Extracting RTL Models from Transistor Netlists

K. J. Singh and P. A. Subrahmanyam

AT&T Bell Laboratories, Holmdel, NJ 07733

Abstract

This paper addresses the problem of deriving a register-transfer level (RTL) model from a transistor-level circuit. Using existing techniques, the transistor-level circuit is converted into a relation that describes the evolution of the signals in the circuit with respect to the simulator clock. This simulation relation is then manipulated to derive the stable behavior of the circuit. Given this stable behavior and information about the clocking scheme, we determine if the circuit is combinational, asynchronous or synchronous. For combinational and synchronous circuits we derive an equivalent register-transfer level model. This development enables full-custom circuit designers to use tools that were till now available only to designers working at the gate-level. The algorithm has been successfully used to characterize several custom designs, as well as the entire AT&T standard-cell library.

1 Introduction

Circuits that can be described at the register-transfer level constitute a large percentage of digital designs, and analysis and optimization tools are widely available for such circuits. In contrast, fewer tools support higher level reasoning about *transistor level* integrated circuit designs. This paper describes a technique that, given a netlist of transistors, derives an equivalent register-transfer level description; the technique first identifies if a given circuit is combinational or synchronous before attempting to derive an equivalent model.

This research was initially motivated in the context of developing formal verification techniques for custom VLSI designs, where there was a need to reason about transistor-level circuit descriptions. Fortuitously, the ability to perform such circuit extraction additionally enabled full-custom designers to use commercial EDA tools developed for gate-level ASIC design (like fast logic-simulation, fault-simulation, test pattern generation, timing analysis). Furthermore, logic simulation is at least an order of magnitude faster than switch level simulation. Having an automatically extracted gate-level equivalent circuit from a transistor representation allows simulation based validation to be sped up considerably, reducing design time.

In Section 2 we review some previous work on developing logic models for transistor circuits and contrast them with the current approach. In Section 3, we then describe our algorithm for extracting logic models from transistor circuits. The algorithm has been used to derive the functionality of gates in a standard-cell library and fragments of full-custom designs used in production. These experimental results are presented in Section 4. Finally, in Section 5 we present conclusions and comments.

2 Background

Pattern matching has been proposed [4] as an approach for extracting equivalent logic circuits from transistor netlists. This technique is effective when there is a restricted style of designing circuits. However, since it is difficult to capture all the different circuit variations and optimization tricks employed by designers, pattern-matching techniques are not very effective for full-custom design styles. Our extraction approach is based on the circuit functionality and not on the structure of the netlist and consequently does not suffer from this limitation. Our approach can complement pattern-matching: modules that are not recognized by template matching may be input to the abstraction tool we have developed.

There has been prior work [1, 3, 2] in transforming transistor netlists into equivalent logic netlists for improving the speed of switch-level simulation. Switch-level simulators differ in the effects that are modeled (transistor strengths, directionality, charge storage, etc.), the delay assumptions for the transistors, as well as the accuracy to which the node voltages are represented (binary, ternary, four-valued, etc.). However they all use models that describe the evolution of the circuit from one time instant to the next. We call such models **simulation models**. Each simulation cycle is represented by one occurrence of the **simulation clock** which is used by the simulator to schedule different parts of the circuit for evaluation. When a computation is indicated to be of unit-delay type, it produces the logic value for a node at the next simulation cycle while if a computation is of zero-delay type, it determines the node value to be used in the current simulation cycle. During each simulation cycle the relevant equations are evaluated to determine the node voltages. Unit-delay elements are associated with feedback

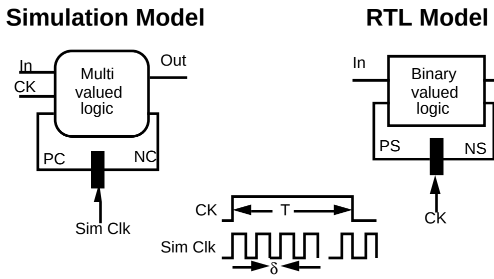

Figure 1: Simulation model vs. RTL model

arcs and charge-storage nodes in the circuit. The set of signal values associated with unit-delay elements is called the circuit **configuration**.

