A Large Scale MOSFET Circuit Analyzer Based on Waveform Relaxation.

J.F.Beetem, P.Debefve[1], W.Donath, H.Y.Hsieh, F.Odeh, A.E.Ruehli, J.White[2], P.K.Wolff,Sr.

IBM T.J. Watson Research Center

Yorktown Heigths, NY 10598

ABSTRACT

Recently, it has been shown that the waveform relaxation method is a very promising approach for the analysis of large MOSFET circuits. Also, a key problem in the design of VLSI chips is the verification of the timing waveforms. In this paper, we describe a progı ım for the hierarchical analysis of these large scale circuits. The program employes the most recent advances made in the waveform relaxation techniques. The input and output are hierarchical employing graphics so that circuits can be analyzed interactively.

1. INTRODUCTION

The verification of the timing waveforms is a key problem for VLSI chips especially for high performance technologies. A timing verification tool must be able to accommodate arbitrary connections of transistors (e.g. pass transistors circuits) [1] since they may lead to very efficient designs. A modern program like the one described in this paper must accept hierarchical circuit descriptions and the use of graphical input and output is mandatory. Also, the efficiency of the analysis is very important since an ever increasing number of transistors are involved in the chips and designers are interested in the verification of larger portions of a circuit. Approximate timing simulators [e.g.2] can sometimes be used since they offer a speed advantage over the analyzers. However, the uncertainty of the accuracy of these tools makes it very desirable to have an accurate timing analyzer at least as a backup for the timing simulator. We distinguish between a simulator and an analyzer where we assume that a simulator involves some heuristic guesses while mathematically self consistent numerical analysis techniques are employed in the analyzer.

The waveform relaxation method has recently been shown to be an effective method for the exact analysis of MOSFET circuits. Considerable progress has been made in the last three years since the first experimentation with the waveform relaxation (WR) method [3]. One of the main motivations for its discovery was the desire to analyze large scale circuits globally in time rather than by the the global incremental approach. The problem with the global incremental approach is that **all** subcircuits must be analyzed for each time step until the final time is reached. This approach puts undue restrictions on the analysis of large scale circuits [4]. However, it is important to note that the incremental approach is appropriate for a general purpose circuit analysis programs like Spice [5] or ASTAP [6]. The WR technique has been proven to converge for realistic circuits [7] and practical implementations [8,9] have shown the usefulness for the analysis of large scale MOSFET circuits. An example of a speedup factor of a WR program over SPICE is a factor of 64 [7]. However, this is a function of many parameters like circuit size and type, devices, models etc., and is only an indication of the actual gain.

In this paper, we are reporting on a large scale MOSFET circuit analysis program which implements the latest WR techniques. Section 2 of this paper addresses the system overview and points out the new algorithm used. The software implementation as well as some algorithms are described in Section 3 and some experimental results and experiences with our program are presented in Section 4.

[1] on leave from the Swiss Federal Institute of Technology, 1007 Lausanne, Switzerland.
[2] EECS Departement, University of California, Berkeley, CA 94720

2. SYSTEM OVERVIEW

The time domain circuit analysis involves the solution of a system of differential equation describing the circuit, mathematically formulated as

$$\begin{aligned} F(\dot{y}, y, u) &= 0 \\ y(0) &= y_0 \end{aligned} \qquad (1)$$

where the dot denotes differentiation with respect to time and y and u usually represent nodal voltages [7]. The system described by (1) is decomposed into M decoupled subsystems:

$$\begin{aligned} F_i(\dot{y}_i, y_i, \tilde{u}_i) &= 0 \\ y_i(0) &= y_{0i} \end{aligned} \quad 1 \leq i \leq M \qquad (2)$$

where the $\tilde{u}_i$ are the decoupling inputs (loading currents and/or voltages $\dot{y}_j y_j$ with j≠i). During each iteration, each subsystem is solved for its assigned variables over the given time interval [Tstart, Tstop] by using the approximate vectors u . The relaxation process is carried out among the subcircuits until satisfactory convergence is achieved. An acceleration of the convergence is obtained with the use of the latency detection [10] for each subsystem and also by allowing each decomposed subsystem to have more than one unknown [9,10]. This corresponds to a block relaxation method. In that case, the size of the subsystems is carefully chosen and will be detailed later. Modified versions of latency and block relaxation methods are included in our program. Features like automation of some procedures and "easy-input" are implemented in order to facilitate the use of the program by relatively inexperienced users and to make it "user friendly".

