net clk2 ((CB3 IN (width 4) (term-base 5) (term-incr 16) (net-base 5) (net-incr 0))))
(net clk2 ((CB3 IN (width 4) (term-base 6) (term-incr 16) (net-base 6) (net-incr 0))))
(net clk2 ((CB3 IN (width 4) (term-base 7) (term-incr 16) (net-base 7) (net-incr 0))))
(net clk2 ((CB3 IN (width 4) (term-base 8) (term-incr 16) (net-base 8) (net-incr 0))))
(net clk2 ((CB3 IN (width 4) (term-base 9) (term-incr 16) (net-base 9) (net-incr 0))))
(net clk2 ((CB3 IN (width 4) (term-base 10) (term-incr 16) (net-base 10) (net-incr 0))))
(net clk2 ((CB3 IN (width 4) (term-base 11) (term-incr 16) (net-base 11) (net-incr 0))))
(net clk2 ((CB3 IN (width 4) (term-base 12) (term-incr 16) (net-base 12) (net-incr 0))))
(net clk2 ((CB3 IN (width 4) (term-base 13) (term-incr 16) (net-base 13) (net-incr 0))))
(net clk2 ((CB3 IN (width 4) (term-base 14) (term-incr 16) (net-base 14) (net-incr 0))))
```

```

(net clk2 ((CB3 IN (width 4) (term-base 15) (term-incr 16) (net-base 15) (net-incr 0))))
(net clk2 ((CB3 IN (width 1) (term-base 64) (term-incr 1) (net-base 16) (net-incr 0))))
(net clk2 ((CB3 IN (width 1) (term-base 65) (term-incr 1) (net-base 17) (net-incr 0))))
(net clk2 ((CB3 IN (width 1) (term-base 66) (term-incr 1) (net-base 18) (net-incr 0))))
(net clk2 ((CB3 IN (width 1) (term-base 67) (term-incr 1) (net-base 19) (net-incr 0))))

(instance parent(
    ((terminal CLKOUT          (TERMTYPE CLOCK))      CLKOUT (width 68))

    ((terminal Vdd             (TERMTYPE SUPPLY))      Vdd)
    ((terminal GND             (TERMTYPE GROUND))      GND)
))

(end-sdl)

```

```
:XLATCH.sdl --XOUT memory for use in FORMAT.sdl
;Rodney Whitted, Mar. 1992
```

```
(parent-cell XLATCH)
(parameters (N 8))
(structure-processor dpp)
(layout-generator Flint a)
(subcells
  (mux2to1      MUX  ((N N)))
  (scanmslatch XL   ((N N)))
)
(instance MUX(
  (IN1      XIN)
  (IN2      XOUT)
  (OUT      xlin)

  (SEL1      S1)
  (SEL2      S1INV)

  (Vdd       Vdd)
  (GND       GND)
))
(instance XL(
  (IN      xlin)
  (OUT     XOUT)

  (LOAD     PHI2)
  (LOADINV  PHI2INV)
  (PHI1     PHI1)
  (PHI1INV  PHI1INV)

  (SHIFT    GND)
  (SHIFTINV Vdd)
  (Vdd      Vdd)
  (GND      GND)
))
(instance parent(
  ((terminal  XIN      (TERM_EDGE LEFT))  XIN)
  ((terminal  XOUT     (TERM_EDGE RIGHT)) XOUT)

  ((terminal  S1       (TERM_EDGE TOP))    S1)
  ((terminal  S1INV    (TERM_EDGE TOP))    S1INV)

  ((terminal  PHI1     (TERM_EDGE TOP))    PHI1)
  ((terminal  PHI1INV  (TERM_EDGE TOP))    PHI1INV)
  ((terminal  PHI2     (TERM_EDGE TOP))    PHI2)
  ((terminal  PHI2INV  (TERM_EDGE TOP))    PHI2INV)

