RELAX2: A MODIFIED WAVEFORM RELAXATION APPROACH TO THE SIMULATION OF MOS DIGITAL CIRCUITS

J. White and A.L. Sangiovanni-Vincentelli
Department of EECS
University of California at Berkeley
Berkeley, CA 94720

Abstract: Waveform Relaxation (WR) algorithms have been proven to be effective in the transient analysis of large scale integrated circuits. A new waveform relaxation simulator for MOS digital circuits, RELAX2, is described. Several speed-up techniques included in RELAX2, such as adjusting the length of the inteval of simulation, using simpler models in the first few iterations, and allowing looser timestep control in the first few iterations, are also presented.

INTRODUCTION

Waveform Relaxation (WR) is a family of relaxation-based algorithms for the solution of large scale systems of mixed algebraic-differential equations[1,2]. A particular algorithm of the WR family, the "Gauss-Seidel" WR algorithm, was successfully implemented in RELAX, an experimental simulator for MOS digital circuits[3]. This algorithm is guaranteed to converge to the solution of the circuit equations for a large class of circuits[1,2] and exploits the almost unidirectional properties of the basic components, logic gates, of digital circuits.

Due to their favorable numerical properties, WR algorithms have captured considerable attention. WR algorithms have been applied to the solution of piecewise-linear differential equations[4], they have been used in mixed-mode simulators[5], and special purpose multi-processor architecture is being studied to implement the WR algorithm.

In this paper we give a brief outline of the WR method, followed by a description of of the RELAX2 program, a new WR simulator for MOS digital circuits. We then discuss the simulation of circuits that contain logical feedback loops, and explain why adjusting the length of the interval of simulation improves the rate of convergence of the WR method. Results of the simulation of a test circuit with logic feedback using RELAX2 are presented. Finally, two new speed-up techniques tested using RELAX2, using simpler models in the first few iterations, and allowing looser timestep control in the first few iterations, are presented along with test results.

1. AN OUTLINE OF THE WAVEFORM RELAXATION ALGORITHM

We start with a simple illustrative example, and then give the general "Gauss-Seidel" algorithm in the WR family. A more detailed and complete description of these techniques is available in [1,2]. Consider the 1st order two dimensional differential equation in x(t) $\in \mathbb{R}^2$ on t $\in$ [0,T].

$$\dot{x}_1 = f_1(x_1, x_2, t) \qquad x_1(0) = x_{10} \tag{1.1a}$$

$$\dot{x}_2 = f_2(x_1, x_2, t) \qquad x_2(0) = x_{20} \tag{1.1b}$$

The basic idea of the "Gauss-Seidel" waveform relaxation algorithm is to fix the waveform x_2:[0,T] $\rightarrow \mathbb{R}$ and solve (2.1a) as a one dimensional differential equation in $x_1(\cdot)$. The solution thus obtained for x_1 can then be substituted into (2.1b) which will reduce to another first order differential equation in one variable, x_2. We then return to (2.1a) and repeat the procedure.

In this fashion, an iterative algorithm has been constructed. It replaces the problem of solving a differential equation in two variables by one of solving a sequence of differential equations in one variable. As described above, the waveform relaxation algorithm can been seen as an analogue of the Gauss-Seidel technique for solving nonlinear algebraic equations. Here, however, our unknowns are waveforms (elements of a function space), rather than real variables. In this sense, the algorithm is a technique for time domain decoupling of differential equations.

WR algorithms applied to circuits can have several formulations. Here we present the "Gauss-Seidel" WR algorithm for solving a the following system of differential equations.

$$f(\dot{x}(t), x(t), u(t)) = 0 \qquad x(0) = x_0 \tag{1.2}$$

where x(t) $\in \mathbb{R}^n$ on t $\in$ [0,T]; u(t) $\in \mathbb{R}^r$ on t $\in$ [0,T], piecewise continuous; and f: $\mathbb{R}^n x \mathbb{R}^n x \mathbb{R}^r \longrightarrow \mathbb{R}^n$ is a continuous map, and is Lipschitz continuous in x(t).

