Verification of VLSI Circuits: Signal Value Modeling and HDL Translation

Jody W. Gambles
Microelectronics Research Center, University of New Mexico, Albuquerque NM 87131

Phillip J. Windley
Laboratory for Applied Logic, Department of Computer Science Brigham Young University Provo UT 84602[1]

Abstract. The ever increasingly number of transistors possible in VLSI circuits compounds the difficulty in ensuring correct designs. Formal verification has been touted as a means of overcoming this problem, but most of the work in this area has been directed at standalone theorem provers. Before verification is accepted by the VLSI design community, the stand alone verification tools that are in use in the research community must be integrated with the CAD tools used by designers. The primary obstacle is that the hardware description languages (HDL's) used in verification tools are often not syntactically or semantically compatible with the hardware description languages commonly used in the VLSI design community.

The research presented in this paper is directed at this problem. We have built a parser for the BOLT hardware description language in the HOL theorem proving system. This provides a means of using the same language in the theorem proving system and the VLSI CAD tools used in our work. We have also built a semantic model in HOL for the signal values used in the simulator in our CAD tool suite. We show how this semantic model can be used for reasoning about the signal value data types that are used in the BOLT hardware description language and the VHDL Standard Logic Package.

1 Introduction

As higher transistor counts increase the complexity of VLSI circuits and the number of potential test cases explode, traditional simulation methods can expose only a fraction of design faults. Formal methods, such as symbolic simulation, state machine analysis, and theorem-proving provide a way to verify the correctness of a circuit design. Each of these methods have their own strengths and weaknesses. Symbolic simulation is well suited to low level circuit verification but the computational requirements can become prohibitive for large devices at higher levels of abstraction. The effort required to direct a theorem prover has forced most attempts at theorem-proving to be directed toward more abstract levels. These factors have led to suggestions that

[1]The work reported in this paper was performed while both authors were at the University of Idaho.

symbolic simulation and theorem-proving could be combined in a complementary manner to play an important role in design fault exclusion [17].

It is common in modern design methodologies to utilize abstract circuit models in a hierarchical design:

- An architectural model (i.e. highly abstract) can be used to simulate an entire system, at an early date, to help confirm that the system specification truly meets the customers needs.
- In a top-down design, a model of the system's architecture is decomposed iteratively from algorithmic description, to large functional blocks, to detailed logic, and on down to the circuit level.
- After the circuit structure is modeled and designed, the logic simulation of complex systems can become very slow. Large simulations may run faster using behavioral models.

A problem with these design approaches is that there is no *formal* way to relate a circuit's structural model to its abstract behavioral model. Formal verification allows these models to be related through mathematical analysis so that designers can enjoy increased confidence that behavioral models are *correct* abstractions of their structure. Before formal verification is accepted by design engineers, stand alone verification tools that are used in academic research must be integrated with the CAD tools used by VLSI designers.

The hardware description languages (HDL) used by VLSI CAD tools can provide the link between these tools and the verification environment. Engineers can design using an HDL and the models can be automatically translated for use in the verification tool. The translation process consists of two steps.

- Parsing the concrete syntax of the HDL to an internal abstract syntax.
- Mapping the abstract syntax to the HDL's semantic domain.

The parser, for recognizing the syntax, and the semantic construction functions can be built directly into the verification system.

This paper describes the results of integrating the NOVA simulation engine, one of the CAD tools being developed and used at the NASA Space Engineering Research Center (SERC) for VLSI Systems Design with the HOL theorem proving system. NOVA simulates designs described in the hardware description language BOLT (Block Oriented Logic Translator). BOLT was chosen as a starting point for this research because it provides ready access to many real-world VLSI designs at the SERC. The BOLT parser was built using the parser-generator library provided with the HOL system [19]. The semantic construction functions were written in ML, the language of the HOL theorem prover.

2 Related Work

Much has been published about theories for modeling MOS circuits in a verification environment [7, 9, 11, 15, 22]. Our work linking verification with VLSI design tools is related, but has a different motivation. While we are concerned that the model accurately reflect the true behavior of the devices being specified, we must also be concerned that the HOL circuit primitive definitions are consistent with the HDL primitives.

Work has also been published dealing with translating HDLs to verification logics [4, 20]. Our interest in tying HOL to BOLT and the NOVA simulator has motivated our work to include lower level primitives and indeterminate strength signal values and their resolution functions, where these problems have been largely ignored by others. These signal values are also useful in modeling the interconnection of higher level system blocks via tri-state busses. In order to show that our theories and methods have utility beyond the scope of BOLT, we also present an extended lattice representation of the VHDL Standard Logic Package signal value data type and resolution function.