  ((terminal  Vdd      (TERMTYPE SUPPLY))  Vdd)
  ((terminal  GND      (TERMTYPE GROUND))  GND)
))
(end-sdl)
```

XLATCH.cmds -- simulation command file for XLATCH.sdl
|Rodney Whitted, Mar. 1992

h Vdd
l GND

!** 8.333 MHz simulation **
stepsize 5

clock PHI1 000000000000011111111111
clock PHI1INV 1111111111111100000000000
clock PHI2 1111111111110000000000000
clock PHI2INV 000000000001111111111111

vector XIN XIN[{7:0}]
vector XOUT XOUT[{7:0}]

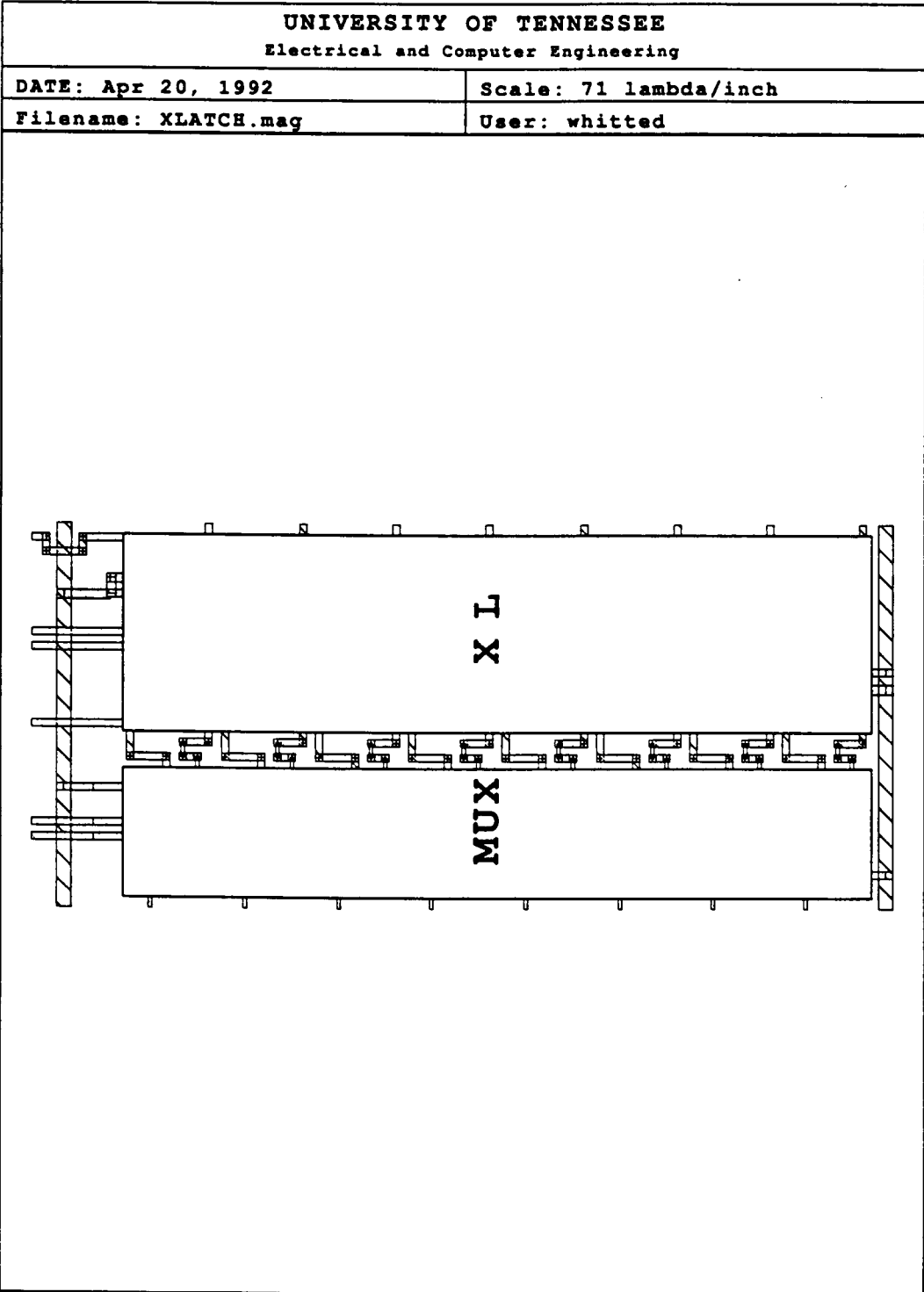
V XIN 00000001 00000010 00000011 00000100 00000101 00000110 00000111 00001000
00001001

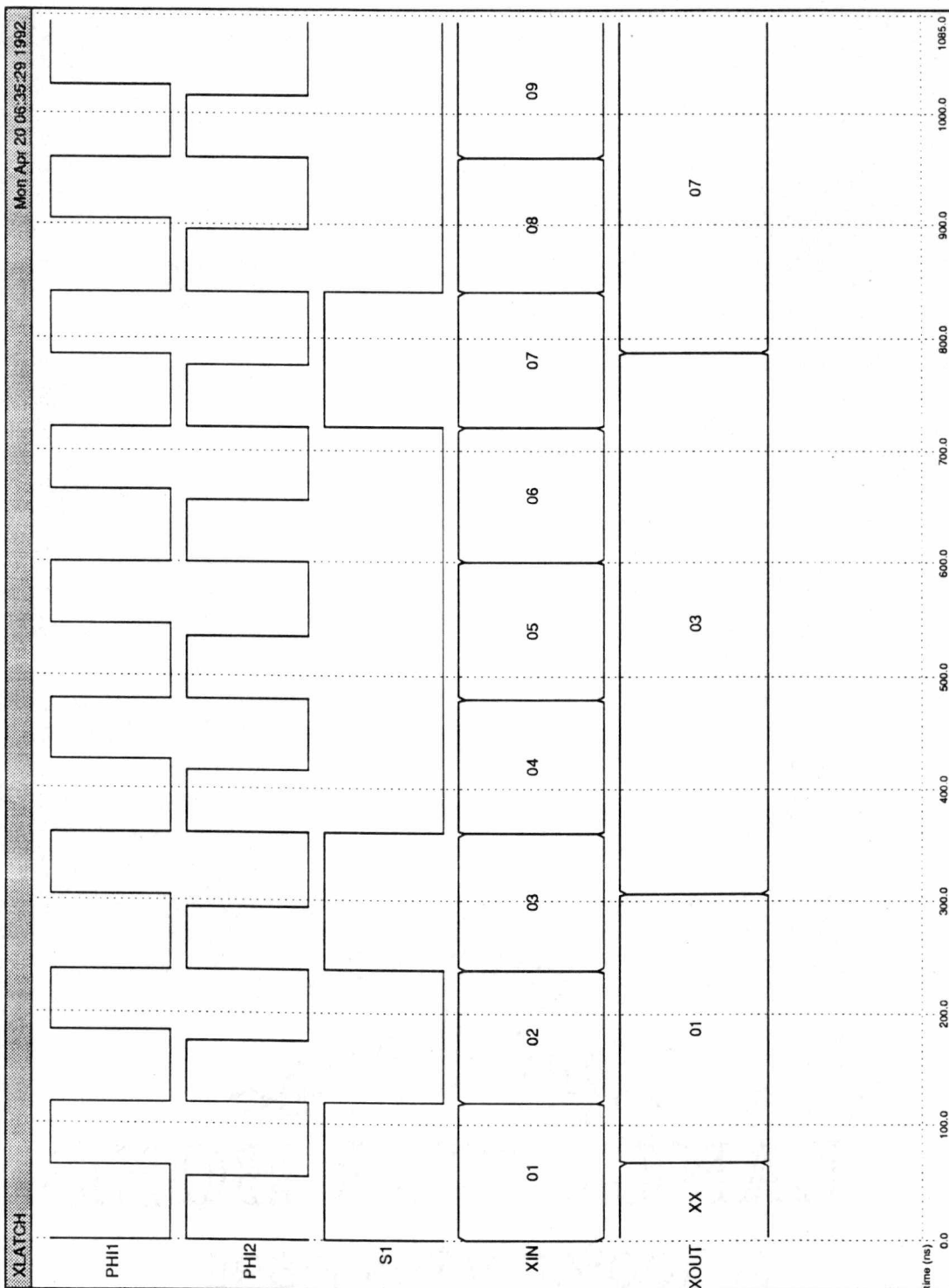
V S1 101000100

V S1INV 010111011

analyzer PHI1 PHI2 S1 XIN XOUT

R