The **input** to the program can be accomplished either via a graphics representation of the schematic diagram or a language description. There are many advantages to graphical input : it is natural, accurate, easily maintained and is self documenting. The data representation is hierarchical since this is an efficient way to represent large circuits [11].

We partition a system into a hierarchy of **modules.** There are three kinds of modules:

Primitives :	indivisible circuit elements such as resistors, capacitors, and transistors,
Leaves :	circuits containing only instances of primitives,
Composites:	instances of leaf and/or composite modules, which may also contain primitives.

Medium and large digital circuits consist usually of many subcircuits which can be designed automatically from a transistor list specification extracted, for example, from IC mask data. Indeed, the single node decomposition , suggested by the Canonical Waveform Algorithm [7], does not represent the most efficient strategy for these circuits. For this reason, special algorithms have been developed for the **partitioning** of the circuits into subsystems (subcircuits) which can be analyzed efficiently using a small incremental program.

The waveform relaxation method is then employed to compute the waveforms between the subcircuits while a new **scheduling** algorithm determines which subcircuit or cluster of subcircuits must be analyzed next. Although, the scheduling process is not strictly necessary to guarantee the convergence of the waveform algorithm, it has a profound impact on the convergence speed. For example, if a chain of N inverters analyzed in a backward order will require at least 2*N waveform iterations, the scheduling from input to output (forward direction) may require only 2 iterations.

The three kinds of modules (composite, leaf, primitive) are analyzed in a different manner. Primitives are only analyzed if they are part of a leaf or composite module since they are not interesting by themselves. Leaves are analyzed using a **small circuit analyzer (SCA)** shown in Figure 1and the result is a set of updated waveforms. These new waveforms will be used to update other waveforms in the system using WR.

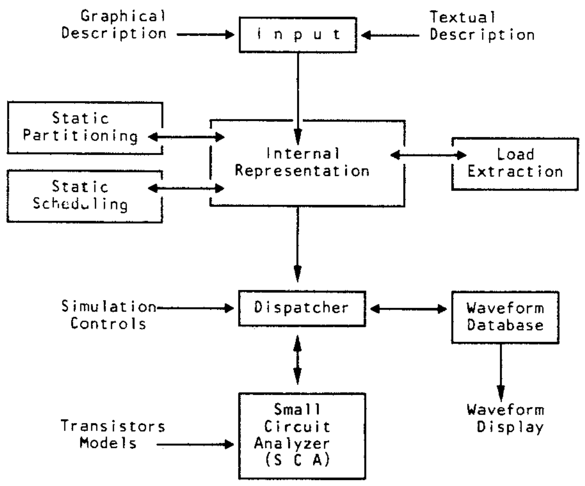


Figure 1. Program Structure.

Composite modules are analyzed hierarchically using a recursive algorithm. The simple algorithm for composite simulation is to analyze each of its non primitive components in some predefined order specified by the Scheduler. Components which are leaves are analyzed using the leaf rule just described. Components which are themselves composite are analyzed by applying this rule recursively, i.e. by analyzing each of its non primitive components. Primitives contained in composite modules are handled as loads.

Load analysis is done in two parts: a static load extraction before actual analysis, and dynamic load generation, done by a Dispatcher during the analysis. The inaccuracies generated by separate analysis of components are removed by successive iterations of WR.

The performance of the program may be limited by the amount of main storage available for the program data and waveforms since frequent disk swapping is required. The conventional approach is to keep the last two waveform iterations in main memory so that convergence of the solution for each node can be tested, or

$$\max_{[t \in (0, \mathrm{TMax})]} \left| V^{I}(t) - V^{I-1}(t) \right| \leq \varepsilon_{tolerance} \quad (3)$$

where $V^{I}(t)$ and $V^{I-1}(t)$ are the latest and previous voltage waveforms and I is the iteration number. Main storage must be conserved since the waveform for each node may require several kBytes. The **adaptive waveform algorithm** [12] removes this restriction. The fundamental idea behind the adaptive strategy is to analyze the chain of leaf modules by subsets where a subset is composed of a few successive leaf modules. This subset is analyzed until the waveforms converge. These subsets are chosen like a "sliding window in space" covering subportions of the circuit sequentially. Only the subset components and the corresponding waveforms are kept in main storage. With this scheme, the limit on the circuit size is removed and the maximum circuit depends now on the disk capacity. The output waveforms are available for all nodes, once convergence of the waveforms is achieved.