Synchronous digital circuits are described using a **register transfer level model** (RTL model) and our goal is to derive such a model, if possible, from a transistor netlist. An RTL model describes the circuit behavior as an interconnection of logic gates and memory devices, latches and flip-flops, whose behavior is pre-defined. The **state** of the circuit is set of values stored in the memory devices. The state changes in response to input values and the previous state when the **clock signals** change. When using an RTL model the user is concerned mainly with the evolution of the state of the system and a clock_cycle-by-clock_cycle description of the outputs.

Even though both the simulation model and the RTL model represent the circuit behavior as finite-state machines, the FSMs differ in their interpretation. The *configuration* in the simulation model is a microscopic representation of circuit state, while the *state* in an RTL model is a macroscopic representation. It is important to understand the distinction between the simulation model and the RTL model in order to appreciate how our logic extraction differs from that of [3, 2]. This difference is illustrated in Figure 1. In the simulation model, a new configuration is computed by evaluating the logic at every simulation tick, denoted by a time interval δ, till the circuit response is unchanged. Due to the fine granularity of time, the simulation model can model the transient values at the nodes in the circuit in addition to their final values. Also, since events on all signals (clock and data inputs) are processed in the same manner the simulation model can model both synchronous and asynchronous circuits. On the other hand, in an RTL model the clocks have a special status since the equations describing the circuit state are evaluated only when the clock inputs change. This evaluation computes the new state of the circuit which is stored in the registers for use during the next clock cycle. Since the circuit behavior is described in terms of state changes, there is no modeling of the transient voltages in the circuit. RTL models represent synchronous circuits and the interpretation of the RTL model depends on the clocking scheme. Thus to derive an RTL model from a simulation model, the clocking scheme must be specified so that the appropriate analysis can be performed.

A previous attempt to address the extraction is described in [5]. The simulation relation was manipulated to abstract the transient behavior and the resulting description was mapped to edge-triggered flip-flops. This work could not recognize level-sensitive latches; further, it was the user's responsibility to determine if the extracted functionality was a complete description of the circuit. In our current method we first propose conditions to check for synchronous and combinational behavior, and only when the simulation model fits these conditions do we derive equivalent RTL models based on level-sensitive latches.

2.1 Terminology and notation

The *characteristic function* of a set A is a Boolean function $A(x)$ such that $A(x) = 1$ if and only in $x \in A$. Sets will be represented by their characteristic functions. Since a relation is a set of n-tuples, it is represented by its characteristic function as well.

The *substitution*, within a Boolean function F, of a set of variables X by a corresponding set of variables Y, is denoted by $[X \rightarrow Y]F$.

The *cofactor* of a Boolean function F with respect to a *literal* x_i (respectively $\overline{x_i}$), denoted by F_{x_i} (respectively $F_{\overline{x_i}}$), is a new function obtained by substituting 1 (respectively, 0) for x_i in F.

The *existential quantification* or *smoothing* of F over a variable x_i is denoted by $\mathcal{S}_{x_i} = F_{x_i} + F_{\overline{x_i}}$. The new function $\mathcal{S}_{x_i}F$ is independent of the variable x_i. When existential quantification is required over a set of variables, the smoothing operator is applied once for each variable.

A 4-valued signal x_i can assume a value in the set $\{0, 1, X, Z\}$. It is encoded by two binary variables (x_i^H, x_i^L). Symbols 1 and 0 are the *binary values* and are encoded as (10) and (01) respectively, while symbols X and Z represent the *non-binary values* and are encoded as (11) and (00) respectively.

The *binary restriction* of a function T over a set V, denoted $\mathcal{B}_V(T)$, is a new function which is defined only over all the binary values of the 4-valued signals in the set V. This function is computed as

$$\mathcal{B}_V(T) = T. \prod_{x_i \in V} (x_i^H \oplus x_i^L)$$

where $\oplus$ denotes the exclusive-or of Boolean variables. $\mathcal{B}_V(1)$ denotes all the possible binary values that the 4-valued signals in the set V can take on.

