Digital MOS Circuit Partitioning with Symbolic Modeling*

Lluís Ribas Xirgo Jordi Carrabina Bordoll
Computer Science Department
Universitat Autònoma de Barcelona
08193 Bellaterra (Cerdanyola), Spain
{ribas, bordoll}@microelec.uab.es

Abstract

This paper presents a method to automatically recognize and model single and multi-output logic gates out of a switch-level network, even for irregular transistor structures. Result subcircuit models are directly used in a symbolic simulator for circuit analysis purposes. Other applications of derived netlists cover switch-level simulation acceleration and test generation tool enhancement.

1. Introduction

Transistor networks must be used in those cases in which a high level of flexibility is demanded to adapt circuit designs to their ever-increasing stringent performance requirements. Unfortunately, switch-level CAD tools are necessarily more complex than the more conventional gate-level ones. However, the former tools gain efficiency if input networks are maximally simplified. More specifically, if groups of transistors/switches are packed into gates whose associated model could be used instead.

The process of deriving clusters of transistors out of a switch level network is also called *circuit partitioning* because it divides original networks into smaller parts that increase the coarseness of the final network.

Such network transformation can significantly reduce the time of switch-level simulations because network traversals are greatly simplified in terms of number of elements to be analyzed and, even, their model complexity. For instance, the model of an ideal switch in a four-valued Boolean algebra (described later in this paper) implies eight Boolean operations, while the model for a 2-input AND is resolved in just four.

Subcircuit models of derived partitions are usually stored as input/output tables, where every possible output is related to a given set of input stimuli. The size of such tables depends on the subcircuits' size, on the type of node values (i.e. continuous or discrete), and on the delay model (zero, unit or variable) of the devices. For zero/unit delay logic/symbolic simulators it is also possible to have several device templates to be used instead of the corresponding derived partition models. Those device templates are usually simple because derived models do not require detailed timing information, nor they have a wide range of discrete values. However, there is a limited number of such device primitives.

For instance, the switch-level circuit analyzer ANAMOS [3] associates any detected subcircuit to a combination of a reduced set of elementary primitives. However, it makes all the identification process much more complicated, and effectiveness is not ensured when several primitives have to be used for a single subcircuit.

The switch-level symbolic simulator used in this work has embedded the widely used inverters, buffers, n-input AND, NAND, OR, NOR, XOR, and XNOR gates because they also have a simple identification from the derived models [11]. As this approach is restricted to a set of gates, it is complemented by a sort of table-based solution that uses Boolean expressions instead of input-ouptut tables.

The ability to derive models of transistor groups can be used in most test generation (TG) tools to operate with a broad scope of logic gates. Thus diminishing the TG complexity, and eliminating the false faults caused by the use of sets of elementary gates for non-primitive gates [8].

The remainder of the paper is organized as follows. Next section reviews some previous works on transistor network partitioning before sketching our approach in the two following sections. The first one introduces circuit representation in symbolic processing environments because our circuit partitioning algorithm identifies and models transistor group functions through a symbolic analysis method inspired in event-driven simulators. The second one is devoted to the actual circuit partitioning and the final algorithm. Finally, we conclude by presenting a set of illustrative results, as well as by outlining switch-level based tools.

* This work was partially supported by the Spanish CICYT Programme, under project TIC98-0410-C02-02.

2. Circuit Partitioning Overview

Although sometimes network descriptions are hierarchical and do use subcircuit references for logic gates and, even, contain higher level groupings, there are applications in which these descriptions are totally flat. For instance, in a cell development process or for a reverse engineering use.

Transistors allow a bidirectional communication between source and drain terminals. The consequent unknown information flow causes extra operations to the simulation engine. Furthermore, the inherent complexity of such type of networks due to the number of components to be treated clearly imposes the need for automated methods to partition those netlists into more manageable subnetworks which are independently analyzed for further use as single components of a higher-level network.

Generally, the proposed methods in the related literature consist of partitioning the network into channel-connected transistors that usually match logic gate bounds.

Cerny and Gecsei [4] presented a simulator that used Boolean models of *connector-switch networks (CSN)* [6] to accelerate switch-level simulation. Models are stored in the form of decision diagrams (DDs) for CSNs that encode a ten-valued logic to deal with several levels of discrete signal strength. DDs are derived from a direct analysis of CG paths.

