AN EVENT DRIVEN APPROACH FOR MIXED GATE AND CIRCUIT LEVEL SIMULATION

by

Karen A. Sakallah & Stephen W. Director

DRC-18-45-82

April, 1982

An Event Driven Approach for Mixed Gate and Circuit Level Simulation[1]

K. A. Sakallah and S. W. Director

Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

A new algorithm for mixed gate and circuit level simulation is described. The algorithm is based on a modular view of electronic networks in which individual modules may be described either at the circuit or at the logic level. Consistency is ensured by employing a novel logic gate model which is derived by abstraction from the underlying (and more detailed) circuit model. Computational efficiency is achieved by exploiting temporal sparseness - both for circuit and logic level modules - through the use of event driven techniques. The implementation of the algorithm in the SAMSON program is briefly described and a sample simulation example is presented.

1. Introduction

Computer simulation has become an indispensable tool in the design of very large scale integrated (VLSI) circuits. Traditionally a number of levels ranging from high-level behavioral to low-level electrical descriptions have been successfully used to model and simulate digital electronic networks. In the past modeling and simulation were restricted to a single level of description at any one time. More recently, however, an emerging need for the simultaneous representation of an electronic network at more than one level of description has spawned an intense research effort in multi-level modeling and simulation. Mixed-level simulation, i.e., simulation which simultaneously combines circuit- and logic-level descriptions, has been particularly prominent in this first evolution [1,2].

2. Overview of SAMSON

SAMSON (System for Activity-directed Mixed Simulation Of Networks) is a new mixed-level simulator which harmoniously combines the seemingly disparate techniques of circuit and logic simulation. Two complementary premises help achieve this harmony. The first is that temporal sparseness [3], as an attribute of a dynamic system, is level-independent. This led to the adoption of event-driven simulation techniques, previously limited to the logic-level, as a common framework for mixed-level simulation. The second premise is that the logic and circuit models are *different* representations of the *same* entity and as such have to be compatible. This resulted in the development of a new logic-level model which permits a smoother interface between the circuit and logic parts in a mixed-level network.

2.1. Network Model in SAMSON

SAMSON operates on a network which is modeled as a set of n interconnected subnetworks. Individual subnetworks may be described either at the circuit or at the logic level. Circuit-to-logic and logic-to-circuit signal converters are automatically inserted at the appropriate interfaces. Each subnetwork is considered a dynamic entity whose time-domain response can be represented by a sequence of events. Events are associated with those instants of time at which the subnetwork equations have to be solved. For circuit-level subnetworks, such events correspond to the instants at which the subnetwork equations are discretized (with an appropriate integration formula) and solved (using an iterative scheme such as Newton-Raphson). For logic-level subnetworks, such events correspond to the instants at which the discrete-valued logic equations (which express the subnetwork outputs in terms of its inputs) are evaluated.

2.2. Temporal Sparseness and Exclusive Simulation of Activity

Large networks tend to be temporally as well as spatially sparse. Spatial sparseness reflects the low level of connectivity among distant parts of a network, whereas temporal sparseness reflects the low level of activity in a network at any given point in time. Both types of sparseness can be advantageously exploited in simulation algorithms in order to increase simulation speed. Spatial sparseness is exploited by applying sparse-matrix methods. Temporal sparseness, on the other hand, is exploited by using event scheduling techniques. The exploitation of temporal sparseness, frequently referred to as exclusive simulation of activity (ESA), has been identified with logic simulation in the past [4]. The association of logic simulation with ESA stems, in part, from the simplicity of the logic-gate model which, in turn, allows a simple event-driven implementation. The ESA principle, however, is applicable to large networks regardless of the complexity of their models. In particular, it can be applied to networks described with circuit-level models as we show in the next section. In SAMSON, logic as well as circuit level events are scheduled in precisely the same manner using a noninteger-time indexed-list scheduler [5].

3. Event-Driven Circuit Simulation

The application of the ESA principle to a network composed of n circuit-level subnetworks proceeds by allowing each subnetwork to be integrated with an individually tailored sequence of integration steps. This forces the network equations to be temporally decoupled.