3 Extracting RTL models

The overall approach we adopt is shown in Figure 2 and details of the operations and the various tests suggested in Figure 2 are discussed in this section. The transistor netlist

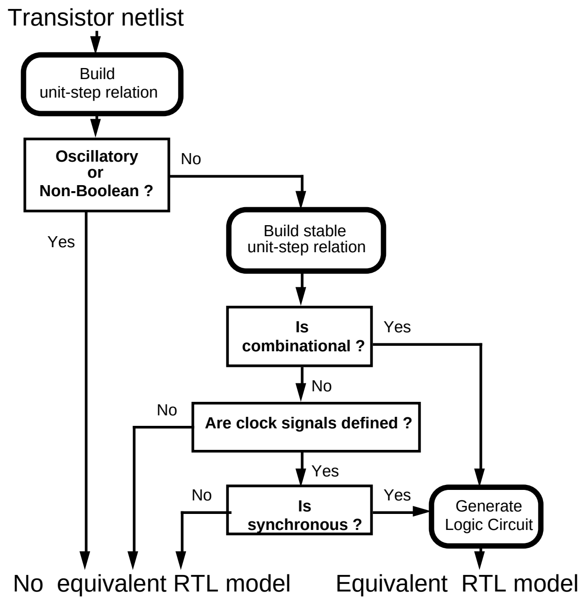

Figure 2: Outline of the abstraction procedure

is processed to obtain a unit-step relation that describes the evolution of the circuit configuration (Section 3.1). We stop the analysis if this relation represents an oscillatory circuit or a circuit whose output is not binary valued for some binary input condition (Section 3.2). For stable circuits, we then derive a *stable unit-step relation* that abstracts away the transient behavior (Section 3.3). We examine the stable unit-step relation to determine if the circuit is combinational or sequential (Sections 3.4 and 3.5). Combinational circuits are converted into an acyclic interconnection of logic gates. Sequential circuits may be either synchronous or asynchronous. If a designer wants to distinguish between these, he/she can define a clocking scheme for the circuit to constrain the circuit behavior. If the circuit is synchronous under the clocking scheme, an equivalent RTL model is derived.

3.1 Deriving a unit-step relation

We use ***Tranalyze*** [2] as a pre-processor to derive the simulation model. ***Tranalyze*** uses a 4-valued representation for the signals and produces an interconnection of zero-delay logic functions and unit-delay elements. Unit-delay elements represent feedback arcs or charge-storage nodes in the circuit. Associated with each unit-delay element n_i, are 4-valued variables x_i and y_i — x_i represents the value at the output of n_i and y_i represents the value at the input of n_i. Variables x_i and y_i are represented using pairs of Boolean variables, (x_i^H, x_i^L) and (y_i^H, y_i^L) respectively. The **present-configuration**, denoted PC, is the set $\{x_1, \ldots, x_m\}$ and the **next-configuration**, denoted NC, is the set $\{y_1, \ldots, y_m\}$. ***Tranalyze*** also generates two Boolean functions f_i^H and f_i^L for each output and unit-delay element. These determine the value of the variables y_i^H and y_i^L based on the present-configuration and the circuit inputs, I.

The **unit-step relation**, $R(I, X, Y)$, is derived from the simulation model to describe the global behavior of the circuit. It describes a circuit whose configuration X changes to Y when an input I is applied. The characteristic function of the unit-step relation is obtained by setting the variables y_i to be equal to the function that determines them. For the case when f_i^H and f_i^L both evaluate to 0, the signal is not driven and retains its old value,i.e. $y_i \equiv x_i$.

$$R(I,X,Y) = \prod_{y_i \in NC} \left(\begin{array}{l} y_i^L . y_i^H . f_i^L(I,X) . f_i^H(I,X) \ + \\ \overline{y_i^L} . y_i^H . \overline{f_i^L(I,X)} . f_i^H(I,X) \ + \\ y_i^L . \overline{y_i^H} . f_i^L(I,X) . \overline{f_i^H(I,X)} \ + \\ (y_i \equiv x_i) . \overline{f_i^L(I,X)} . \overline{f_i^H(I,X)} \end{array} \right)$$

The unit-step relation describes the circuit response to inputs that are 4-valued. In practice we are interested in the response of the circuit to a digital input stream. The **binary unit-step relation**, denoted R_b, is simply the unit-step relation R, with the input variables restricted to binary values.

$$R_b = \mathcal{B}_I(R)$$

A **transition** from configuration x to y under input i is denoted $[i, x, y]$. The transition represents valid circuit behavior if $R_b(i, x, y) = 1$.

