CHAMP: Concurrent Hierarchical And Multilevel Program for Simulation of VLSI Circuits †

Daniel G. Saab, Robert B. Mueller-Thuns, David Blaauw, Jacob A. Abraham and *Joseph T. Rahmeh*

Computer Systems Group
Coordinated Science Laboratory
University of Illinois
Urbana, IL 61801

Department of Electrical and
Computer Engineering
University of Texas at Austin
Austin, TX 78712

ABSTRACT

This paper discusses the design and implementation of a hierarchical switch-level simulator for complex digital circuits. The hierarchy is exploited to reduce the memory requirements of the simulation, thus allowing the simulation of circuits that are too large to simulate at the flat level. The algorithm used in the simulator operates directly on the hierarchical circuit description. Speedup is obtained through the use of high level models. The simulator is implemented on a SUN workstation and has been used to simulate a switch level description of the Motorola 68000 microprocessor.

1. Introduction

Advances in Very Large Scale Integration (VLSI) technology have made possible the implementation of large and increasingly complex systems on a single integrated circuit chip. This has led to the need for efficient simulation tools capable of handling complex designs. These are necessary for both conventional logic simulation (to verify the correctness of a design) and fault simulation (to grade the quality of a set of test patterns).

Logic simulation has traditionally been performed at the gate level [1]. For Metal Oxide Semiconductor (MOS) technology this is an inappropriate level of abstraction. MOS circuits may contain ratioed logic and pass transistors exhibiting bidirectional signal flow and charge sharing effects. Furthermore, the associated gate level stuck-at fault model does not model realistic physical failures in MOS circuits adequately [2-5].

Switch level simulation, introduced by Bryant [6], is a practical alternative; it handles bidirectional signal flow in transistor networks and models signal strengths without incurring the overhead of a detailed electrical simulation. Also, transistor level fault modeling can be easily incorporated. However, when simulating a large design, the memory requirements for storing and operating on a 'flat' transistor level description become a limiting factor of simulator performance. Hence, in order to perform simulation using limited memory and computation time, circuit regularity and hierarchy must be exploited. Using hierarchy, the structure of commonly used subcircuits needs to be stored only once and can be referenced when needed. Furthermore, parts of the circuit can be substituted by higher level software descriptions for faster evaluation.

Existing switch level simulators [7-9] do not exploit a hierarchical circuit representation. A hierarchical logic and fault simulator was introduced in [10]. It is, however, limited to circuits described at the gate level. Similarly, multilevel simulation has typically been performed from the gate level upwards (see e.g. [11]). It is relatively easy to incorporate and exploit hierarchy in a simulator where primitives are unidirectional with well defined input and output ports. However, when switches or transistors are allowed as primitives, the difficult problems of bidirectional signal flow and charge sharing need to be considered. Hitherto, the problems of incorporating such effects in a hierarchical simulator have not been addressed.

This paper reports on the development, implementation and application of a hierarchical switch level simulator that operates directly on a hierarchical circuit description. We address the issue of bidirectional signal flow between circuit blocks, which arises when treating switch level simulation in a hierarchical framework. The simulator handles general MOS circuits and has the following features.

(1) It allows bidirectional signal flow inside circuit blocks that are represented as transistor networks, as well as across the boundaries of higher level blocks. Thus, no restrictive conditions are placed on the circuit description.

(2) It allows mixed mode simulation: parts of the circuit can be simulated faster at a behavioral level by supplying a high level software description.

(3) At any level of the hierarchy, the circuit can be described as a mixture of transistor networks, gates, and high level functions.

(4) It allows assignable delays to primitives or subfunctions.

(5) It provides automatic generation of high level descriptions (C-functions) of primitives.

A key objective in our work is the application to large circuits. The simulator we implemented has successfully simulated a switch level description of the Motorola MC68000 [12] microprocessor on a SUN Microsystems workstation.

2. Circuit Model

The circuit is considered to be a collection of functional blocks interconnected through boundary nodes. Each functional block can be either an interconnection of other functional blocks or an interconnection of primitives. A primitive is either a network of MOS transistors or a higher level behavioral descriptions.

A boundary node is one of three types: Input, Output and Input/Output (Ioput), with the following properties.

(1) Input nodes provide strong signals to a block and their logic behavior is not affected by the internal changes of other nodes in the block.