[1] This research was supported in part by the Intel Corporation and by the Army Research Office under grant DAAG29-79-D-1003.

Previous efforts in c\cnt-dri\cn circuit simulation curry out such decoupling in ;»n ad hoc m-inncr. generally by assuming that the coupling, between adj:ivht subnetworks is weak |(\ 2|. 'Iliis approach may Icail to erroneous simulation results or even to instability |7|. In SAMSON, the decoupling of the network equations is based on a rigorous model which takes into account the resulting decoupling errors. 'Hie accuracy of event-driven circuit simulation in SAMSON is. therefore, comparable to thai of traditItm.il circuit simulation regardless of the amount of coupling among different subnetworks. 'IV basic steps of the e\cm-dri\cn circuit simulation algorithm in SAMSON at a given instant of time t" arc as follows. Let A denote the set of subnetworks which ha\c pending circuit-level events at t, and I) denote the set of subnetworks which have events in the future (t > I*). Subnetworks in the set A wilt be referred to as alert and those in the set I) as dormant.

1. 1-lxtrapolatc the outputs of dormant subnetworks.

2. Discreti/c the equations of alert subnetworks.

3. Assemble and solve the equations of alert subnetworks.

4. Check the status yf dormant subnetworks. If any dormant subnetwork should be alerted, transfer it from set D to set A. discrcti/.c its equations and go to Step 3.

5. Calculate, for each alert subnetwork, the truncation errors (TF). If the TF arc smaller than a given tolerance, calculate the size of the next step, and schedule a corresponding event in the future.

Assuming that a k^-ordcr integration formula is used to integrate a dormant subnetwork, the extrapolation of c;ich output signal in Step 1 is done using a (k-I-1)*-order polynomial which depends on the previous (k+1) computed solutions as well as the List computed truncation error [8]. Prediction-1Jascd Differentiation [9] formulae arc used for discretization in Step 2. 'I"he equations in Step 3 arc solved using a Ncwion-Kaphson iteration and Block I.U factorization (10]. The status check in Step 4 is equivalent to a truncation error check on the inputs of dormant subnetworks.

4. The Logic Simulation Model

'Ilic logic-level models used in many existing logic simulators arc essentially the result of a *I op-down refinement* process. Stalling from the concept tif an "ideal" zero-delay boolean gale, such processes typically involve the incorporation of extra signal Mates and various delay assumptions in order to adequately represent "rcaP gates, i.e. gale; which arc constructed from physical devices. It can he argued, however, that *htttom-up abstraction* is a. more natural approach to logic-level iruidciing. especially in the context of mivcd-lcvcf simulation. Using this appioach. the ideal lojjic model is augmented with elements *inferred* from its underlying circuit-level realization. This is in contrast to the top-down process in which such additional elements .ire postulated. •

'Hie logic-gate model used in SAMSON, which results from such an abstraction process is characterized by 4 signal states and a 4-paramcter back-end delay operator. 'Two of the suites (II and L) arc static and correspond to logical truth and falsehood respectively. The other two states (R anfl K) arc d\namk and correspond to a signal in transition between the static states. The delay parameters arc two set-up times ($A_{,,}$ and A_L) and two transition times (A_R and $A_{I:}$) as defined in Figure I. In addition to the pure-delay action character}/cd by these 4 delay parameters, the delay operator has an incrtial component which fillers out a small class of narrow spikes.

'11K most noteworthy feature of the above logic-level model is the absence of an ambiguous or unknown state X commonly employed in logic simulators, Iiy replacing X with the more descriptive R and F transition states many anomalies in existing logic simulators disappear. Furthermore, spikes which arc treated as error conditions in simulators using an X state arc given the more natural interpretation of incomplete transitions.