A **steady transition** is of the form $[i, x, x]$. It denotes that the circuit configuration remains unchanged, equal to x, as long the input is held constant at i. Such a circuit configuration is called a **stable configuration**. Formally, the set of stable configurations, SC can be described as

$$SC = \{x | \exists i, R_b(i, x, x) = 1\}$$

The set of **stable binary configurations**, SBC, represents the stable configurations that have binary values for all components. This represents the set of stable configurations of the circuit where all nodes have attained either a high or low voltage value.

$$SBC = \mathcal{B}_{PC}(SC)$$

The **steady binary transitions**, SBT, associated with a binary unit-step relation, R_b, are the self-loops at the stable binary configurations.

$$SBT = \{[i, x, x] | (R_b(i, x, x) = 1) \wedge (x \in SBC)\}$$

We next identify the **relevant circuit behavior** that we will extract. *Starting from a stable configuration where all the components have binary values, the circuit behavior is the stable response of the circuit to binary valued inputs that are consistent with the clocking scheme.* This definition of circuit behavior means that we first have to determine if the circuit is stable or not.

3.2 Stability Test

After building the unit-step relation we determine if the circuit is stable or oscillatory. *We assume that if there exists a stable binary configuration corresponding to an input condition, then the circuit will settle in that configuration when the corresponding input is applied.* This gives us an easy way to check if the circuit is stable — all we need to check is that there is a stable binary configuration corresponding to every input.

A binary unit-step relation, R_b, represents a non-oscillatory circuit if $\mathcal{S}_{PC,NC}(SBT) \equiv \mathcal{B}_I(1)$, where SBT represent the steady boolean transitions of R_b and $\mathcal{B}_I(1)$ represents the set of all possible binary input conditions.

Circuits that have oscillations will not have a stable configuration corresponding to some input and will fail this test. The algorithm we describe here will not derive equivalent RTL descriptions for oscillatory circuits.

3.3 Computing a stable unit-step relation

The next step in the extraction is to abstract the transient behavior. Consider the fragment of a simulation relation shown in Figure 3(a). Configurations A and D are stable binary configurations, i.e. there is some input for which the circuit will remain in these configuration till the input changes. Thus as long as the input is $i1$ and the circuit configuration is A, the circuit configuration remains A. Now assume that the circuit input changes to $i2$. The circuit goes through intermediate configurations B and C before stabilizing at configuration D. Since we are not concerned with modeling the transient behavior in the RTL model (we assumed that the circuit has time to settle after every input change), we replace the configuration space with the reduced configuration space shown in Figure 3(c) where there is a direct transition from one stable configuration to the other on the given input. This mimics the analysis that a human performs when trying to understand the circuit operation — they simulate it till they get a stable circuit configuration in response to an input change.

The **stable unit-step relation**, denoted R_s, represents the transition structure obtained by eliminating the transient configurations from the unit-step relation R_b. To compute R_s, which represents the steady-state response of the circuit to an input, we use the algorithm described in Figure 4. The basic idea is to replace consecutive transitions by a single transition and repeat the process till all transitions occur between stable binary configurations. A similar technique was proposed in [5].

A **stable transition** originates at a stable binary configuration and terminates at a stable binary configuration under the input condition for which the terminating configuration is stable. $[i, x, y]$ is a stable transition if $x \in SBC$ and $[i, y, y] \in SBT$. We denote the set of stable transitions derived from a relation N as $\mathcal{T}_s N$.