The switch-level circuit simulator COSMOS [3] includes program ANAMOS which partitions a network description into a set of channel connected subnetworks from which Boolean representations of their behaviors are built. The simulation of result netlists is more straightforward, with bidirectionality related problems mostly solved by the constructed models. Related Boolean representations are a couple of BDDs to encode a three-valued response. To avoid unnecessary subnetwork analysis, Beatty and Bryant [1] modified ANAMOS to work in an incremental fashion computing a hash signature for every analyzed subcircuit.

Yokomizo et al. [16] present a subcircuit identification method that is based on comparing subnetwork *logic trees* with a set of logic trees in a library of functions that can be used in pattern based (electrical) circuit simulators. Appropriate net terminals with capacitive loads are added after each subcircuit to prevent decreasing accuracy. However, parasitic elements might be eliminated if their effect does not exceed the allowed error tolerance.

Hübner and Vierhaus [8] also depicted a method for digital CMOS circuit partitioning. Their algorithm first determines longest paths from Vdd or Vss to an N or P transistor, respectively. Then, those connection points are regarded as starting points for stage determination, which are always channel-connected regions (CCRs) of the network, i.e., groups of transistors interconnected through drain or source ends. Functional analysis is done by on and off path traversal to find corresponding on and off path functions. If resulting on and off sets are not complementary, a specific check is performed to determine eventual high-impedance states.

The same authors and Camposano [9] have recently published a drastic improvement on the original method. In this new version, the transistor type is not taken into account when finding paths to Vdd or Vss and conflictive nodes are marked for a specific flow determination algorithm that requires simple inverters be given beforehand. Such modifications result in a more general and effective partitioning algorithm for static CMOS circuits.

Wehbeh and Saab [15] propose a method in which circuit is traversed while constructing on and off set functions for nodes. As they take into account signal strength, such traversals have to be done for every discrete strength level. In the end, result models handle subnetwork functionality as well as signal strength, and are used inside the CHAMP simulator [14].

A different approach is presented by Huang and Overhauser [7]. Circuit description is first converted to an adapted adjacence matrix which is partitioned. Then each matrix partition is encoded for its identification with previously defined elements.

Our approach is slightly different to the former ones: CCRs are not restricted by any way (on-paths may include N transistors, CCRs may have more than one output, dynamic logic structures are permitted, et cetera), and their analysis is performed by symbolic event-driven simulation. The result output functions are used to build the corresponding model.

3. Symbolic Representation of Switch-Level Circuits

The circuit structure is formally represented by a *circuit graph* (CG). A CG(V, E) is a directed bipartite graph [5] where V can be decomposed into two disjoint subsets (N and D), and $E \subseteq (N \times D) \cup (D \times N)$.

The set of adjacent input vertices of $v \in V$, denoted by $\text{adj}^-(v)$, contains every vertex for which vertex v is the last endpoint in an edge. Conversely, the set of adjacent output vertices of v_i is $\text{adj}^+(v) = \{w \in V \mid \exists (v, w) \in E\}$. Note that the adjacency sets of a vertex belonging to subset N are subsets of D, and vice versa.

N is the set of (electrical or logical) nodes that can hold a value. In symbolic analysis frameworks such as the one of this work, values of nodes $n_i \in N$ are represented by Boolean variables y_i. The *primary inputs* (PI) and the *primary outputs* (PO) are distinct subsets of N. We shall consider present state (PS) nodes to be part of PI and next state (NS) nodes included in PO.

The analysis of a circuit network is possible when each node holds a *nodal function* which determines its value

under any possible combination of input and present state nodes, i.e. any assignment to their corresponding variables.

Because binary values (0 and 1) are not enough to correctly model some of the common effects occurring at the switch level, we extend the number of discrete values to four: low (L), unknown (X), high (H) voltage level, and high impedance state (Z). Unknown value X refers to a situation in which there is a value conflict, while high impedance Z is related to those cases in which no device is driving a known value (L or H) into the affected nodes.