```
;XSTATE.sdl -- x-input vector state memory used in IP
;Rodney Whitted, Mar. 1992
```

```
(parent-cell XSTATE)
(parameters (N 8) (delay 17))
(structure-processor dpp)
(layout-generator Flint a)
(subcells
  (mux2to1      MUX      ((N N)))
  (scanmslatch L_IN    ((N N)))
)

(instance MUX(
  (IN1      XIN)
  (IN2      XOUT)
  (OUT      muxout)
  (SEL1     S1)
  (SEL2     S1INV)
  (Vdd      Vdd)
  (GND      GND)
))

(instance L_IN(
  (IN      muxout)
  (OUT     X)
  (LOAD    PHI2[0])
  (LOADINV PHI2INV[0])
  (PHI1    PHI1[0])
  (PHI1INV PHI1INV[0])
  (SHIFT   CNTL_GND)
  (SHIFTINV CNTL_VDD)
  (Vdd     Vdd)
  (GND     GND)
))

(dotimes (i (- delay 1))
  (subcells
    (scanmslatch L ((N N)))
  )
  (instance L(
    (IN (indum (ALIAS (if (= i 0)
                      (format nil "X")
                      (format nil "in[~D]" i)
                    ))))
    (OUT (outdum (ALIAS (if (= i (- delay 2))
                          (format nil "XOUT")
                          (format nil "in[~D]" (+ i 1))
                        ))))
    (LOAD (ld1 (ALIAS (format nil "PHI2[~D]" (+ i 1))))))
    (LOADINV (ldi1 (ALIAS (format nil "PHI2INV[~D]" (+ i 1))))))
    (PHI1 (p1 (ALIAS (format nil "PHI1[~D]" (+ i 1))))))
    (PHI1INV (pi1 (ALIAS (format nil "PHI1INV[~D]" (+ i 1))))))
    (SHIFT CNTL_GND)
    (SHIFTINV CNTL_VDD)
    (Vdd Vdd)
    (GND GND)
  ))
)
```

```

)

(instance parent(
  ((terminal XIN (TERM_EDGE LEFT)) XIN)
  ((terminal XOUT (TERM_EDGE RIGHT)) XOUT)
  ((terminal X (TERM_EDGE LEFT)) X)

  ((terminal S1 (TERM_EDGE TOP)) S1)
  ((terminal S1INV (TERM_EDGE TOP)) S1INV)

  ((terminal PHI1 (TERM_EDGE TOP)) PHI1 (width delay))
  ((terminal PHI1INV (TERM_EDGE TOP)) PHI1INV (width delay))
  ((terminal PHI2 (TERM_EDGE TOP)) PHI2 (width delay))
  ((terminal PHI2INV (TERM_EDGE TOP)) PHI2INV (width delay))