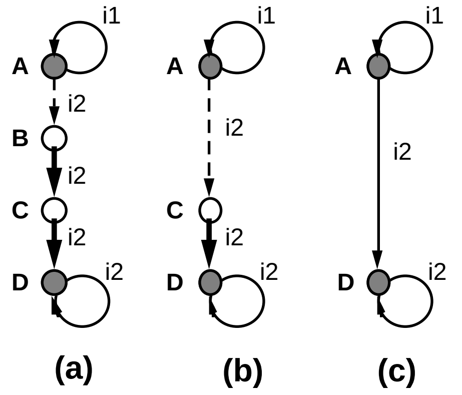

Figure 3: Notion of stable unit-step relation

An **unstable transition** originates at a configuration that is not a binary stable configuration. A transition $[i, x, y]$ is unstable if $x \notin SBC$. The set of unstable transitions obtained from a relation N, is denoted as $\mathcal{T}_u N$.

The algorithm starts by computing the set of transitions in R_b that will form part of the final stable unit-step relation R_s (line 1). It then computes the set of transitions that originate at stable configurations but are not part of the steady transitions (line 2). This set is denoted as R_{su} (transitions from *s*table to *u*nstable configurations). We then iteratively compute the consecutive transitions (line 4) and the initial transition of the two-step transition (line 5). We partition the set of consecutive transitions A, into the transitions that will be part of the final relation, $\mathcal{T}_s A$ and the set of transitions that end at unstable configurations A' (line 6). We terminate the iteration when no new transitions are added to R_s (line 7). If there are new transitions we add them to the result R_s (line 8) and update the set R_{su} (line 9). Figure 3 shows the different types of transitions — dashed arcs denote the starting transitions, those in R_{su}; the bold arcs denote the unstable transitions, those in R_u; while normal arcs are part of the stable unit-step relation R_s.

The classification of the circuits is based on characteristics of the stable unit-step relation as well as the clocking scheme. Whether we perform the synchronous test or the combinational test depends on whether a clocking scheme has been specified for the circuit. These tests are described next.

3.4 Identifying Combinational Circuits

A relation R_s represents a combinational circuit if there is a unique, binary output value for every binary input applied to the primary inputs. If there exists an input configuration that can produce two or more distinct output configurations, then the output is not determined uniquely by the circuit inputs and must also depend some signal that is not a circuit input. This indicates sequential behavior.

For output o_j, denote by $I_j(a, b)$ the set of input con-

```
stable_unit-step_relation($R_b$, $SBT$, $SBC$)
1  $R_s = \mathcal{T}_s R_b$;                                           : Initialize the stable unit-step relation
2  $R_{su} = R_b.SBC.\overline{SBT}$                                   : Starting transitions
3  while (TRUE)
4      $A = \mathcal{S}_Z\ (R_{su}(I,X,Z).\mathcal{T}_u R_b(I,Z,Y))$          : Consecutive transitions on same input
5      $B = [Z \rightarrow Y]\mathcal{S}_Y\ (R_{su}(I,X,Z).\mathcal{T}_u R_b(I,Z,Y))$ : Initial transition of above pair
6      $A' = A - \mathcal{T}_s A$                                      : New set of starting transitions
7      if ($\mathcal{T}_s A \subset R_s$) break;                       : Terminate if no new transitions
8      $R_s = R_s + \mathcal{T}_s A$                                   : Update stable unit-step relation
9      $R_{su} = (R_{su} - B) + A'$                                    : Update set of starting transitions
10 end while
11 return $R_s$
```

Figure 4: Algorithm to compute the stable unit-step relation

ditions for which the output variables (o_j^H, o_j^L) evaluate to (a,b). This is computed from the stable unit-step relation.

$$I_j(a,b) = \mathcal{S}_{XY}(R_s(I,X,Y).(o_j^H \equiv a).(o_j^L \equiv b))$$

We first check that the sets $I_j(0,0)$ and $I_j(1,1)$ are empty, i.e., that there is no input condition for which the output o_j is non-binary. We then apply two checks — the **completeness** check and the **uniqueness** check for output o_j. We say that an output o_j passes the *completeness test* if $I_j(1,0) + I_j(0,1) \equiv \mathcal{B}_I 1$, i.e., the output must evaluate to a binary value for all binary inputs. The *uniqueness test* verifies that $I_j(1,0).I_j(0,1) \equiv \emptyset$, i.e. the input patterns producing a binary 1 and 0 at the output are mutually exclusive. If both these checks are satisfied the output o_j is declared combinational. The logic function for output o_j is the Boolean function whose on-set is $I_j(1,0)$ and offset is $I_j(0,1)$. Note that this function is defined only in terms of the primary inputs of the circuit.