(2) Output nodes do not provide signals to any other nodes in a block. Their logic behavior is affected by internal changes in the block.

(3) Bidirectional nodes act either as Input or as Output nodes, depending on the logical behavior of other nodes in the block. Ioput nodes are created when functional blocks are interconnected by pass transistors.

As an illustration, consider the primitive block shown in Figure 1. Note that nodes A, B, C, and D are Input nodes, while Out is Input/Output node.

† This work was supported in part by Motorola, Inc. Austin, TX, and in part by the Semiconductor Research Corporation Contract 87-DP-109.

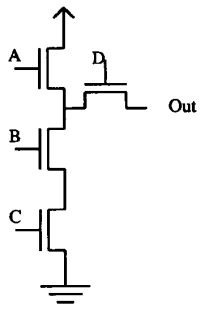

Fig. 1. Primitive Block

A primitive, as mentioned above, is either a functional block described in the C programming language [13] or an interconnection of MOS transistors. The first type of primitive is simply a program that describes the input/output behavior of the block. The C program corresponding to a block is either provided by the user or automatically generated by the program using compiled simulation techniques [1]. The second type of primitive is formed by the interconnection of MOS (both nMOS and pMOS) transistors. A transistor is modeled as a device with three nodes (source, drain, and gate). All transistors act as voltage-controlled switches which can be in one of three states: **on** (high conductance), **off** (open circuit), and **undefined** (**on** or **off** or intermediate). The nodes of the circuit may assume one of three values: **high** (1), **low** (0), or **undefined** (X). An nMOS transistor is **on** when its gate is **high**, **off** when its gate is **low**, and **undefined** when its gate is **undefined**. A pMOS transistor assumes the opposite state of an nMOS transistor under the same gate state. All transistors are bidirectional elements (i.e., no distinction is made between the source and the drain). Using switch-level transistor models the circuit is represented by an undirected vertex-weighted, edge-weighted 'switch' graph G(V,E) similar to the graphs described in [6, 14]. Switch-level simulation techniques are applied switch graph [6, 14-15] to find the steady-state of the corresponding primitive blocks.

Although only MOS transistors have been mentioned in this paper, this simulator is not limited in its scope to this particular technology. In fact, other technologies such as Bipolar Emitter Coupled Logic (ECL), Emitter Function Logic (EFL), and Current Mode Logic (CML) can be simulated using the switch graph techniques [16].

3. Data Structure Representation

To exploit the modularity and regularity of the circuit, the structure of every functional block is stored just once. The state of the nodes belonging to the block are duplicated every time the functional block is used. The data structures are defined as follows.

(1) A CELL is a structure which contains connectivity information such as node names, node types, and node fanouts.

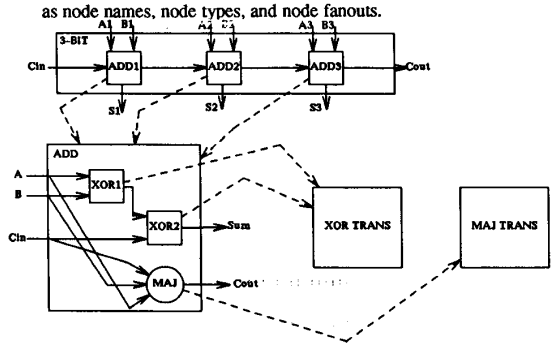

Fig. 1.2. Three-bit Adder Circuit

(2) An INSTANCE is a structure which contains the state of an instance of a CELL.

The state of a node N at any level is encoded in a triplet <t,δ,λ>, where 't' indicates whether the path to N originates at an input or storage node, i.e. whether the signal at N is driven or floating; δ gives the strength of the path driving N; λ represents the logic state of N and can take the values '0', '1', 'X'. This compact encoding allows the computation of states by simple comparisons of strengths and bit operations.

Figure 2 shows a circuit implementation of a three-bit full adder and demonstrates how the data structure is constructed. Note that the dashed lines point to CELL structures, while the solid lines represent electrical connections.