The program accepts CMOS as well as NMOS transistor circuits. Each technology is described completely in independent segments which are plugged into the program as they are needed. This modularity allows the use of different technology files.

3. SYSTEM IMPLEMENTATION

The software structure of the program is in the form of modular program segments as shown in Figure 1. The program is written in Pascal [13] and each segment is compiled separately. As a consequence, an upgrade done in one segment does not implicate a recompilation of the whole program.

3.1. Input Phase :

Circuit topologies are entered into the program primarily using an interactive graphics editor (GE). The GE is a fast and powerful tool for entering circuit diagrams. It extracts circuit topology as a diagram is being entered or changed, and has many features to help the designer to enter a correct circuit. The editor fully supports hierarchical design, allowing the designer to specify the logical partitioning of a system at entry time. The GE makes it easy to expand a component library and create new symbols for circuit elements. The user interface has been optimized for efficient entry and editing of diagrams.

Although the GE is the best way for designers to enter circuit diagrams, there are cases where the circuit topology is not entered by hand, e.g. topologies extracted automatically from IC mask data. To support these cases, we use a textual interconnect Pascal like language. An example is reproduced at Figure 2. Like the GE, it supports hierarchy. A system can be described using any combination of GE diagrams and automatically generated language statements.

3.2. Preprocessor :

To improve the speed of convergence and exploit the structure of the class of circuits to be analyzed (MOS digital circuits), a reordering of the circuit is performed in the preprocessor phase. Debugging functions as connectivity and referencing checks are also executed at this stage.

The reordering occurs in two steps : first, a static partitioning of the circuit into Strong Connected Components (SCC) [14], and second, a static ordering (called scheduling) of the elements in accordance with the signals flow from inputs to outputs through the circuit.

A static load extraction is also performed in the preprocessing phase: it consists of creating a load model of the inputs of a module.

3.2.1. Static Partitioning :

The basic idea of the partitioning is to subdivide the circuit into small pieces: the break is done at the "high impedance branches" and voltages sources since a cut at the "low impedance" points results in a slower convergence. These pieces are called Strong Connected Components and are composed of elements (primitives) connected by drain-source or resistor paths and at least one element connected to the voltage supply. The automatic partitioning of a transistor list is done by the procedure **Partition**. This procedure divides the list into a leaf module list by applying an Union-Find algorithm [15] following the SCC rule. The circuit has now a two level representation : the first is a global description of the interconnections between the INPUTS and OUTPUTS and the leaf modules, the second is the description of the contents and the interconnections between the elements (primitives : capacitances, resistances and transistors) inside each leaf module.

As an alternative, the user may subdivide the circuit hierarchically by hand via the GE. This partitioning is usually based on logical elements. However, to insure that this partition is efficient for the WR algorithm in terms of convergence and speed, the procedure **CheckPartition** will review the proposed partitioning and, if necessary, modify the data structure according to the SCC algorithm.

In both cases, the result is a hierarchical description of the circuit. The modifications done to the data structure are completely transparent to the user.

3.2.2. Static Scheduling :

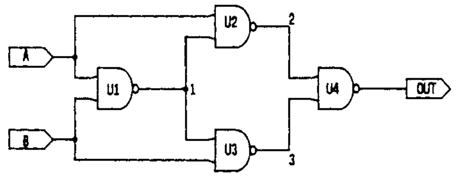
In general, the internal description of the circuit reflects an arbitrary order for the lists of components of a module. The order in which modules are processed during the analysis is determined by a procedure called **Scheduler**.