Consequently, nodal functions, f_i, are Boolean functions of a four-valued Boolean algebra ($B_4 = \{Z, L, H, X\}, +, \cdot, Z, X$) that are uniquely characterized by two binary Boolean functions [13], i.e. any f_i is represented by a pair of switching functions (f_i^1, f_i^0). The *on-set function (f_i^1)* indicates the cases for which node n_i is driven to high voltage level, and the *off-set function (f_i^0)* determines the conditions, or variable combinations, for which the node is electrically connected to ground.

In particular, input nodes n_i have functions of the form

$$f_i(\underline{x}) = x_k = (x_k^1, x_k^0) \tag{1}$$

where $\underline{x}$ is the vector of variables in B_4, and x_k is a variable characterized by two binary variables.

D is the set of devices $d_i \in D$ that are associated Boolean functions $\phi_i(\underline{z}) : B_4^{|\text{adj}^-(d_i)|} \mapsto B_4^{|\text{adj}^+(d_i)|}$, where $\underline{z}$ is the vector of device input variables from $\sup(\phi_i) = \{y_j \mid n_j \in \text{adj}^-(d_i)\}$, i.e. from variables related to input nodes of d_i. (These variables are substituted by nodal functions in terms of input node variables when symbolically simulating the network.)

Nodal functions are derived from an event-driven simulation algorithm [12] which starts with all nodes set to Z and continues by creating events to set PI nodes to their corresponding variables. The rest of events (i.e. nodes that are added some Boolean function to the one they hold) is caused by device activity.

Transistors process symbolic information as it flows through the network. Their behavior is reduced to a simple switch, i.e. it can be *off* completely isolating both end nodes or *on*, letting data flow in both directions. Hence, transistors are considered bi-directional devices that modify both drain (f_d) and source (f_s) nodal functions. The complete set of equations $\Phi = (\phi_d, \phi_s)$ for a PMOS switch is expressed as follows.

$$\begin{aligned} \phi_d(f_d, f_g, f_s) &= f_d(\underline{z}) + f_g^0(\underline{z}) \times f_s(\underline{z}) \\ \phi_s(f_d, f_g, f_s) &= f_s(\underline{z}) + f_g^0(\underline{z}) \times f_d(\underline{z}) \end{aligned} \tag{2}$$

where products $f_g^0 \times f_s$ and $f_g^0 \times f_d$ are products of functions by a vector of functions (those function pairs that characterize f_s and f_d), and sums between function vectors are made component by component, i.e. $f + g = (f^1 + g^1, f^0 + g^0)$. Note that only two device functions (ϕ_d and ϕ_s) are required because the transistor's gate nodal function (f_g) is not modified by the transistor activity.

A similar system describes the behavior of a NMOS switch, with the off-set function of the gate substituted in (2) by the on-set function.

The model of a network consists of all nodal functions of primary output and next state nodes once symbolic simulation has finished (i.e., there are no new events).

However, this step must be done after obtaining a transistor subnetwork. This partitioning process will be described in the next section.

4. Partitioning Algorithm

Circuits are partitioned into CCRs. Elements other than transistors are not included in such partitions, but transistors are included regardless of their position (in on or off-paths). Each CCR is analyzed by taking profit of the symbolic simulation engine rather than by specific traversal algorithms as applied in the former methods. This solution is not so effective but completely generic.

As in [8], determination of CCRs are done by detecting a set of possible stage outputs (SO) that are used as starting points to obtain complete CCRs by "channel traversing" the circuit. Once a CCR is detected, their input/output nodes are determined to be simulable. Then, their model is built after the symbolic simulation results, and their inner nodes and all their devices removed. Also, CCR outputs are withdrawn from SO. Figure 1 shows such process in an algorithmic form.

The process starts by determining SO before trying to detect any CCR. SO is initially assigned to PO, and then increased with nodes that do not have any voltage source attached to them and are gates of transistors or connected as inputs to other devices. For the example in Fig. 2(a), SO=$\{y, z, v, u\}$. This set is diminished at each iteration of the main loop until it is empty.

Detection of a particular CCR is done by function DFS(CG, n_s). DFS(CG, n_s) performs a depth-first search over transistor channels, returning a CG that contains all nodes and transistors effectively visited, i.e. not including those only tested for search continuation. For instance, DFS(CG,y) on the circuit in Fig. 2(a) returns the CG with the transistors in the shadowed part marked, as well as nodes $\{v, w\}$.