4. Simulation Algorithm

The simulation algorithm starts at the highest level. The states of Input nodes are propagated down to lower level primitives where the logical behavior of Output and Ioput nodes is computed and propagated up to higher levels. At every level, for every Output and Ioput node, the result of an evaluation is reduced to a single edge with appropriate strength (i.e. a conductance), in series with an independent input signal source. Hence, the evaluation produces the logical Thevenin's equivalent circuit of the block as seen from the output. The evaluation algorithm is illustrated by procedure EVAL_INSTANCE shown in Figure 3. This procedure operates on a single stack. Each entry of the stack contains an instance 'INST' and a flag 'UP'. The flag UP indicates whether the evaluation is top-down or bottom-up. Initially the stack contains the top instance with UP being FALSE indicating top-down evaluation. The procedure starts by testing the cell corresponding to the instance to determine if it is a primitive cell. If so, procedure EVAL_PRIMITIVE is called to deduce the logical behavior of the instance. EVAL_PRIMITIVE is invoked repeatedly until the top of the stack contains a non-primitive instance. When a non-primitive instance occurs, the flag UP is examined to find the direction of the evaluation. If the evaluation is top-down, then the instance environment is loaded by calling procedure LOAD_ENVIRONMENT; here all instances in the fanout list of nodes that have changed state are pushed on the stack. The changes propagate to the rest of the circuit through evaluation of these fanouts. In case the evaluation is bottom-up, the new environment is computed by a call to procedure IMPORT_STATE. This procedure imports the state from the lower level to the current level; it may, in turn, cause new events which are pushed on the stack and are evaluated as outlined above.

It should be noted that the evaluation should not proceed in arbitrary order for the following two reasons:

(1) A block should be evaluated once all blocks feeding signals into it have reached their steady state (feedback paths are cut arbitrarly). This leads to a reduction in the number of evaluations.

```
procedure EVAL_INSTANCE( stack )
    while stack is not empty
        while TOP( stack ) is primitive
            EVAL_PRIMITIVE( TOP(stack) );
            POP( stack );
        end while
        if evlaution is top-down
            LOAD_ENVIRONMENT( TOP(stack) ) ;
            if no activities resulted from loading
                the new environment
                    POP( stack ) ;
        else
            IMPORT_STATE( TOP(stack) );
            if no activities resulted from importing new state
                    POP( stack ) ;
    end while
end procedure
```

Fig. 3. Top Level Hierarchical Simulation Procedure

(2) In the presence of latching elements, evaluation with an intermediate state can lead to a wrong result as the following example illustrates (Figure 4). Assuming zero-delay simulation, if gate B is evaluated before the change in the output value of inverter A has occurred, the following transistor gate receives an intermediate enable signal and latches in a new value b.

Hence, in a preprocessing step, the entire circuit is levelized. In the hierarchical setting this can be done conveniently in a recursive fashion, thus also avoiding very large adjacency lists as in the case of a non-hierarchical flat circuit description.

4.1. Evaluation of Bidirectional Nodes

Allowing the use of bidirectional pins on the boundary of cells adds difficulty to the simulation problem but gives more flexibility in the way a circuit is partitioned into cells. A bidirectional node on the boundary of a cell can function as both Input or Output, depending on whether its current state is determined by a path leading to it from the outside or the inside of the given cell. Hence, for proper event scheduling, it is necessary to insert it into both the fanin and fanout lists of the node it is connected to at the next higher level. Furthermore, one must ensure that the proper environment is passed to a bidirectional pin. Consider the situation in Figure 5. Note that one cannot pass the current upper level state at N to the bidirectional pin P for evaluation, because the state may have been produced

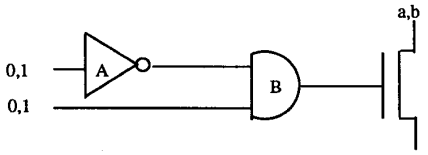

Fig. 4. A simple logic circuit

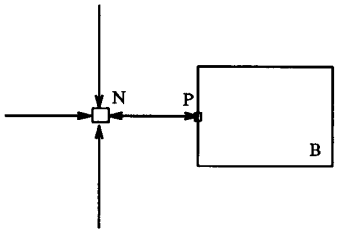

Fig. 5. Circuit configuration with bidirectional node

previously by a path from within block B passing through P. The path may no longer exist and hence one would 'reimport' a fictitious state. Instead, one has to compute the combined state at N of all paths excluding the one passing through P. Then, if there is a stronger path from within B it will dominate and produce the next state at N; otherwise, P functions as an input to B.