Our work is done in the HOL verification system. HOL is a general theorem proving system developed at the University of Cambridge [7, 12] based on Church's theory of simple types. Higher-order logic is suitable for specifying all aspects of hardware, including both structure and behavior [12, 14]. HOL is not an automated theorem prover but is more than simply a proof checker. It could, more appropriately, be called a proof assistant.

3 Signal Value Data Type

Few modern VLSI circuits are designed using only classical logic gates [6, 16]. In designs using pass-transistor, tri-state, and pre-charge logic, it is common for circuit nodes to be driven from multiple circuit devices. These multiple drivers are designed to have differing drive strengths in order for one to dominate over another in cases of contention. The drive strength is considered to be closely related to current drive or charge sourcing capability [5, 13]. These design techniques dictate that digital logic circuit models and simulations must include devices down to the transistor level. Transistors, at the logic level, are modeled as simple switches, with electrical properties such as threshold voltage abstracted away. The physical attributes such as transistor size can be represented by the strength of the signal that the transistor drives. The signal values used in the NOVA simulation engine are an extension of Bryant's lattice theoretic approach, where the elements in the set of signal values represent the combination of logic state (i.e. one of $\{1, 0, X\}$) and signal strengths[13, 18, 6]. The signal values form a partially ordered set with their order based on strength dominance when circuit output values are interconnected.

Bryant abandoned the logic state/strength combination approach to signal values in favor of switch graphs and path blocking for his MOSSIM II simulator [5], stating "while this approach at first seems very elegant, it cannot adequately describe the effects of transistors in the X state." Cameron and Shovic have shown that the problem with the X state can be overcome by extending the range of signal values to include some degree of strength indeterminacy [6].

The lattice extension may have an advantage over the switch graph approach in that the switch graph approach *requires* working at the transistor switch-level. The extended lattice approach supports accurate modeling at the switch-level but also allows higher level models. The extended lattice model can also accurately model tri-state buses and combined system blocks implemented with different technologies such as TTL, NMOS, CMOS, and ECL [8].

The unknown state can be the result of a node connected to two drivers, one driving to a 1 and the other driving to a 0, neither driver having sufficient strength to dominate the other (i.e. an invalid logic level). In a logic simulator, the unknown state can also be used to model a node whose voltage is not yet known. Combining the cases of *invalid* logic level and *valid but not known* into a single X state simplifies the simulation algorithm but may make the simulator pessimistic since it will propagate the X state when resolving some circuit nodes [5].

We call the data type representing the combination of state and strength STATE. The STATE representation presented here follows from that presented in [6, 16] except the total number of strengths, N, is extended to include a weakest strength, Nil, which represents a node that is disconnected from all charge sources. By definition, a signal being driven by the Nil strength must be at the unknown state.

3.1 Representation of STATE

Given the set of states $\{1, 0, X\}$ and a fully ordered set of strengths $\{\sigma_1, \sigma_2, \ldots, \sigma_N\}$ we can define the data type STATE. The STATEs corresponding to the 1 and 0 states are represented as a triple **Kbd** where:

K is 1 or 0;

bd represents a *range* of strengths where:

- **b** is the strongest possible strength ($\sigma_1 \leq \mathbf{b} \leq \sigma_{N-1}$) which sets a lower bound on the strength of a signal that can overdrive this state;
- **d** is the weakest possible strength ($\mathbf{b} \leq \mathbf{d} \leq \sigma_{N-1}$) which sets a upper bound on the strength of a signal that this state can overdrive.

The STATEs corresponding to the X state are represented as a triple **Xpq** where:

p is the strongest possible strength driving toward 0 ($\sigma_1 \leq \mathbf{p} \leq \sigma_{N-1}$) which sets a lower bound on the strength of a signal that can overdrive this state to a 1;

q is the strongest possible strength driving toward 1 ($\sigma_1 \leq \mathbf{q} \leq \sigma_{N-1}$) which sets a lower bound on the strength of a signal that can overdrive this state to a 0.

3.2 The Number of STATEs

For N strengths the number of 1 and 0 STATEs is:

$$\begin{aligned} TF\_STATES(N) &= 2((N-1)+(N-2)+\ldots+1) \\ &= (N-1)(N) \end{aligned} \tag{1}$$

For the X state the number of STATEs is:

$$X\_STATES(N) = (N-1)^2+1 \tag{2}$$

The plus one term in equation (2) represents the combination of unknown state and weakest strength, $\sigma_N = Nil$. This STATE is referred to as `Nil`. Thus, the total number of STATEs for N strengths is equal to:

$$TOTAL\_STATES(N) = 2N^2 - 3N + 2 \tag{3}$$

4 STATE Theory

At the device primitive level, the semantic domain of BOLT is the signal value data type. A complicated imperative style resolution algorithm, for determining the result of interconnecting

STATEs, is presented in [6]. The algorithm is intended for circuit simulation programs and is not satisfactory for reasoning about circuits in HOL. Other simulators simply use look-up tables (LUTs) to perform signal value resolution [8], again not suitable for reasoning.