WR ALGORITHM TO ANALYZE (1.1) FROM t = 0 TO t = T

```
Comments: The superscript k denotes the iteration count,
          the subscript denotes the component index of a vector.

Step 0: (Initialization)
        Set k=0 and make an initial guess of the waveform
        $x^0(t)$ ; $t \in [0,T]$ such that $x^0(0) = x_0$.

Step 1: (Iteration)
        Repeat
          k = k+1
          For i = 1, 2, ...., n

            Solve for $x^{k_i}(t)$, $t \in [0,T]$ from
            $f_i(\dot{x}_1^k, \ldots, \dot{x}_i^k, \dot{x}_{i+1}^{k-1}, \ldots, \dot{x}_n^{k-1}, x_1^k, \ldots, x_i^k, x_{i+1}^{k-1}, \ldots, x_n^{k-1}, u) = 0$
        Until convergence.
```

2. RELAX2 PROGRAM STRUCTURE

Node-by-node decomposition, suggested by the above waveform relaxation algorithm, is a poor choice for large digital circuits. Digital circuits are usually made up of many subcircuits, each with a few tightly coupled nodes, (flip-flops or gates, for example) but these subcircuits are loosely coupled to each other. It is therefore both natural and advantageous (convergence is faster) to decompose a large digital system along subcircuit boundaries. The RELAX2 program insists the user define his large circuit by first defining subcircuits, such as gates or flip-flops, and then specifying how subcircuits are connected. A user may not refer to a transistor when describing his large circuit, but can define the transistor as a subcircuit and then refer to that subcircuit in the large circuit description.

Subcircuits may be made of any number of MOS transistors and grounded capacitors, and may contain any number of nodes. The user must explicitly state which nodes in the subcircuit can connect to other subcircuits. These nodes are refered to as **external nodes.** All other nodes in the subcircuit are **internal nodes.** The user must also specify the directionality of his circuit by indicating which of the external nodes are outputs, and which are inputs. This information is used to determine the order of subcircuit processing, or the scheduling, which will be described later.

Once the subcircuits are defined, the large circuit is defined by describing the interconnection of subcircuits. The large circuit desciption may also contain grounded capacitors, which allows the user to insert parasitic capacitances to model delays along long wires.

The RELAX2 program generates a device list "master" for each of the defined subcircuits. There is an entry in the device list master for each transistor or capacitor in the subcircuit. The entry contains a node number assignment and a space for a pointer to a node waveform for each terminal in the device. The subcircuit "masters" are stored on a simple linked list.

A copy of a subcircuit master is generated for each reference to a subcircuit in the large circuit description. Each time a copy of a master is made, the node waveform pointers must be defined. This process is refered to as **subcircuit instantiation**, and the subcircuit device list copies are refered to as **subcircuit instances**. If the node is an external node, and if space for that node waveform has already been allocated because the node was referenced by previously instantiated subcircuit, then a pointer to that space is placed in the device list entry. If not, space for the waveform is allocated and then the pointer to that space is placed in the entry. If the node is an internal node, no other subcircuit instance can reference the node, so space for the internal node waveform is immediately allocated, and a pointer to that space is placed in the entry.

Often designers use wired-or logic, and they may describe this by connecting together outputs from different subcircuits. Pass transistors that reference the same node may also be described as separate subcircuits that share an output node. The RELAX2 program must detect this construction and convert the several subcircuits involved into one collection of subcircuits. These collections are refered to as **circuit lumps**. Note that no two lumps will have an output node in common; they may however share input nodes.

Once the instantiation of subcircuits has been completed, and subcircuit instances that share a common output have been lumped together, the loading effects of other lumps must be incorporated into each lump. A lump that has input nodes that are common to a given lump's output node is refered to as a **load lump** of the given lump. One approach to incorporating the loading effects of load lumps would be to make circuits comprised of a lump and all its load lumps This would create very large circuits, which is contrary to the intent of this decomposition method. Therefore, onlv the load devices are extracted from the load lumps, and only these load devices are appended to the original lump. This is refered to as **load extraction**. Given the structure of the device list generated for each of the subcircuit instances, it is easy to extract the load devices. The device entry in a circuit lump contains pointers to its node waveforms, so copying the device entry mantains the reference to the device's node waveforms.