Initial, global CG should not contain loops. Instead, they should be broken into two nodes for the present state (PS) and next state (NS) sets, which are implicitly included in the PI and PO sets, respectively. However, if any loop is not broken before applying the partitioning algorithm, result netlist preserves this loop outside CCRs (see the result of partitioning the output latches of the DCVSL gate in Fig. 2), unless

```
CG = partition( CG, PI, PO )
  /* Detect all possible stage outputs, SO. */
  SO ← PO
  for each node n_i ∈ CG do
    if ( ∄d_j ∈ adj⁻(n_i))[type(d_j) = VSOURCE] then
      for each device d_j ∈ adj⁺(n_i) do
        if (type(d_j) = OTHER) ∨
        ∨ (type(d_j) = {P,N}MOS) ∧ (n_i = gate(d_j)) then
          SO ← SO ∪ {n_i}
  /* Replace CCRs by higher-level devices. */
  while SO ≠ ∅ do
    get first node n_s in SO, i.e. SO ← SO \ {n_s}
    CCR ← DFS( CG, n_s )
    PO_CCR, PI_CCR ← ∅
    /* Derive CCR inputs. */
    for each n_i ∈ CCR do
      if (n_i ∈ PI) ∨ (CCR ∩ adj⁻(n_i) ≠ ∅) then
        PI_CCR ← PI_CCR ∪ {n_i}
        CCR ← CCR ∪ { deviceCreate(VSOURCE, n_i) }
      endif
    /* Determine CCR outputs. */
    for each n_i ∈ CCR do
      if n_i ∈ SO then
        PO_CCR ← PO_CCR ∪ {n_i}
        SO ← SO \ {n_i}
      endif
    /* Create CCR model. */
    simulate( CCR )
    Φ_CCR ← modelCreate( CCR )
    d_CCR ← deviceCreate( SUBCKT, CCR )
    CG ← (CG \ CCR) ∪ PI_CCR ∪ PO_CCR ∪ {d_CCR}
  endwhile
end
```

Figure 1. Partitioning algorithm.

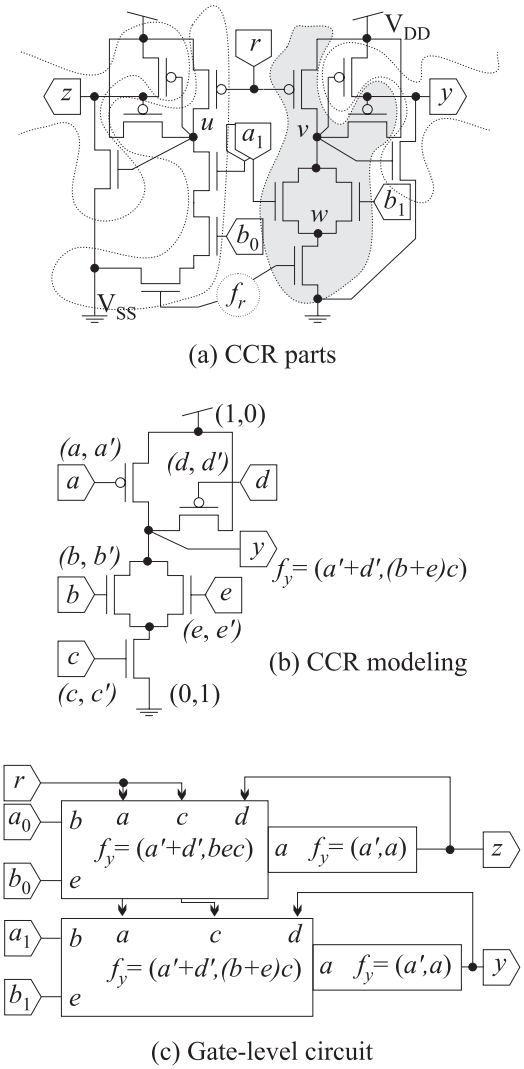

Figure 2. A DCVSL gate partitioning example.

it is a self-loop for some CCR, i.e. the nodes in the loop are internal to that CCR.