This scheduling assumes that the leaf cells and the composites cells are one-way models i.e. there is a strong signal flow from inputs to outputs. The basic algorithm applied for a module is :

```
Repeat {
  visit the list of unscheduled elements;
  if (ALL input nodes of an element are scheduled)
      then push it in the Stack of scheduled elements
          mark output nodes as scheduled; }
Until (Stack is full);
```

The program starts with the root which is the highest level in the hierarchy and schedule it by applying this algorithm. Then, the resulting ordered list is visited starting from the first cell and the algorithm is repeated recursively in a depth first search manner. Each module is scheduled only once. The last level scheduled is the leaf module level. The program stops after everything is scheduled.

A feedback loop is detected when none of the remaining elements could be scheduled and the stack is not full. In that case, the program calls a procedure called **FeedbackSolver** which has two functions : first, to find the exact composition of the feedback loop by applying the SCC algorithm on a directed graph where the dominator is imposed by the scheduler procedure itself, and second, to push the elements of the strong connected component in an *ordered* (levelized) manner in



```
/* Global signals... */
%global GND,VDD;
/* Primitive modules for this example */
%define %prim EFET(=D,G,=S,=B-GND,&W,&L);
%define %prim DFET(=D,G,=S,=B-GND,&W,&L);
%define %prim CAP(=A,=B,&C);
/* Code generated for a NAND module   */
%define O = NAND(A,B);
T1:EFET(O,A,1,*,W=WE,L=LE);
T2:EFET(1,B,GND,*,W=WE,L=LE);
TL:DFET(VDD,O,O,*,W=WD,L=LD);
C2:CAP(1,GND,C=C1);
C1:CAP(O,GND,C=C2);
/* Code generated for a XOR module   */
%define OUT = XOR(A,B);
1 = U1:NAND(A,B,WD=2,LD=4,WE=4.1,LE=1,C1=0.5,C2=0.5);
2 = U2:NAND(A,1,WD=3,LD=5,WE=4.2,LE=1,C1=0.5,C2=0.5);
3 = U3:NAND(1,B,WD=3,LD=5,WE=4.1,LE=1,C1=0.5,C2=0.5);
OUT = U4:NAND(2,3,WD=2,LD=4,WE=4.2,LE=1,C1=0.5,C2=0.5);
```

Figure 2. Example of a graphical and textual representation for a XOR

the stack. Special references are given on the existence (Feedback flag) and size of the feedback loop. Those references will be used by the dispatcher. The complete algorithm is outlined in the Figure 3.

3.2.3 Load Circuit Extraction :

For composite modules, it is necessary that components are analyzed in the correct electrical context, i.e. electrical loading effects be properly modeled.

Static load circuit extraction consists of creating a load model of the inputs of a module. The extracted loads are identical for all instances of a module, thus reducing storage and program execution time. The static load model of a leaf module is simply those components strongly connected to an input of the module. The static load model of a composite module consists of primitive components strongly connected to an input and also all loads of any non primitive components which are strongly connected to an input of the composite module (Figure 4).

When generating load models, it is necessary to process the modules in a reverse hierarchical order, i.e. a module must have its loads extracted before any composite module is instantiated.

```
Procedure Scheduler( Module );

 Begin
  Mark Input nodes of this Module;
  Open a Stack for the scheduled elements;
  Repeat
    { IF (ALL input nodes of an element
                          are marked)
        then
          Mark all output nodes for this element;
          Push it in the stack;
      IF (Feedback detected)
        then  FeedbackSolver;
    }
  until (all elements are stored in the stack);

  for (each element in the stack) do {
     IF (it is a composed element)
        then (if the corresponding Composed
              Module is not scheduled)
              then Scheduler(Composed Module); }
End;
```

Figure 3. Scheduling Algorithm.

3.3. Hierarchical Circuit Analysis :

3.3.1. Dispatcher :

The hierarchical analysis is controlled by the **Dispatcher**. The Dispatcher has three main tasks:

1. It selects which module(s) should be analyzed next, as specified by the static scheduler, the feedback flag, and the adaptive waveform algorithm.
2. It manages the **waveform database**: it selects which waveforms must be in memory for a leaf analysis (procedure WDBREAD), and updates (procedures WDBALLOC and WDBWRITE) the database with new waveform values at the end of a leaf analysis.
3. It generates dynamic loads for a circuit being analyzed.