A description of the algorithm is the following:

LOAD EXTRACTION ALGORITHM

```
Step 0: Start with an arbitrary circuit lump
Step 1: Pick an output external node of that lump
Step 2: Visit all the circuit lumps that share that external
        output node (note that this must be an input node
        for other lumps)
Step 3: Copy only the device entries in the other lumps
        that have a terminal connected to the that external
        node.
Step 4: Append the copied device entries to the original
        circuit lump.
Step 5: Pick the next external output node from the original
        lump and Go to Step 2. If there are no more pick the
        next circuit lump and go to Step 1.
```

Once extraction has been completed, the order in which the node waveforms for the circuit lumps will be solved for must be determined. This is an important because the speed of convergence of the WR method is strongly dependent on how well this order follows the directionality of the circuit. As the subcircuit definitions specify input and output nodes, it is easy to follow the directionality of the circuit unless there are feedback loops. If there are feedback loops, the loops are temporarily broken at an arbitrary point, and the ordering is completed. The actual ordering algorithm used is quite straightforward and is similar to the one used in the original RELAX program[2].

Finally the RELAX2 program feeds the circuit lumps to a standard SPICE-like[6] circuit simulator which is capable of handling circuits with an arbitrary number of nodes. This simulator uses a first order predictor-corrector numerical integration algorithm (Backward Euler) with local truncation error timestep control[7]. Because MOS transistors, grounded voltage sources, and capacitors are usually the only elements used in MOS digital circuits, the simulator uses nodal analysis rather than the more complicated and more general modified nodal analysis[8]. Each circuit lump is simulated in the order determined by scheduling algorithm, and the entire schedule is repeated until the node voltage waveforms for each of the subcircuits converges.

3. HANDLING CIRCUITS WITH LOGIC FEEDBACK LOOPS IN RELAX2

Digital circuits can be broken up into two very broad classes, circuits with logic feedback loops (finite state machines, asynchronous circuits, digital oscillators) and circuits without logic feedback loops (most combinational logic, programmable logic arrays). Our experience simulating MOS digital circuits using RELAX2 shows that most MOS digital circuits without logic feedback loops converge in less than ten iterations. However, circuits with logic feedback loops may take many more iterations to converge, and the number of iterations required is proportional to the length of the simulation interval. In this section we examine the problem of circuits with logic feedback. We start with a simple circuit to illustrate the feedback problem; logic feedback is then defined precisely; and finally the waveform relaxation algorithm is applied to a simplified linear system to provide insight into the problem and to motivate a solution.

We used the WR algorithm to decompose and simulate the circuit in fig. 3.1, cross-coupled nand gates, which contains a tight logic feedback loop. (This was done to demonstrate the difficulty of simulating circuits with logic feedback using waveform relaxation. Normally a small tightly coupled circuit would not be decomposed). The node voltage waveforms of this circuit are graphed in figures 3.2a,b,c at three different iterations of the waveform relaxation. The graphs demonstrate a unique property of the WR algorithm when applied to circuits with logic feedback: the error is not reduced at every time point in the waveform. Instead, each iteration lengthens the interval of time, starting from zero, for which the waveform is correct.

The above behavior is consistant with the convergence theorems for the WR method. The convergence of the WR method was proved in the following norm on function spaces

$$\max_{[0,T]} e^{-vt} \|f(t)\|$$

where $v > 0$, $f(t) \in \mathbb{R}^n$, and $\| \ \|$ is any norm on $\mathbb{R}^n$. Note that $\|f(t)\|$ can increase as e^{vt} without increasing the value of this function space norm. If f(t) grows slowly, or is bounded, it may be possible to reduce the function space norm by reducing $\|f(t)\|$ on some bounded interval of t, where this interval increases as the function space norm decreases. The waveforms in the above circuit converge in just this way; the function space norm is decreased after every iteration of the WR algorithm because significant errors are reduced over larger and larger intervals of t.