The construction of a CCR model is done by first creating virtual inputs to it, and then symbolically simulating it so its output nodes contain its functional model, as shown in Fig. 2(b). CCR inputs are nodes attached to devices that do not belong to the CCR, but that are able to supply some functions to them. In particular, PI nodes are attached to voltage sources that cannot belong to any CCR (they are not transistors) and supply (are in the input adjacency) a constant function to the nodes.

CCR outputs can be derived in a similar form, so to know where will be placed the corresponding model. An output is a CCR node, n_i that constitutes an input to a device, d_j, not present in the same CCR, i.e. $d_j \in \text{adj}^+(n_i)$. Also, all observable nodes (PO nodes) are considered to be outputs of a CCR. Therefore, all these nodes are nodes present in SO. Consequently, CCR outputs are nodes in the intersection of CCR and SO.

The simulation results are used to build the associated model via modelCreate(CCR) function, and a new device is generated to replace all transistors in the substituted CCR. Models contain either a sum-of-product (SOP) expression or a binary decision diagram (BDD) for each CCR output and some additional data. Replacing devices have a reference to the corresponding model and their actual "pinout" within the network.

The cost of each simulation depends on the number of inputs of the simulated CCR and on the Boolean function complexity, and is usually affordable for most common circuits.

The result CG is made of as many devices as detected CCRs plus extra devices, i.e. other than transistors. Differently from other approaches, a CCR may contain more than just one output. However, primary inputs and outputs of any CCR are disjoint sets, i.e. $PI_{CCR} \cap PO_{CCR} = \emptyset$.

Note that the number of logic gates in a circuit might not be equal to the number of CCRs because there are logic gates made out of several stages and pass-transistors that may connect several logic gates into one CCR. For instance, an OR gate will be divided into a NOR gate and an inverter at its output. Conversely, pass transistor logic will cause CMOS buffers and inverters be considered in the same CCR than the pass transistors they are attached to.

5. Results and Conclusion

In this paper we have presented a switch-level network partitioning algorithm based on grouping channel-connected transistors without any other limitation. Correct identification of result partitions and functional model generation is a unique procedure that is carried out through symbolic simulation to enable generic partitions be correctly modeled. Result models are correct even for dynamic logic gates, multiple output subcircuits, circuits containing devices other than transistors and any type of irregular transistor structures, which surpasses the capabilities of the previous works in literature.

Incorrect partitions are also modeled accordingly. Therefore, should the global network contain a wrong transistor subnetwork, the derived network preserves the same faulty functional behavior [12].

Derived models can be used in a quite straightforward form within logic simulators and automatic test patters generators. In particular, we have embedded this partitioning algorithm into a symbolic analysis tool for switch-level circuits [12] that also relies on symbolic simulation.

Table 1 shows the execution results for the ISCAS85 [2] circuit set. Original circuits are described at the gate level and they have been flattened to the transistor level and reorganized into CCRs later with the help of the partitioning algorithm.

Differences between the number of partitions in the last column and the number of original gates shown in the second one are due to the two-stage cells (AND and OR gates), and to the XOR gates used in circuits c432 and c499. Such XOR gates (see Fig. 3) have been designed with pass-transistor logic to illustrate how this type of logic may collapse two different logic stages: in this case, the inverter in the middle of the XOR is included in the gate's CCR. Also, the logic gate that is attached to input a is included into the XOR CCR. As a side-effect, the result CCR may have two outputs if this last gate output is used as input for other gates. Consequently, the validity of our method for multiple output CCRs is proved.

Table 1. Switch-level Partitioning

	size		CPU secs.			
circuit	gates	MOS	(4)	(5)	(6)	parts
c432	160	752	–	0.2	0.1	164
c499	202	1384	–	0.5	0.3	338
c880	383	1802	0.7	0.5	0.2	529
c1355	546	2308	–	–	0.2	604
c1908	880	3446	1.4	1.0	0.7	1105
c2670	1193	5364	–	–	2.3	1875
c3540	1669	7504	3.2	2.1	2.1	2460
c5315	2307	11262	–	3.2	6.2	3552
c6288	2416	10112	–	3.0	3.1	2672
c7552	3512	15396	6.4	4.6	10.2	5067

The fourth column of table 1, labeled with number four between parentheses, gives the CPU time on a SUN 3/60 for the signal flow analysis by Lee et al. [10]. The fifth column shows the results of the partitioning algorithm by Hübner et al. [9] on a SUN IPX (about 7.4 times faster than a SUN 3/60, according to Dhrystone 1 benchmarks). Next column gives the CPU time of our method on a SUN Sparc 2. Though not directly comparable, results of the two first methods are proportional to the number of gates/MOS, while there is no such relation in our case. This is due to the fact that symbolic modeling depends on the CCR function complexity rather than on its structure. Anyway, CPU time is small enough to enable its application even for large circuits.