5. The Mixed-Level Interface

Logic-to-circuit and circuit-to-logic signal converters arc automatically inserted by SAMSON at the appropriate interfaces in a mixed-level network, I-ogic-lo-circuil conversion involves the transformation of a 4-statc logic signal (a sequence of transitions between the suites L, R. H, and I) into a continuous voltage signal. The transformation is accomplished by emitting prc-storcd rising and falling voltage waveforms in response to input state transitions into R and F respectively. Spikes arc generated if the rising and falling waveforms overlap. Circuit-to logic conversion is basically a thresholding operation which transfroms a continous voltage waveform into a discrete sequence of stile transitions. In addition to thresholding, the slope of the voluigc signal in die transition region is monitored to detect spikes and generate appropriate logic suite transitions (R to F or F to R).

6. SAMSON - The User Interface

'ITic basic structure of the SAMSON software is shown in Figure 2. The description of a network to SAMSON consists of two parts: model definitions and model instantiations. Basically, a model is a parametcri/cd multi-terminal structure which serves as a template for creating subnetworks of the' same structure but possibl) diHerein parameters. 'Two kinds of models arc allowed in SAMSON: logic- and tircuit-lcvcl. Within each category models can be specified hierarchically, i.e.. "larger" models cm be constructed from previously defined smaller models. At the lowest lex el, logic models arc specified in terms of boolean equations which compute the value of each output signal in the nuxlcl as a function of the values of its input signals, and a 4-paiamctcr back-end propagation delay operator. For circuit-level models, the primitive is a 2-tcnnin«i1 branch whose branch relationship may be specified by a user-supplied procedure. The more common linear (resistance. capacitance, voltage source.....etc.) and nonlinear (diode) branches arc built-in. Circuit level models arc constructed from interconnections of these primitive branches and any previously defined circuit level models. Kxamplcs of model descriptions in SAMSON arc shown in Figure 3.

Fvcry subnetwork model, whether it be a logic- or circuit-level model is prcproccsscd by SAMSON resulting in a PASCAL[2] solution procedure specific to the model. For a logic-level model the solution procedure evaluates the 4-valued logic function of the input signals for each output terminal. For a circuit-levcl model; the solution procedure includes PASCAL code for loading the coefficients of the Jacobian matrix, performing I.U factorization, and forward and back substitution. 'Iticsc model solution procedures arc then compiled and added to a model library.

[2]SAMSON is written in PASCAL and currently runs cm a VAX-11/780 computer.

'\hc actual network description consists of a sequence of subnetwork declarations. Rich declaration refers to the name of a model of which the subnetwork is an instance. Models can, of course, be cither circuit- or logic-level. 1Tiis establishes an asN(>cintk>n between the subnetwork and the model solution procedure. Of course, different subnetworks which arc instances of the same model share this solution procedure but maintain separate data structures. Figure 4 shows an example network description which references the three models defined in Figure 3. Figures S and 6 show sample simulation commands and simulation results for the network described in Figure 4.

References

1. Do Man. H. and Arnout. C. "ITic Use of Boolean Controlled FJcmcnts for Macro-Modeling of Digital Circuits," *Pruc. IEEE ISCAS.* IR-K, IV>78. pp. 522-526.

2. Newton. A. R.. "Techniques fur the Simulation of l^irgc-St.ilc Integrated Circuits." *ILL'/.' Tunis, Circuits and Systems.* VoL CAS-26. No. 9. September 1979. pp. 741-749.

3. I)c Man. Hugo J., "Computer-Aided Design for Integrated Circuits: Trying to Bridge the Gap," *IEEE Juunial of Solid-Shite Circuits.* Vol. S C R No. 3. June 1979, pp. 613-621.

4. Ulrich. KG.. "Inclusive Simulation of Activity in Digital Networks," *CACM.* Vol. 12, No. 2. February 1969, pp. 102-110.

5. Vauchcr, Jean G. and Duval. Pierre, "A Comparison of Simulation Event List Algorithm*;,*' *Communications of the ACM.* Vol. 18. No. 4, April 197S. pp. 223-230.