Not all digital circuits converge in this manner, however. Circuits with pass transistors, for example, converge uniformly throughout the interval of simulation (see example below). The difference in the case of circuits with logic feedback is that there is "gain in the loop", which causes any small error to grow very quickly, until it is limited by the power supply rails.

We will now define logic feedback in terms of the following general nonlinear dynamical system of equations that describe the digital circuit.

$$f(\dot{x}(t), x(t), u(t)) = 0 \qquad x(0) = x_0 \qquad (3.1)$$

where x(t) $\in \mathbb{R}^n$ on $t \in [0,T]$; u(t) $\in \mathbb{R}^r$ on $t \in [0,T]$, piecewise continuous; and f: $\mathbb{R}^n x \mathbb{R}^n x \mathbb{R}^r \dashrightarrow \mathbb{R}^n$ is a continuous map. In our case x(t) is the vector of node voltages of the circuit, and u(t) is the vector of input voltages.

Definition 3.1 Suppose f is Lipschitz continuous in x(t) for all $x(t) \in \mathbb{R}^{nxn}$, then we can define a G $\in \mathbb{R}^{nxn}$ be such that g_{ij} is the minimum value that satisfies

$$\begin{gathered} g_{ij} \| x_j^1 - x_j^2 \| \geq \\ \| f_i(\,(),\ (x_1, x_2, \ldots x_j^1, \ldots x_n)^T,\ (),\ ) \\ f_i(\,(),\ (x_1, x_2, \ldots x_j^2, \ldots x_n)^T,\ (),\ ) \| \\ \text{for } all\ x_j^1,\ x_j^2 \in \mathbb{R} \end{gathered} \tag{3.2}$$

Let L be a lower triangular matrix, and U a strictly upper triangular matrix such that $L + U = G^*$, where G^* is any permutation of G. A circuit with logic feedback is defined as one in which there exists no G^* (defined above) such that

$$l_{ii} > u_{ij} \text{ for } all\ i,j \in [1,\ldots,n] \tag{3.3}$$

Remark 3.1 The matrix G describes, in the worst case, how tightly the nodes of the dynamical system are tied together. The "gain" in a logical feedback loop would produce large symmetric off-diagonal terms in G, and both these off-diagonal terms could not be forced into the lower triangular matrix by reordering the rows in G. ■

In order to gain some insight into the behavior of the WR algorithm on nonlinear circuits with logic feedback, we will analyze a linear system for which we can prove some theorems. Consider using the Gauss-Seidel WR algorithm to solve a linear circuit which has grounded capacitors at every node. The equations for the system are

$$\dot{x}(t) = -C^{-1}Gx(t) + Bu(t) \qquad x(0) = x_0 \tag{3.4}$$

where x(t) $\in \mathbb{R}^n$ on $t \in [0,T]$; u(t) $\in \mathbb{R}^r$ on $t \in [0,T]$, piecewise continuous; C,G $\in \mathbb{R}^{nxn}$; B $\in \mathbb{R}^{nxr}$. C is a capacitance matrix; G is a conductance matrix; and B is an input matrix. In this case we assume C is diagonal, therefore this linear system does not include floating capacitors. The equation of the WR algorithm iteration for this system is

$$\dot{x}(t)^{k+1} = Lx(t)^{k+1} + Ux(t)^k + Bu(t) \quad x(0) = x_0 \tag{3.5}$$

where L is a lower triangular matrix, U is a strictly upper triangular matrix and $L + U = -C^{-1}G$. We have the following theorem[10]:

Theorem 3.1: If the diagonal terms of L are strictly negative[1] then we have the following bound

$$\max_{[0,T]} \| x^{k+1}(t) - x(t) \| \leq \tag{3.6}$$

$$(1 - e^{-vT})\,(1/v)\,cond(S) \|U\| \max_{[0,T]} \| x(t)^k - x(t) \|$$

where $\| \ \|$ is the L_∞ norm on $\mathbb{R}^n$ or the induced L_∞ norm on $\mathbb{R}^{nxn}$, v is the absolute value of the least negative diagonal element of L, cond(S) is the condition number of the matrix of eigenvectors of L, x(t) is the solution to equation 3.4, and x^k, $x^{k+1}(t)$ are the results of the k and k+1 iterations of the WR algorithm (equation 3.5).