Consider the circuit shown in Figure 6 for an illustration of the steps in the evaluation. This circuit consists of two instances 'z' and 'w' of the same cell. The two instances are connected through node 'O1'. Also note that both 'z.o1' and 'w.o1' are Ioput nodes. An extra transistor is added to the cell with source at 'o1' (i.e. at 'z.o1' and 'w.o1' respectively) and with drain floating. This is done to import driving signals to the lower level.

Assume that z.A = z.B = 1, z.C = 0, while w.A = 1, w.B = 0, and w.C = 1. When the evaluation starts, the state of O1 is set to <F,δ_0,X>. (Here the ordering of conductances is: $\delta_i < \delta_j$ if $i < j$). When importing the state of O1, the gate of the extra transistor is set to zero in both instances because the strength of O1 is 'F'. Due to the changes in the environment, instances 'z', and 'w' need to be evaluated together. After the evaluation of instances 'z' and 'w', the resulting states are:

$$z.i1 = < D, \delta_2, 1>, z.o1 = < D, \delta_2, 1>$$
$$w.i1 = < D, \delta_1, 0>, w.o1 = < D, \delta_1, 0>$$

After the two instances are evaluated, the state of O1 is determined by comparing state(z.o1) to state(w.o1) which results in state(O1)=<D,δ_2,1>. This change is propagated back down to instance 'w' by switching on the extra transistor and by setting the strength of its drain to Input and its conductance to δ_2. Therefore, when evaluating instance 'w' with these conditions, the states become:

$$w.i1 = < D, \delta_2, 1>, w.o1 = < D, \delta_2, 1>,$$

which is a steady state solution.

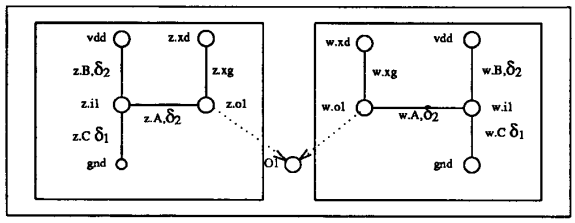

Fig. 6. Illustrating Input/Output Evaluation

5. Implementation and Application

The algorithms outlined above have been implemented in C on a SUN Microsystems workstation. The simulator accepts a hierarchical C-like description language.

As mentioned, the simulator supports multilevel evaluation. We conducted experiments that demonstrate the speedup one obtains when going from a low level to a high level description. The results are given in Table I for adder circuits ranging from 4-bit to 32-bit, described in terms of transistors, gates, and some in terms of 1-bit C-functions. Other results are given for a PLA with 19 inputs, 17 outputs and is roughly 290 equivalent gates. One observes speedups of around 5 and 6 in the case of the adder and around 25 and 36 for the PLA. It should be noted that the gate level description is automatically generated by CHAMP in a preprocessing step. This generation process can be skipped if so desired by the user.

Table I. Simulation results

Circuit	Level	#Elements	#test	CPU time(sec)
4 bit	Trans	184	512	11.4
	Gates	12	512	2.5
8 bit	Trans	368	2048	100.8
	Gates	24	2048	14.5
16 bit	Trans	736	4096	170.2
	Gates	48	4096	30.8
32 bit	Trans	1472	352	249.740
	Gates	96	352	48.540
	1-bit	32	352	37.200
PLA	Trans	4280	1000	804.220
	Gates	290	1000	568.380
	Func	1	1000	22.320

Figure 7 graphically illustrates the gains made when using high level representations as a function of circuit complexity.

A key objective of our work is to be able to handle large designs on engineering workstations. We simulated a switch level model of the Motorola MC68000 microprocessor including some surrounding 'glue logic'. The circuit is described as a mixture of gates (given as transistor

networks) and pass transistors. Its size amounts to the equivalent of roughly 80,000 MOS transistors. The hierarchical description of the circuit fits into 0.5 MBytes of memory; at run time, the program uses 7 MBytes. 16,000 clock phases were simulated and verified against the output supplied by Motorola Inc. An unoptimized version of the simulator took around 8 CPU seconds on a SUN microsystems 3/280 computer to simulate one clock phase without the use of any high level constructs. A major speedup may be obtained when replacing PLAs and other

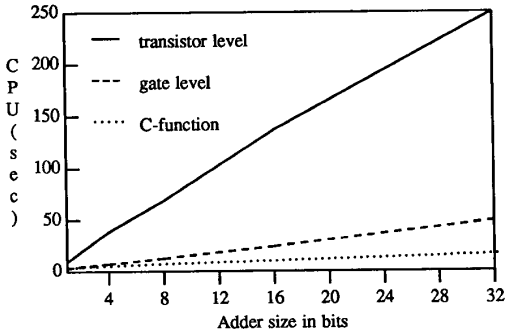

Fig. 7. Computation time as a function of circuit complexity

submodules by high level C-functions.