To analyze a leaf module, the Dispatcher must call the Small Circuit Analyzer (SCA). Before it does, it makes sure all necessary node waveforms are in memory by fetching them from the waveform database. In addition, it adds appropriate load models to the circuit. The resulting augmented model is a complete description of the leaf circuit and its environment, which the SCA can analyze as a complete circuit. The result of the SCA is that some of the waveforms have new values closer to the final solution. The Dispatcher stores the updated waveforms back in the waveform database.

Analyzing a composite module requires close attention to dynamic loads. To analyze a composite module, the Dispatcher analyzes each of its non primitive components in the order specified by the scheduler. (This analysis is done by calling the Dispatcher recursively.) The load model of a given component **x** consists of the following :

1. Any primitive components of the composite module which are strongly connected to an output of **x**.
2. Any static loads of any non primitive components which are strongly connected to an output of **x**.
3. Any external loads of the composite module which are strongly connected to an output of **x**. These external loads appear when the Dispatcher calls itself.

The dynamic loads become the external loads of module **x** when the Dispatcher is called to analyze component **x**. If **x** is a leaf, then the unloaded circuit for **x** is augmented by the loads, all waveforms in **x** and its environment (including all waveforms connected to the loads) are fetched from the waveform database, and the SCA is called. If **x** is composite, then it is analyzed by successive simulation of its components.

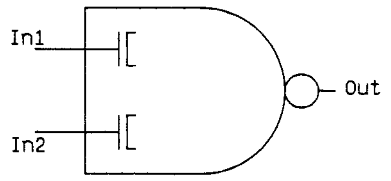


Figure 4. Load model for a NAND gate.

The new **A**daptive **W**aveform **A**lgorithm (**AWA**) is applied to define a small collection of independent and successive subcircuits (subset) and to analyze them until the waveforms have converged. First, a subset is defined starting from the first leaf module to be analyzed; the size is determined by the error tolerance $\varepsilon(T)$. This subset will act as a "sliding window" along the chain. The waveform analysis is performed for the first subset until waveform convergence occurs for the first module. Then, these waveforms are stored on disk and discarded from the main storage as well as the first module itself. The next module following this subset is loaded afterwards into the main storage to define a new subset ready for analysis. When the Dispatcher detects the flags created by the Scheduler for a feedback loop, a decision is made about the function and the type of feedback. For a weak feedback, the dispatcher will collapse the constituting leaf modules into one module, while for a global feedback, the time window is adjusted to the clocking scheme.

Nevertheless, the size of a leaf module or a consortium of leaves may be larger than the size which is efficient for a full matrix solution. The program has an automatic switch to activate a sparse matrix solution in the SCA for this specific module. This procedure is repeated until all leaf modules are analyzed.

3.3.2. The Small Circuit Analyzer (SCA) :

The mission of the SCA is to compute the waveforms for a leaf module or a composite module. The SCA is located at the lowest level of the hierarchical structure as shown in Figure 1. The activation of the SCA is completely controlled by the Dispatcher. Consequently there is a "master" and "slave" relationship between the Dispatcher and the SCA.

The software structure is in the form of modular program segments, the calling sequence is shown in Figure 5. A separate modular segment is used for each technology.

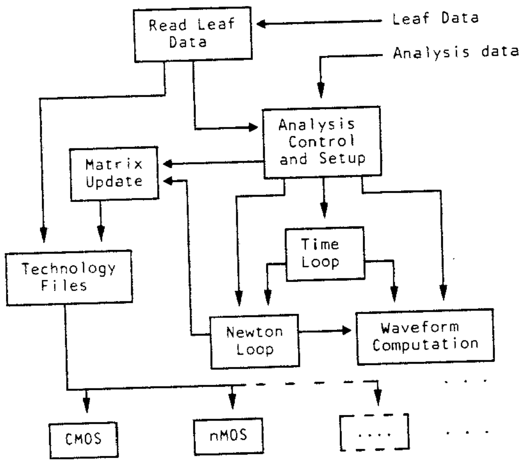


Figure 5. Small Circuit Analyzer (SCA)

The leaf module matrix equation is formulated in terms of a nodal type analysis approach that yields N equations in N unknown node voltages. To this end, all the assumptions for the nodal analysis are also applied here. However, for the sake of integration stability, we choose to use the charge "q" as the state variable for the nonlinear transistor capacitances instead of the nodal voltage "v". i.e.

$$\dot{q} = f(g(v)) \qquad (4)$$

$$q = g(v) \qquad (5)$$

where $\dot{q} = dq/dt$.