Definition 3.2 Define K(T) by

$$K(T) = (1 - e^{-vT})\,(1/v)\,cond(S) \|U\|.$$

Then if K(T) < 1 we say that the WR algorithm uniformly decreases the maximum error over the interval [0,T] at each iteration.

Corollary 3.1: If the diagonal elements of L are negative then there exists some $T^* > 0$, such that $K(T^*) < 1$.

Remark 3.2 Corollary 3.1 follows immediately from the fact that $(1 - e^{-vT})$ goes to zero as T goes to zero. ■

Corollary 3.2: If $(1/v)cond(S)\|U\| < 1$ then at each iteration the WR algorithm uniformly decreases the maximum error over the interval $[0, \infty)$.

Remark 3.2: Consider the shift register example in figure 3.3a This is an example of a system for which the maximum error of the WR algorithm decreases uniformly over the interval $[0, \infty)$. As the circled regions of figure 3.3b and 3.3c show, the errors of the first iteration (3.3b) are reduced throughout the waveform in the second iteration(3.3c). ■

If the circuit described by equation 3.4 has logic feedback according to definition 3.1, then no matter how the equations of 3.4 are reordered, the assumptions of corollary 3.2 will not be satisfied and the WR algorithm will not converge as described in definition 3.2 over the interval $[0, \infty)$. Given corollary 3.1, we can find a T^* so that the WR algorithm will converge as in 3.1 for the interval $[0, T^*]$ (Note that this T^* may be quite small, and it is inversely proportional to how tightly coupled the circuit is). This suggests that the interval of simulation should be broken into "windows", so that the relaxation will converge as in definition 3.2 over the entire window. If we reconsider the cross-coupled nand gate circuit mentioned above, and "window" the simulation using RELAX2, convergence is quite rapid (see table 3.1). There is a trade-off, however, as table 3.1 shows. As the window size gets smaller some of the advantages of waveform relaxation are lost. One cannot take advantage of a digital circuit's natural latency over the entire waveform, but only in that window; the scheduling overhead increases when the windows become smaller, as each circuit lump must be scheduled once for each window; and if the windows are made very small, timesteps chosen to calculate the waveforms will be limited by the window size rather than by the local truncation error, and unnecessary calculations will be performed.

4. SPEED-UP TECHNIQUES BASE ON CHANGING CALCULATIONS WITH THE ITERATION IN RELAX2

When using iterative decomposition methods for solving systems of nonlinear equations, it may be possible to reduce the calculations required by not solving the decomposed nonlinear equations exactly at each iteration. In some cases the convergence of the algorithm is not affected by the inaccurate solutions. In the Gauss-Seidel-Newton method[9], for example, a system of nonlinear algebriac equations is solved by decomposing the system into nonlinear equations in one unknown. But, at each iteration these equations are only solved approximately by performing one iteration of the Newton-Raphson method. Yet it has been shown that if the Newton-Raphson and the Gauss-Seidel methods will converge for a problem when applied independently, the mixed Gauss-Seidel-Newton method will also converge[9]. We applied this idea to the WR method for solving MOS digital circuits. In our case we use simpler approximate methods for calculating the node waveforms for the first few iterations, and switch to more complex and more exact methods for the last few iterations. The convergence of this class of methods for WR has been proven[2].