  ((terminal CNTL_VDD (TERMTYPE SUPPLY)) CNTL_VDD)
  ((terminal CNTL_GND (TERMTYPE GROUND)) CNTL_GND)
  ((terminal Vdd (TERMTYPE SUPPLY)) Vdd)
  ((terminal GND (TERMTYPE GROUND)) GND)
))
(end-sdl)

```

|XSTATE.cmds -- IRSIM command file for testing XSTATE
| Rodney B. Whitted, March 1992

h Vdd CNTL_VDD
l GND CNTL_GND

!** 8.333MHz simulation **
stepsize 5

vector PHI1 PHI1[{16:0}]
vector PHI1INV PHI1INV[{16:0}]
vector PHI2 PHI2[{16:0}]
vector PHI2INV PHI2INV[{16:0}]

clock PHI1 0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 1111111111111111 1111111111111111 1111111111111111
1111111111111111 1111111111111111 1111111111111111 1111111111111111
1111111111111111 1111111111111111 1111111111111111 1111111111111111

clock PHI1INV 1111111111111111 1111111111111111 1111111111111111 1111111111111111
1111111111111111 1111111111111111 1111111111111111 1111111111111111
1111111111111111 1111111111111111 1111111111111111 1111111111111111
1111111111111111 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000

clock PHI2 0000000000000000 1111111111111111 1111111111111111 1111111111111111
1111111111111111 1111111111111111 1111111111111111 1111111111111111
1111111111111111 1111111111111111 1111111111111111 1111111111111111
0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000

clock PHI2INV 1111111111111111 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000