Furthermore, our algorithm is shown to be completely generic, thus able to deal with non-complementary logic design styles, multiple output subnetworks, and other irregular transistor structures, even if mixed with other devices. In other words, the symbolic/functional approach can deal with any type of switch level circuits, especially with those that may possibly contain transistor configurations that cannot be correctly treated with structural methods.

In the short term, we will include a hash table to store models in order to reduce the number of different models to be called. In the meantime, each subcircuit has its own model.

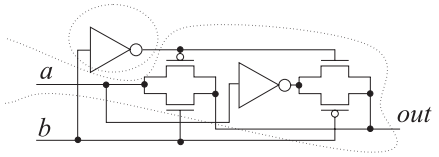

Figure 3. A 8-transistor XOR gate.

References

[1] D. Beatty and R. Bryant. Fast incremental circuit analysis using extracted hierarchy. In *25th ACM/IEEE Design Automation Conference*, pages 495–500, 1988.

[2] F. Brglez and L. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in FORTRAN. In *Proc. of ISCAS'85*, pages 662–698, 1985.

[3] R. Bryant, D. Beatty, K. Brace, K. Cho, and T. Sheffler. COSMOS: A compiled simulator for MOS circuits. In *24th ACM/IEEE Design Automation Conference*, pages 9–16, 1987.

[4] E. Cerny and J. Gecsei. Simulation of MOS circuits by decision diagrams. *IEEE Transactions on Computer-Aided Design*, 4(4), October 1985.

[5] A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, 1985.

[6] J. P. Hayes. A unified switching theory with applications to VLSI design. *Proceedings of the IEEE*, 70(10):1140–1151, October 1982.

[7] K.-T. Huang and D. Overhauser. A novel graph algorithm for circuit recognition. In *ISCAS*, pages 1695–1698, 1995.

[8] U. Hübner and H. Vierhaus. Efficient partitioning and analysis of digital CMOS circuits. In *International Conference on Computer-Aided Design*, pages 280–283, 1992.

[9] U. Hübner, H. Vierhaus, and R. Camposano. Partitioning and analysis of static digital CMOS circuits. *IEEE Transactions on Computer-Aided Design of ICAS*, 16(11):1292–1310, November 1997.

[10] K.-J. Lee, R. Gupta, and M. A. Breuer. A new method for assigning signal flow directions to MOS transistors. In *IEEE/ACM International Conference on Computer-Aided Design*, pages 492–495, 1990.

[11] F. Mailhot and G. DeMicheli. Algorithms for technology mapping based on binary decision diagrams and on Boolean operations. *IEEE Transactions on Computer-Aided Design*, 12(5), May 1993.

[12] L. Ribas and J. Carrabina. Analysis of switch-level faults by symbolic simulation. In *32nd ACM/IEEE Design Automation Conference*, pages 352–357, 1995.

[13] L. Ribas, R. Peset, and J. Carrabina. Two-rail switch-level symbolic analysis. In *3rd International Workshop on Symbolic Methods and Applications to Circuit Design (SMACD)*, pages 307–314, 1994.

[14] D. Saab, R. Mueller-Thuns, D. Blaauw, J. Abraham, and J. Rahmeh. CHAMP: Concurrent hierarchical and multilevel program for simulation of VLSI circuits. In *International Conference on Computer-Aided Design*, pages 246–249, 1988.

[15] J. A. Wehbeh and D. G. Saab. Hierarchical simulation of MOS circuits using extracted functional models. In *International Conference on Computer Design*, pages 512–515, 1992.

[16] G. Yokomizo, C. Yoshida, M. Miyama, Y. Motono, and K. Nakajo. A new circuit recognition and reduction method for pattern based circuit simulation. In *Custom Integrated Circuit Conference*, page 9.4, 1990.