One way of simplifying the calculation of the node waveforms is to use a simple model for the MOS devices, and then switch to the Shichman-Hodges model as the waveforms approach convergence. Our simple device model is a resistor in series with a switch, where the size of the resistance is scaled with the device size. In the SPICE program[6] the Shichman-Hodges model is referred to as the level one MOS device model, so we refer to our simple model as the level zero model. Using such a model in the calculation of waveforms is not straightforward, because the equations describing the model can not be solved easily using the Newton-Raphson method. The Newton-Raphson method often gets "caught"; that is it will oscillate about the point where the level zero model's switch changes state. One solution to this problem is

[1] The diagonal terms of L are strictly negative if the linear circuit described by the conductance matrix G, and the capacitance matrix C, has only positive conductances, and some positive capacitance to ground at each node.

not to carry out the Newton-Raphson method to convergence but to do only one iteration. The result is that the calculation of the waveforms using the level zero model is quite fast, but only approximate, even if the level zero model is assumed to be correct. The results using the level zero model and then switching to the level one model were dissappointing (table 4.1). In circuits without logic feedback, the level zero model did not provide a better guess for the waveforms than one iteration using level one models. It is possible that we need to add another term to our level zero model to make it smoother. Then we can use the Newton-Raphson algorithm, and achieve the accuracy required to produce a useful first guess for the iterations using the level one models. It is likely however, that then much of the level zero speed advantage will be lost.

Another approach to simplifying the calculations performed in the first few iterations of the WR algorithm is to allow the numerical integration algorithm, which is used to solve for the node waveforms of the decomposed circuit lumps, to use a larger local truncation error. Here, unlike changing the device models, it is possible to increase the accuracy of the calculation of the node waveforms at each iteration by screwing down the local truncation error. In our case, since most of our circuits converge in about 5 iterations, we pick a local truncation error that is about 3 times larger than the local truncation error we would chose to calculate the waveforms in our final answer. Then after each iteration the local truncation error is multiplied by 0.7. The results from this approach were more pleasing (table 4.1).

5. CONCLUSIONS

Several features of the RELAX2 program have been presented which allow the program to simulate a broad class of MOS digital circuits. We are in the process of linking RELAX2 to VLSI graphical editors to allow us to test RELAX2 on circuits generated by the Berkeley IC designer community.

ACKNOWLEDGEMENTS

This research has been sponsored by DARPA under contract NESC-N39, a grant of the IBM company and MICRO.

The authors gratefully acknowledge the work of Ekachai Lelarasmee, who both contributed the original algorithm and suggested most of the improvements for RELAX2. We also wish to thank J. Kaye, J. Klechner, R. Newton, and R. Saleh for many valuable discussions and suggestions.

REFERENCES

[1] E. Lelarasmee, A. E. Ruehli, A. L. Sangiovanni-Vincentelli, "The waveform relaxation method for time domain analysis of large scale integrated circuits," *IEEE Trans. on CAD of IC and Syst.*, Vol. 1, n. 3, pp.131-145, July 1982.

[2] E. Lelarasmee, "The waveform relaxation method for the time domain analysis of large scale nonlinear dynamical systems", Ph.D. dissertation, University of California, Berkeley.

[3] E. Lelarasmee and A. Sangiovanni-Vincentelli, "Relax: a new circuit simulator for large scale MOS integrated circuits", Proc. 19th Design Automation Conference, Las Vegas, Nevada, pp. 682-690, June 1982.

[4] J. Kaye and A. Sangiovanni-Vincentelli, "Solution of piecewise linear ordinary differential equations using waveform relaxation and Laplace transforms", *Proc. 1982 Int. Conf. on Circ. and Comp.*, New York, Sept. 1982.

[5] H. De Man, "Mixed-Mode Simulation for MOS-VLSI: Why, Where and How?" *Proc. 1982 Int. Symp. on Circ. and Syst.*, pp. 699-701, Rome, Italy, May 1982.

[6] L.W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Electronics Research Laboratory *Rep. No. ERL-M520, University of California, Berkeley, May 1975.*

[7]R. K. Brayton, F. G. Gustavson, and G. D. Hactel, "A New Efficient Algorithm for Solving Differential-Algebriac Systems using Implicit Backward Differentiation Formulas", *Proceedings of the IEEE, Vol. 60, No. 1, January 1972.*

[8] C. Ho, A. Ruehli and P. Brennan, "The modified nodal approach to network analysis", *IEEE Trans. on CAS*, vol. CAS-22, pp. 504-509, June 1975.