A discretization with Gears formula of order r(r = 2 is used) i.e.

$$\dot{x}_{n-1} = \sum_{i=0}^{r} \alpha_i \, x_{n+1-i} \qquad (6)$$

where α_i is the integration coefficient. For equations (4) and (6) :

$$\sum_{i=0}^{r} \alpha_i \, g_{n+1-i}(v) = f(g(v)) \qquad (7)$$

After linearization with Taylors expansion at v^k_{n+1} , equation (7) becomes

$$\left[\alpha_0 \frac{\partial g(v)}{\partial v} - \frac{\partial f(v)}{\partial v} \right] \Delta v = - \sum_{i=0}^{r} \alpha_i \, g^k_{n+1-i}(v) + f(g(v^k_{n+1})) \qquad (8)$$

where $\Delta v = v^{k+1}(t_{n+1}) - v^k(t_{n+1})$.

The major tasks assigned to the SCA is similar to those performed by others circuit analyzer such as SPICE or ASTAP. In an initialization phase, all the analysis controls and technology dependent parameters are assigned from external files to the proper or default values. Some, like the threshold voltages, are supplied in tabular form [18]. This initialization phase is done once and is valid for the entire analysis.

In the analysis phase, each leaf module will be processed in the same manner. The program reads first the leaf module description and all the arrays and the matrix of the leaf module are initialized to zero. The required input voltages are obtained from the records of the Dispatcher. The variables to be solved are then predicted according to the integration algorithm. In this program the 2nd-order Gear's method is implemented. With the predetermined technology to be used, the proper transistor model is called and all the capacitances, currents, and charges are computed as well as the partial derivatives for the Jacobian matrix . Then, taking into account the load elements, the leaf module matrix equations are created. The discretized nonlinear equations of the leaf module are solved by the Newton-Raphson iteration method. All the nonlinear capacitances, currents, charges, and partial de-

	ALU 7 bit	Ring Oscillator	ALU 32 bit	CMOS inverter chain			
				10	50	100	200
ASTAP	578 sec	59 sec	-	61 sec	460 sec	880 sec	-
our program	37 sec	14 sec	80 sec(*)	7 sec	24 sec	32 sec	49 sec

Figure 6. Execution times (CPU) for an IBM-3081.

rivatives are recomputed for each Newton iteration. If the above Newton-Raphson iteration has converged, the local truncation error of the present time step is evaluated. If the local truncation error is within the predetermined bounds, the present time step is, then, accepted. Note that the time step will be reduced if either the Newton-Raphson does not converge or the local truncation error is not acceptable. After the present time step has been accepted, the computed solutions of that time step are stored in the proper waveform arrays for subsequent interactive output display. To this end, the new time step is estimated for analyzing the leaf module until the stop time is reached.

4. SOME EXPERIMENTAL RESULTS

Several circuit examples for both nMOS and CMOS were used for testing our program. All the analysis were performed on an IBM-3081 with both ASTAP and our program. The transistor models are at least as complex as the levels 2 and 3 of SPICE [19]. However, the implementation is somewhat different.
Consequently, the total CPU times shown in Figure 6 can only be considered as preliminary performance indications of the two programs.

The circuit shown in Figure 7 is a 7 bit ALU with a total of 375 transistors and 418 capacitors. An identical result is obtained after 4 waveform iterations with our program in much less time than the ASTAP program required. The second example is a ring oscillator with 9 stages (46 transistors). The improvement in speed for the ring oscillator is not as large as in the first case. This is due to the fact that here the program detects a feedback loop which includes the entire circuit and collapses all the leaf cells into one cell. To this end, the time step is thus the same for all the nodes, and the advantage of independent time steps disappears for this case. On the other hand, without the feedback loop detection, the circuit will be analyzed hierarchically and the CPU time needed to obtain a waveform convergence was 53 sec.