6. Chawla, Basant R., Gummcl. Hermann K. and Ko/.ak, Paul, "MOl IS - An MOS Timing Simulator," *IEEE Transactions on Circuits and Systems.* Vol. CAS-22, No. 12, Dec 1975, pp. 901-910.

7. Dc Michcii. Giovanni and Sangiovanni-Vinccntelli, Alberto, "Numerical Properties of Algorithms for the Timing Analysis of MOS VLSI Circuits,* /'w. /W *LX'CTD.* The Hague, The Netherlands, 198L pp. 387-392.

8. Sakallah, Karcm A., *Mixed Simulation of Electronic inu'Kraicd Circuits.* PhD dissertation, Carnegie-Mellon University, November 1981.

9. Van Bokhovcn. W. M. G., "Linear Implicit Differentiation Formulas of Variable Step and Order," *IEEE Trans. Circuits and Systems.* Vol. CAS-22, No. 2, February 1975, pp. 109-115.

10. Sakallah. K.and Director, S. W., "An Activity-Directed Circuit Simulation Algorithm." *I>roc. IEEE ICCC.* IKK1:, 1VS0, pp. 1032-1035.

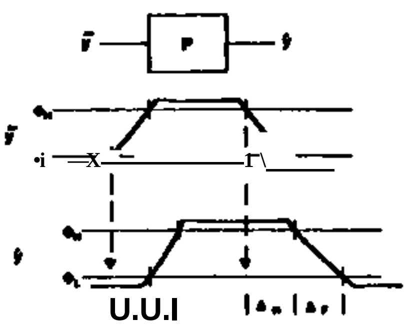


Figure 1: Definition of tlic delay parameters

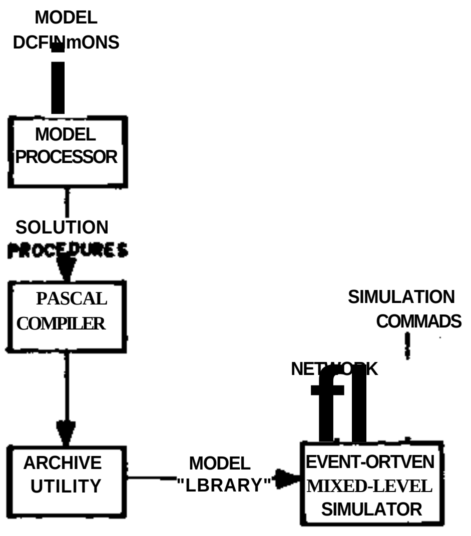
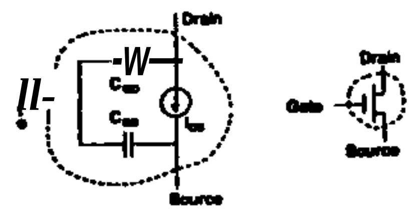
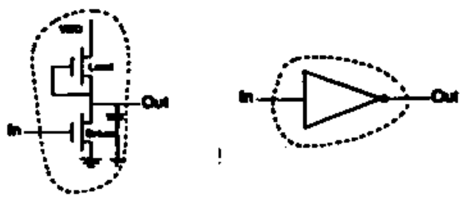


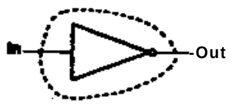
Figure 2: . Hie Structure of SAMSON



```
MODEL  nMOS(Gaie,Drain:VINPUT;Source:VOUTPUT):CIRCUIT;
 PARAMETER
   ChLength  «  6 { micron };
   ChWidth  =  12 { micron };
   VIO  =  +0.9{volis};
   Cox  =  6.15E-4 { oxide capacitance - pf/micront2 };
 PROCEDURE nMOSI(VAR R, JVGS, JVDS: REAL;
        VAR JIDS: REAL: = 1.0; VGS. VDS, IDS: REAL;
        PL, PW, PVTO, PCox: REAL);
BEGIN
 CGS(Gate, Source) -= 0.5 • ChLength • ChWidth;
 CGD(Gate. Drain) = 0.5 * ChLength • ChWidth;
  DS(Drain, Source) ^ nMOSI(CGS.V,IDS.V,IDS.I,
                    ChLength.ChWidth.VTO.Cox)
END;
```