The classification procedure makes no assumption regarding the circuit topology as it operates only on the stable unit-step relation. It is the pre-processor ***Tranalyze*** that handles cyclic combinational circuits by introducing unit-delay elements to break the feedback edges. It is an interesting problem to compare our approach with the condition stated in [6] for determining if a cyclic circuit is combinational.

When a relation R_s fails the combinational test, the analysis may be repeated after specifying the clock signals, if any. The next section discusses how we handle sequential circuits.

3.5 Identifying Synchronous Circuits

There are two categories of sequential circuits — synchronous and asynchronous. Synchronous operation is defined in terms of a set of distinguished “clock” signals. Hence it is necessary for the designer to identify the clock signals and specify their relationship to one another.

For ease of exposition we assume that circuits have two-phase clocking (two distinct configurations of the clock signals repeated indefinitely). The set of clock inputs, C, is a subset of the circuit inputs, I. The clock signals take on two distinct configurations, *phase1* and *phase2*. The two clock phases are represented by their characteristic functions, ϕ_1 and ϕ_2 respectively, which are obtained by forming a product (with the appropriate polarity) of the clock signals during that phase. As an example, in a master-slave clocking scheme, where the master-clock, MCK, and the slave-clock, SCK, are complementary, $\phi_1 = MCK.\overline{SCK}$ and $\phi_2 = \overline{MCK}.SCK$.

Our approach classifies the signals that comprise the circuit configuration as either *well-defined* or *undefined*. A signal is **well-defined** if it is a combinational or level-sensitive function of other signals. A combinational signal is independent of the clock phase, while a signal is level-sensitive if its value remains unchanged during one clock phase. All other behavior cannot be represented in an RTL model and we call the signals that exhibit such behavior **undefined**.

When all circuit outputs are *well-defined* functions of signals that are themselves *well-defined*, we get a representation of the output that is equivalent to an RTL model. By using a constructive algorithm, we are assured that circuits for which an RTL model is derived do embody the extracted behavior.

The stable unit-step relation, $R_s(I,X,Y)$, for the circuit was computed earlier. We focus on a single component y_i in the next-configuration space, NC. The relation $R_s^i = \mathcal{S}_{(NC-y_i)} R_s$ describes the value of the signal y_i as a function of the circuit inputs (data and clock signals) and the present configuration. From R_s^i we obtain two relations $A_i^{\phi_1} = \mathcal{S}_C(R_s^i.\phi_1)$ and $A_i^{\phi_2} = \mathcal{S}_C(R_s^i.\phi_2)$. Relation $A_i^{\phi_1}$ ($A_i^{\phi_2}$) describes the behavior of node n_i during phase1 (respectively, during phase2). The sets SBC^{ϕ_1} and SBC^{ϕ_2} represent the stable binary configurations during phase1 and phase2 respectively.

By examining the relations $A_i^{\phi_1}$ and $A_i^{\phi_2}$ we are able to determine if node n_i is *well-defined*. The test for a node being well-defined consists of first checking whether the node represents a combinational function, failing which we check if the node represents level-sensitive behavior.

Combinational test: For a signal y_i to be represented as a combinational function, its value must be independent of the clock configuration. Since the domains over which $A_i^{\phi_1}$ and $A_i^{\phi_2}$ are defined (SBC^{ϕ_1} and SBC^{ϕ_2} respectively) may differ, we check for equivalence only on the common part of the domains. Node n_i is combinational if the following equality holds.

$$A_i^{\phi_1} \cap SBC^{\phi_1} \cap SBC^{\phi_2} \equiv A_i^{\phi_2} \cap SBC^{\phi_1} \cap SBC^{\phi_2}$$

If this test is satisfied, we derive the logic function f_i for this node. A relation that captures the node value at all configurations of interest is minimized with respect to the total set of stable binary configurations.

$$A' = (A_i^{\phi_1}.SBC^{\phi_1}) + (A_i^{\phi_2}.SBC^{\phi_2})$$
$$f_i = min(A', (SBC^{\phi_1} + SBC^{\phi_2})\ )$$