The third example is a 32 bit ALU also using nMOS thechnology and consists of about 1200 transistors and 4000 capacitors. The analysis time is 80 sec for the first waveform iteration. We are not able to run this 32 bit ALU with ASTAP due to excessive storage and run time requirements.

The last example is a chain of CMOS inverters connected in series. We tested with 10, 50, 100 and 200 inverter stages on both programs. As we expected, the CPU time differences is proportional to the number of stages. Two waveform iterations were required for the first CMOS example chain and three for the last one having 200 inverters.

The CPU time savings obtained with this program are mainly due to the carefull implementation of the waveform relaxation algorithm and to the algorithm itself as well. The amount of data stored for an example is directly correlated to the hierarchical representation and has no impact on the speed of this program.

5. CONCLUSIONS

The program presented in this paper enhances the performance of "exact" circuit analyzers for the computation of exact timing waveforms. We have presented a self-consistent scheme which generalizes the techniques to arbitrary MOS circuits including feedback etc. . Hence, it extends the uselfulness of the waveform relaxation method. The users acceptance of a program is determined by many factors like the reliability, transportability and implementation. We found that Pascal/VS [13] provides an excellent environment for the development of such reliable, transportable and readable code.

Acknowledgments

Valuable suggestions have been made by program users which helped us in our task. The contributions of Vojin Oklobodja with his large example and Paul Ledak for his cooperation are especially acknowledged.

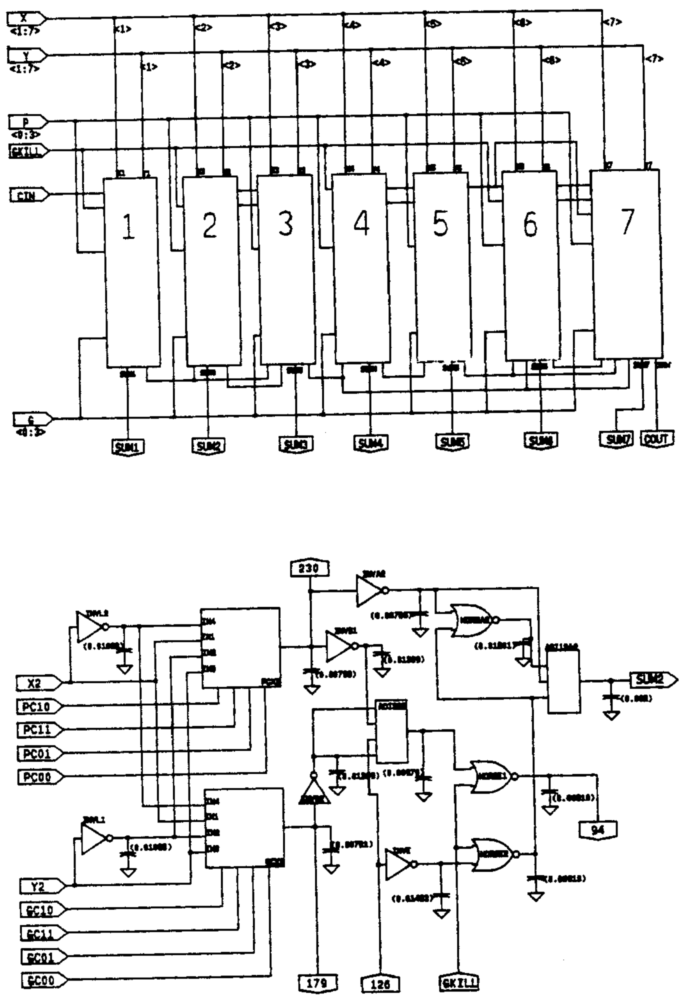


Figure 7. (top)general view of ALU 7 bit, (bottom)internal view of bit 2.