0000000000000000 0000000000000000 0000000000000000 0000000000000000
1111111111111111 1111111111111111 1111111111111111 1111111111111111
1111111111111111 1111111111111111 1111111111111111 1111111111111111
1111111111111111 1111111111111111 1111111111111111 1111111111111111

vector XIN XIN[{7:0}]
vector X X[{7:0}]
vector XOUT XOUT[{7:0}]

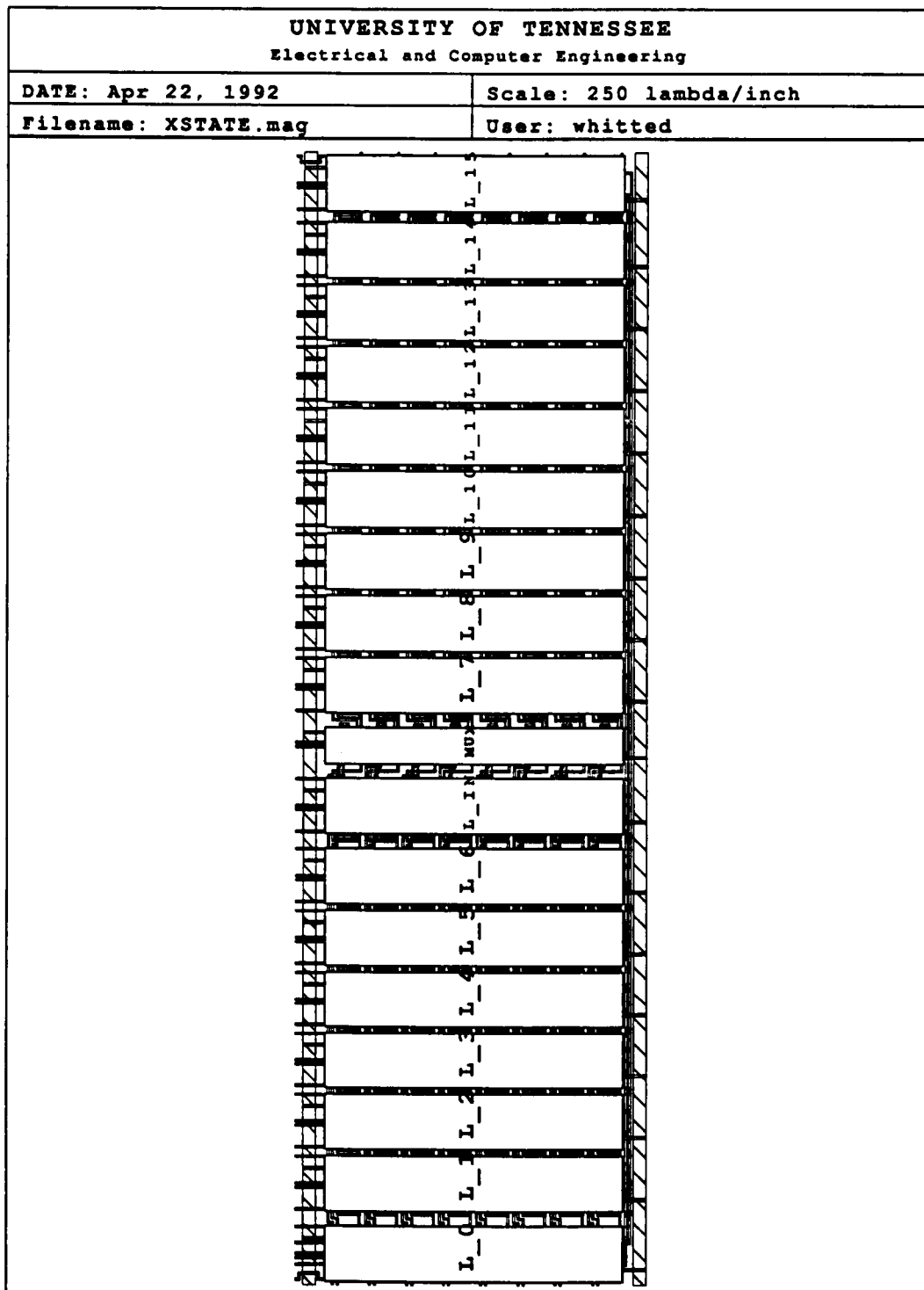
V XIN 11111111

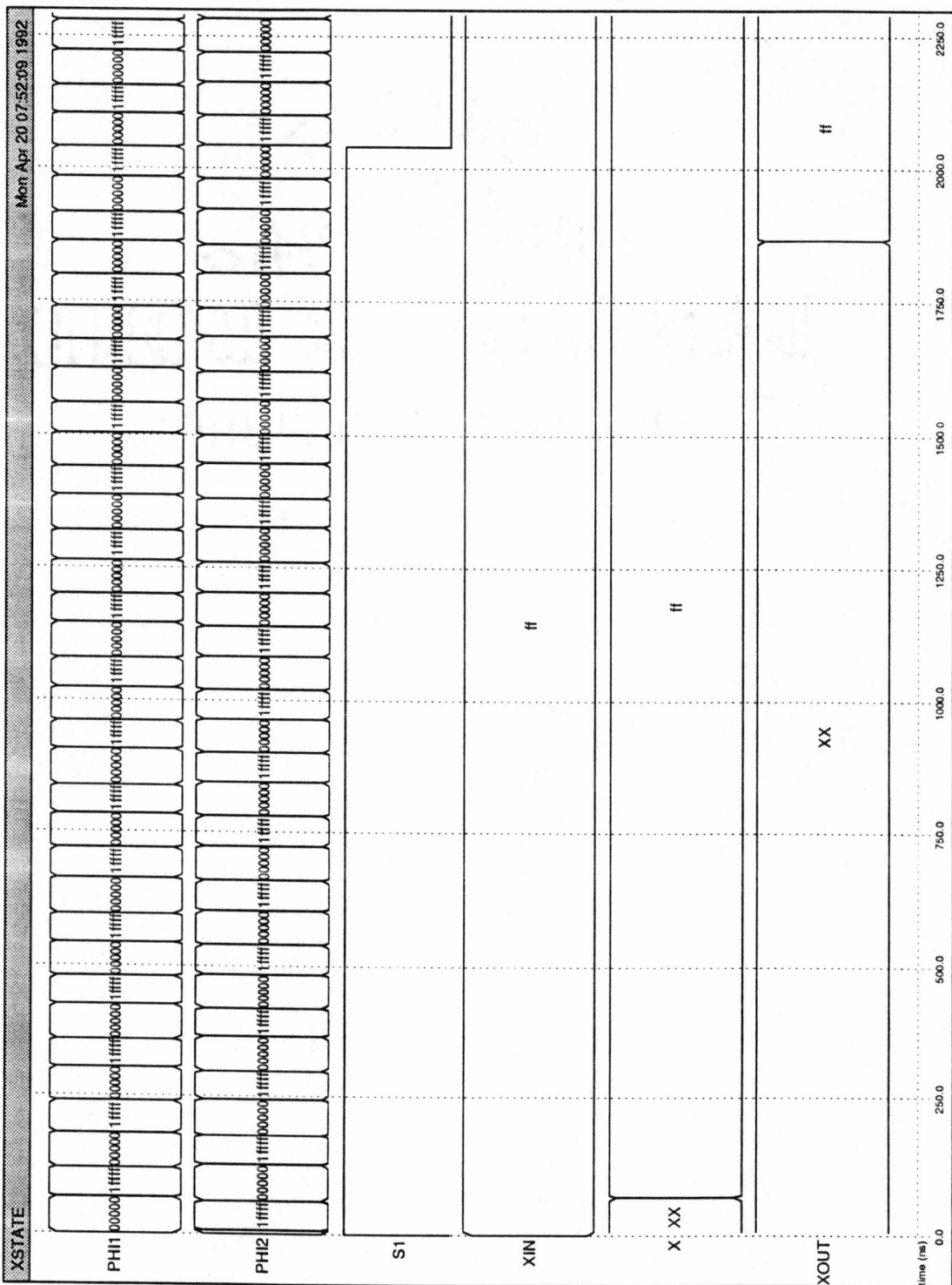
V S1 1111111111111111000

V S1INV 00000000000000000111

analyzer PHI1 PHI2 S1 XIN X XOUT

R





VITA

Rodney Buford Whitted was born in Atlanta, Georgia on January 4, 1965. He was raised and educated in the foothills of eastern Tennessee. In June 1988, he received the Bachelor of Science Degree in Electrical Engineering from the University of Tennessee, Knoxville. During this time he participated in the College of Engineering Cooperative Education Program. After graduation, he fell in love with, and married, Nancy Lynn Smith. He spent the next year working as an electronics design engineer for the Department of Defense in Baltimore, Maryland. He reentered The University of Tennessee at Knoxville as a graduate student in the Electrical and Computer Engineering department. He was awarded the Master of Science degree in Electrical Engineering in May 1992.