Level-sensitive test: When a node n_i fails the combinational test we check if it is level-sensitive. In a level-sensitive latch, the output value is allowed to change during one clock phase and is held constant during the rest of the clock cycle. For node n_i to be classified as level-sensitive, *exactly one* of $A_i^{\phi_1}$ and $A_i^{\phi_2}$ must exhibit this behavior. Suppose that

$$A_i^{\phi_1}.(y_i \neq x_i) \equiv \emptyset$$

This implies that during phase1, there is no condition for which $(y_i \neq x_i)$. Alternately, it means that the value of n_i does not change during phase1 and so it is inactive during phase1. This allows us to classify node n_i as level-sensitive, active during phase2 (the other phase). The function f_i that computes y_i during phase2 is derived by minimizing $A_i^{\phi_2}$ with respect to the care set of stable binary configurations, $f_i = min(A_i^{\phi_2}, SBC)$. A similar test, applied to $A_i^{\phi_2}$ can be used to see if node n_i is level-sensitive, active during phase1. Note that if the node n_i is inactive during both phases then there is no equivalent memory device that can model it and so the node is *undefined*. ■

Once all the components have been classified, we perform a structural check on the netlist to ensure that no output depends on a component that has been classified as *undefined*. When all outputs are independent of the *undefined* nodes, we substitute for each component the appropriate functional block (complex logic followed optionally by a level-sensitive latch of the appropriate clocking) to generate the RTL model.

4 Implementation and Results

The algorithm described in Section 3 has been implemented in a program called ***Smelt***. ***Smelt*** invokes ***Tranalyze*** to perform the pre-processing of the transistor netlist to derive the unit-step relation. All the sets and relations are represented as BDDs and the computations in the algorithm are performed using the procedural interface to a BDD library. The result of the extraction is a circuit composed of logic gates and latches. *The run-times reported in this section are on a SPARC20 with 512MB of RAM.*

4.1 Characterizing a cell library

The first set of experiments we undertook was to see if ***Smelt*** is able to extract the functionality of the cells in the AT&T standard-cell library [7]. The cells have transistor netlists that have been extracted from their layout as well as logical descriptions provided by the library designer. Our goal was to extract the functionality from the transistor netlist. The diversity in the cell library provides a rich set of examples for the extraction algorithm. This is viewed as check for both the abstraction algorithm and for the implementation of the algorithm in ***Smelt*** .

Combinational cells in the cell library are implemented in a static CMOS style, with the PMOS network being the dual of the NMOS network. Using ***Smelt*** *, we were able to correctly recover the functionality for all 207 combinational cells in the standard-cell library*. A majority of the examples (169 cells) are extracted in less than a couple of seconds each, the total time for these examples is 200 seconds. For these small cells, ***Tranalyze*** produces descriptions that are acyclic and that have no unit-delay elements. Performing abstraction on such descriptions is trivial.

For the remaining 38 examples, the descriptions produced by ***tranalyze*** had more than 9 unit-delay elements. These are introduced to model charge sharing among the transistors. In particular, for the cell AOI4444, ***Tranalyze*** produces a description containing 16 unit-delay elements and 235 2-input primitives (see [2] for a description of the primitives). Clearly, such gate-level models are unacceptable, both in terms of their structure and complexity. When we apply our abstraction procedure, the charge-sharing transients are removed in the computation of the stable unit-delay relation, and we do indeed recover the correct functionality for these gates. However, due to the large number of BDD variables (each unit-delay element requires six BDD variables) the run times for these 38 examples totaled 54,000 seconds.

Sequential cells in the cell library include level-sensitive latches, edge-triggered flip-flops, master-slave latches as well as set-reset (RS) latches. Of the 130 sequential cells, 60 cells have either asynchronous inputs (asynchronous preset and/or asynchronous clear) or have no clocks (like RS latches). ***Smelt*** *was able to correctly identify all cells that had asynchronous behavior*. This accomplishment is important since it gives us confidence that the tool is able to correctly distinguish between synchronous and asyn-

Example	Transistors	Modules	Time(sec)
PE	1183	9	266
POLUNIT	4114	26	281
CPONTROL	4908	16	239

Table 1: Results of abstraction of full-custom designs

chronous circuits. For the remaining 70 synchronous cells, we extract an equivalent RTL description. The abstracted functionality has been automatically checked for consistency with the independently provided RTL descriptions that are contained in the cell library. The total time required to process all the 130 cells was only 170 seconds.