References

1. C.A.Mead, L.A.Conway: "Introduction to VLSI system", Addison-Wesley, 2nd Ed., 1980.
2. C.J.Terman: "RSIM- A logic-level timing simulator", IEEE Proc. Int. Conf. on Computer Design, pp.437 -440, Port Chester, N.Y., Nov. 1983.
3. E.Lelarasmee, A.Ruehli, A.L. Sangiovanni-Vincentelli: "Waveform relaxation decoupling (WRD) method", IBM Technical Disclosure Bulletin, Vol.24, No.7B, pp.3720 -3721, Dec. 1981
4. A.E.Ruehli,G.S.Ditlow: "Circuit Analysis, Logic Simulation and Design Verification for VLSI", Proc. IEEE, Vol. 71, pp 34-48, Jan.1983
5. L.W.Nagel: "SPICE2: a computer program to simulate semiconductor circuits", Univ. of California, Berkeley, ERL Memo ERL-M520, May 1975
6. "Advanced Statistical Analysis Program (ASTAP)", Program Reference Manual, Pub. No. SH20-1118-0, IBM Corp. Data Proc. Div., White Plains, NY 10604.

W.T.Weeks, A.J.Jimenez, G.W.Mahoney, D.Mehta, H.Quassemzadeh, T.R.Scott : "Algorithms for ASTAP - A Network Analysis Program", IEEE Trans. Circuit Theory, vol. CT-20, pp.628- 634, Nov. 1973.

7. E.Lelarasmee, A.E.Ruehli, A.S.Sangiovanni- Vincentelli: "The Waveform Relaxation Method for Time Domain Analysis of Large Scale Integrated Circuits", IEEE Trans. on CAD of int. Circ. and Systems, vol. CAD-1, pp.131-145, July 1982.
8. H.DeMan, "Mixed-Mode simulation for MOS-VLSI : Why, Where and How?", IEEE Proc. ISCAS, pp.699 -701, Rome, May 1982.
9. J.White, A.L.Sangiovanni-Vincentelli : "Relax2 : A New Waveform Relaxation Approach for the Analysis of LSI MOS Circuits", IEEE Proc. 1983 Int. Symp. Circuits Syst., May 1983.
10. N.B.Rabbat, A.L.Sangiovanni-Vincentelli,H.Y.Hsieh: "A Multilevel Newton Algorithm with Macromodelling and Latency for Analysis of Large Scale Nonlinear Networks in the Time Domain", IEEE Trans. Circuits Syst., vol. CAS-26, pp.733-741,Sept. 1979.
11. J.F.Beetem: "Structured design of electronic systems using isomorphic multiple representations", Thesis, Stanford Electronics Laboratories, Stanford Cal. 94305, Dec 1981.
12. F.Odeh, A.E.Ruehli, C.H.Carlin: "Robustness aspects of an adaptive waveform relaxation scheme", IEEE Proc. Int. Conf. on Computer Design, pp. 396 -399, Port Chester, N.Y., Nov. 1983.
13. "Pascal/VS", rel.2.1, Program No: 5796-PNQ, Pub. No. SH20-6168-1 and SH20-6162-1, IBM Corporation, 555 Bailey Av., P.O.Box 50020, SanJose, CA 95150.
14. R.Tarjan: "Depth-First Search and Linear Graph Algorithms," SIAM J. Comput., Vol 1, No 2, pp.146 -160, June 1972.
15. A.V.Aho,J.E.Hopcroft,S.D.Ullman: "The Design and Analysis of Computer Algorithms," Reading, MA : Addison-Wesley, 1975.
16. H.Schichman, D.A.Hodges : "Modeling and Simulation of Insulated-Gate Field-Effect Transistor Switching Circuits" IEEE J. Solid State Circuits, vol. SC-3, pp. 285 -289, Sept.1968.
17. V.B.Rao, T.N.Trick, I.N.Hajj : "A Table-driven Delay-operator to Timing Simulation of MOS VLSI Circuits", IEEE Int. Conf. on Comp. Des., pp. 445 -448, Port Chester, Nov. 1983.
18. H.I.Hanafi, L.H.Camnitz, A.J.Dally : "An Accurate and Simple MOSFET Model for Computer-Aided Design", IEEE J. Solid State Circuits, vol. SC-17, pp. 882 -891, Oct.1982.
19. A.Vladimirescu, S.Liu : "The Simulation of MOS Integrated Circuits using SPICE2", Memo. UCB/ERL M80/7, University of California, Berkeley, Feb.1980.