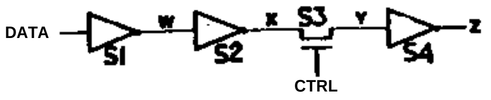


```
MODEL CINV(ln: VINPUT; Out: VOUTPUT): CIRCUIT;
 PARAMETER
   CLoad  »  0.08 { pF };
 BEGIN
  VDD(Pwr, GND)  =  5 { volts }
  Load(Out,Pwr,Out)  =  nMOS(ChLength = 6,ChWidth = 6.VT0» -5);
  Driver(In, Out, GND)  =  nMOS(ChLength = 6, ChWidth = 12);
  CL(Out, GND)  =  CLoad
 END;
```



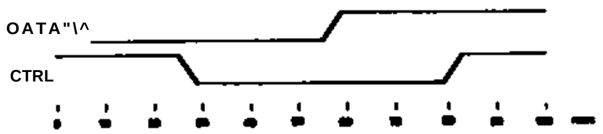
```
 MODEL LINV(In: VINPUT; Out: VOUTPUT): LOGIC;
  PARAMETER
    TL  *  3.75E-9 { sec };
    TR  =  3.4E-9  { sec };
fc TH  =  1.9E-9  {sec};
    TF  =  1.4E9  {sec};
  BEGIN
  Out:«  - In
  END;
```

H;,:i»o 3:   Model definitions



```
NETWORK Pa33Gate(Data, Ctrl: VINPUT);
 CINV(Load.ChLength « 8, Load.ChWkJth » 4, Load.VTO = -3):
      SP(W, X);
      S4(Y, Z);
 nMOS(Chl.ength = 4, ChWidth * 4):
      S3(Ctrl, X, Y);
 UNV(TR = 5.2E9).:
      Si(Data.W)
END.
```

Figure 4:   Network Description



```
NETWORK PassGate;
 EXCITATION
      Data  =  {time/state pairs }
            (0 H, 1E-9 F, 5E-9 Lt 55E-9 R, 59E-9 H);
      Ctrl  =  {time/voltage pairs }
            (0 5. 25E-9 5, 29E-9 0, 79E-9 0, 83E-9 5);
 TRACES
      PLOT Data, Ctrl, W, Xf Y, Z;
 CONTROLS
      FinishT:me  =  100E-9{sec};
      SampleStep  =  1E-9{sec};
      Low  =  0.18{vo)Is};
      High  =  5 { volts };
      LowThreshokJ  =  1 { volt};
      HighThreshokJ  =  4 { volts };
 END.
```

Figure 5:   Simulation Commands

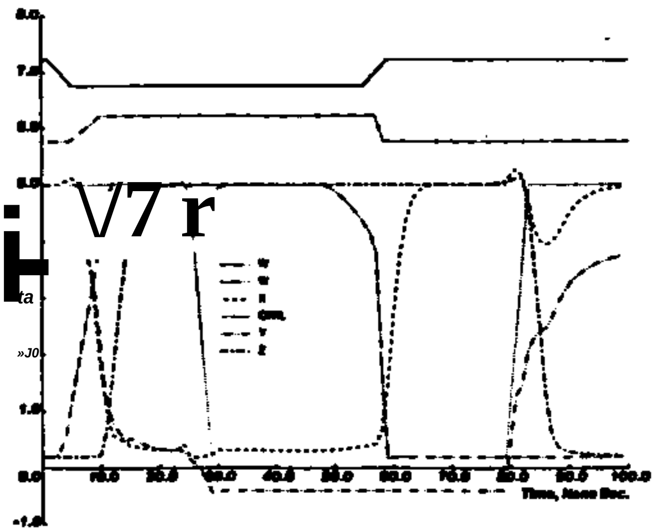


Figure 6:   Simulation results