6. Conclusions

In this paper, techniques have been presented for hierarchical switch level simulation of VLSI circuits. Data structures and evaluation algorithms were outlined. Experiments with the use of high level descriptions showed promising speedup. The application to a large example - the complete switch level model of a microprocessor - demonstrated the usefulness and efficiency of our methods and their implementation.

Current work focuses on the extension of the simulator for fault simulation and the derivation of efficient high-level descriptions.

7. Acknowledgements

We wish to thank Motorola Inc., for providing the circuit model, and Wayne Vineyard, Jim Hamilton, and Ken Rosilier of the Semiconductor Products Group, Motorola Inc, Austin, TX, for technical discussions and support.

8. References

[1] M. A. Breuer and A. D. Friedman, *Diagnosis and Reliable Design of Digital System.* Potomac, MD: Computer Science Press, 1976.

[2] P. Banerjee and J. A. Abraham, "Fault characterization of VLSI MOS circuits," in *Proceedings of the IEEE International Conference on Circuits and Computers*, New York, New York, pp. 564-568, September 1983.

[3] I. N. Hajj and D. G. Saab, "Fault modeling and logic simulation of MOS VLSI circuits based on logic expression extraction," in *Proceedings of the IEEE International Conference on Computer-Aided Design*, Santa Clara, CA, pp. 99-100, September 1983.

[4] Y. M. El-Zig, "Failure analysis and test generation for VLSI physical effects," in *1983 Custom Integrated Circuit Conference*, Rochester, N. Y., pp. 300-303, May 1983.

[5] R. E. Bryant and M. D. Schuster, "Fault simulation of MOS digital circuits," *VLSI Design*, pp. 24-30, October 1983.

[6] R. E. Bryant, "A switch-Level model and simulator for MOS digital systems," *IEEE Transactions on Computers*, vol. C-33, pp. 160-177, 1984.

[7] R.E. Bryant , D. Beatty, K. Brace, K. Cho, and T. Scheffler, "COSMOS: A Compiled Simulator for MOS Circuits," *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pp. 9-16, 1987.

[8] R. E. Bryant, "MOSSIM: A switch-level simulator for MOS LSI," in *18th ACM/IEEE Design Automation Conf.*, Nashville, TN, pp. 786-790., June 29-July 1.

[9] C. J. Terman, "RSIM - A logic-level timing simulator," in *Proceedings of the IEEE International Conference on Computer Design*, New York, pp. 437-440, November 1983.

[10] William A. Rogers and Jacob A. Abraham, "CHIEFS: A concurrent, hierarchical and extensible fault simulator," in *Proceedings of the International Test Conference*, Philadelphia, PA, pp. 710-716., November 1985.

[11] D.D. Hill and W.M. Van Cleemput, "SABLE: Multilevel Simulation for hierarchical design," in *Proc. IEEE Int. Symp. Circuits and Systems*, Houston, TX, pp. 361-365, Apr. 1980.

[12] Motorola Corporation, *MC68000 Programmer's Reference Manual.* Englewood Cliffs, N.J.: Prentice-Hall, 1986.

[13] B.W. Kernighan and D.M. Ritchie, *The C Programming Language.* Englewood Cliffs, N.J.: Prentice-Hall, 1978.

[14] R. H. Byrd, G. D. Hachtel, M. R. Lightner, and M. H. Heydemann, "Switch level simulation: models, theory, and algorithms," in *Advances in Computer-Aided Engineering Design*, ed., A. L. Sangiovanni-Vincentelli. JAI Press Inc., pp. 93-148, 1985.

[15] I.N. Hajj and D. Saab, "Symbolic logic representation and logic and fault simulation of MOS circuits," *submitted for publication.*

[16] D. G. Saab, A. T. Yang,, and I.N.Hajj , "Delay modeling and timing of digital bipolar circuits," in *Proceedings of the 25th ACM/IEEE Design Automation Conference*, Anaheim, Ca., pp. 288-293, 1988.