4.2 Characterizing full-custom modules

The library-characterization experiment was a success but it does not exploit the power of the extraction algorithm since the cell library design follows a rigid design discipline. Furthermore, a pattern-matching techniques can easily be constructed to match the cells in a given library. We therefore tested the abstraction on custom-designed cells.

The 6-transistor XOR gate ([8] Figure 8.11) presents problems to many switch-level simulators. ***Tranalyze*** produces a description which has one unit-delay element. ***Smelt*** extracts the XOR functionality for this circuit. We also derive the correct functionality for transmission-gate based adder structures ([8] Figure 8.13). Cyclic combinational circuits presented in [6] were also correctly extracted by ***Smelt***.

For large circuits it is impractical to perform extraction on the entire circuit. We exploit the design hierarchy to overcome this limitation. We have developed a heuristic to determine the modules that need to be characterized. Using this divide-and-conquer approach we have characterized parts of commercial full-custom designs. These include: **PE** which is one of the processing elements that make up an 8x8 array which performs motion-estimation in a video-encoder, **POLUNIT** which is composed of arithmetic functions and implements a leaky-bucket algorithm for rate control in Asynchronous Transfer Mode (ATM) traffic management, and **CPONTROL** which is the controller of a SPARC microprocessor core.

Table 1 shows data on the full-custom designs described above, as well as the run-time. Even though these circuits are of moderate size, their functionality can be described in terms of a fairly small set of modules. The run-time includes all of the processing — reading in the designs, determining which modules to characterize, performing the abstraction and finally writing out an equivalent logic description in terms of logic gates and latches.

5 Conclusions

We have presented a technique for deriving register-transfer models from transistor netlists that is based on examining the switch-level relation. Before deriving RTL models we first apply tests to determine whether the circuit is combinational or synchronous. For these circuits an equivalent RTL model is derived based on the functionality that is captured by a switch-level simulator. To the best of our knowledge, tests for determining if a circuit has an RTL model have not been available in the past. Consequently, existing techniques could not guarantee the correctness of the extracted models. The technique described here is constructive, and the extracted circuits capture the behavior modeled by the switch-level relation. *We emphasize that our contribution in this paper is the algorithm that classifies the switch-level relations and not the pre-processing required to generate the switch-level relation from the transistor netlist*.

References

[1] P. Agrawal, S. H. Robinson, and T. G. Szymanski. Automatic Modeling of Switch-Level Networks using Partial Orders. In *IEEE Transactions on Computer-Aided Design*, pages 696–707, July 1990.

[2] R. E. Bryant. Extraction of Gate Level Models from Transistor Circuits by Four-Valued Symbolic Analysis. In *Proceedings of the International Conference on Computer-Aided Design*, pages 350–353, 1991.

[3] R. E. Bryant. Boolean analysis of MOS circuits. In *IEEE Transactions on Computer-Aided Design*, pages 434–469, July 1992.

[4] P. Deverchere, J. C. Madre, J. B. Guignet, and M. Currat. Functional Abstraction and Formal Proof of Digital Circuits. In *The Proceedings of the European Conference on Design Automation*, pages 458–462, March 1992.

[5] T. Kam and P. A. Subrahmanyam. Comparing Layouts with HDL models: A Formal Verification Technique. In *IEEE Transactions on Computer-Aided Design*, pages 503–509, April 1995.

[6] S. Malik. Analysis of Cyclic Combinational Circuits. In *Proceedings of the International Conference on Computer-Aided Design*, pages 618–625, 1993.

[7] AT&T Microelectronics. *Standard Cells and Function Blocks (0.9 micron CMOS)*. AT&T, 1990.

[8] Neil H. E. Weste and Kamran Eshraghian. *Principles of CMOS VLSI design*. Addison-Wesley, 1992.