[9] J. M. Ortega and W.C Rheinbolt, *Iterative Solution of Nonlinear Equations in Several Variables* Academic Press, 1970.

[10] J. White, "RELAX2, a generalized program for the simulation of MOS digital circuits using waveform relaxation methods, M.S. thesis, University of California, Berkeley.

CROSS-COUPLED NAND GATES			
# Windows	# Timepoints	Max # Iterations	CPU time*
SPICE	--	--	21.75
1	50	>100	????
2	50	10	13.59
4	48	4	4.48
8	59	4	5.38
16	69	4	5.91

*On a VAX 11/780

TABLE 3.1

TEST CIRCUITS						
Method	Shift Cell		Two Phase Clk		Memory Cell	
	# Iter	Time	# Iter	Time	# Iter	Time
SPICE	--	12.52	--	43.13	--	13.63
RELAX2	4	2.10	4	5.47	4	2.98
Level0	4	3.20	4	8.75	4	4.45
Level0 only	4	0.49	4	0.69	4	0.88
LTE	5	1.24	3	3.81	4	2.71

*On a VAX 11/780

TABLE 4.1

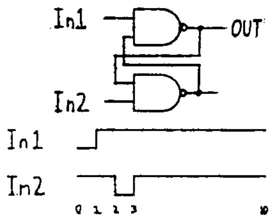


Fig 3.1

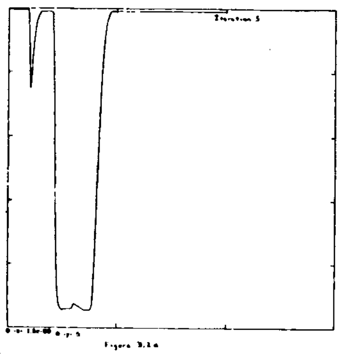


Figure 3.2a

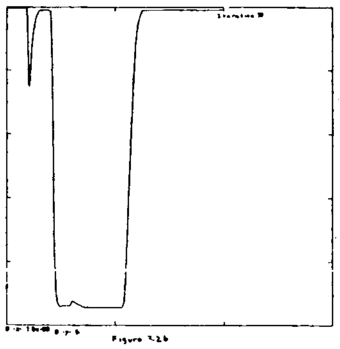


Figure 3.2b

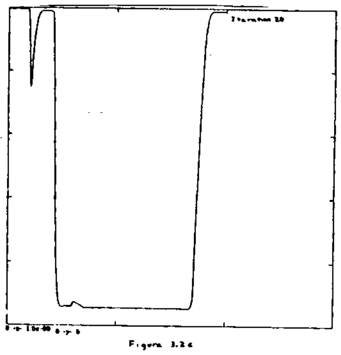
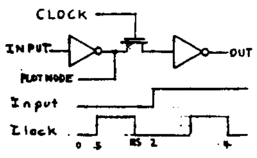


Figure 3.2c



. Figure 3.3 a

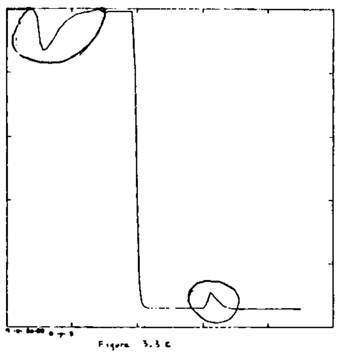


Figure 3.3c

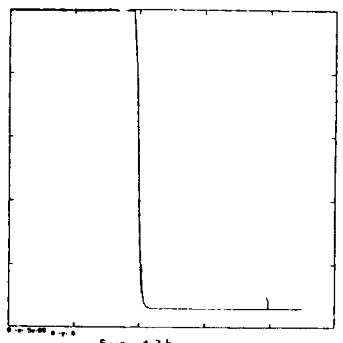


Figure 3.3b