To facilitate reasoning about signal values, we developed a lattice theoretic representation od the STATEs datatype. In the lattice approach, the function for resolving the interconnection of two circuit elements is the *join* [2] of the values being driven by the two elements [18]. The ability to reason about low level signal values enhances the flexibility of the symbolic simulation to theorem prover interface. The signal value lattice theory is used in the BOLT to HOL translator.

A lattice structure is described through the notion of immediate superiors or *covers*. For two elements, a and b of a partially ordered set (poset), *a covers b* if and only if $a > b$ and there exists no element x of the poset such that $a > x > b$ [3]. A list of all of the members of the poset and their covers completely describe a lattice and can be used to define a graph of the lattice. The vertices of the graph are the elements of the poset and the segments or edges of the graph represent the covers. If the graph is drawn such that whenever x covers y, the vertex x is *higher* than the vertex y, it is called a *Hasse diagram* of the lattice [3].

4.1 Defining the STATE Lattice Structure

Given the base case[3], $N = 2$, there are four STATEs and no strength indeterminacy. There is only a single value (σ_1) within the range of possible strengths and there are four covers. The base case STATE Hasse diagram is just as those presented in [13, 18, 22], a simple diamond (see Figure 1).

To extend a N strength STATE lattice to $N + 1$ strengths:

1. Add three STATEs and four covers to form a new diamond at the bottom of the N strength diagram by replacing `Nil` with $X\sigma_N\sigma_N$, adding $0\sigma_N\sigma_N$ and $1\sigma_N\sigma_N$ each covered by $X\sigma_N\sigma_N$ and placing `Nil` at the bottom of the diagram covered by both $0\sigma_N\sigma_N$ and $1\sigma_N\sigma_N$.

2. For each $M = N$ to 2, by -1, add the following STATEs and covers:

    (a) $X\sigma_{M-1}\sigma_N$ covered by $0\sigma_{M-1}\sigma_{N-1}$ and covering $X\sigma_M\sigma_N$

[2] The lattice join function is the least upper bound of elements.

[3] $N = 1$ is a trivial case of one single STATE, `Nil`.

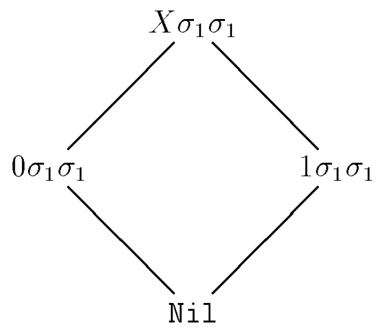


Figure 1: STATE Lattice Base Case (N=2)

(b) $X\sigma_N\sigma_{M-1}$ covered by $1\sigma_{M-1}\sigma_{N-1}$ and covering $X\sigma_M\sigma_N$

(c) $0\sigma_{M-1}\sigma_N$ covered by $X\sigma_{M-1}\sigma_N$ and covering $0\sigma_M\sigma_N$

(d) $1\sigma_{M-1}\sigma_N$ covered by $X\sigma_N\sigma_{M-1}$ and covering $1\sigma_M\sigma_N$

The total number of covers for N strengths is equal to:

$$COVERS(N) = 4N^2 - 10N + 8 \tag{4}$$

4.2 The Lattice Structure for NOVA

In BOLT and the NOVA simulation engine, $N = 4$ with $\sigma_1 = a$ (active), $\sigma_2 = r$ (resistive), $\sigma_3 = f$ (float), and $\sigma_4 =$`Nil`. Note that float $>$ `Nil` and can be used to represent signal levels at charged capacitive nodes. For $N = 4$, equation (3) yields 22 STATEs and equation (4) yields 32 covers. The Hasse diagram for the NOVA STATEs is shown in figure 2. In addition to providing a framework for reasoning about signal values and their resolution function, the Hasse diagram also provides a useful, quick, visual understanding of the interconnection of STATEs during manual circuit analysis.

5 Implementing the STATE Type in HOL

The STATE data type represents the semantic domain for the BOLT expressions we wish to translate. We define the `STATE` data type in HOL by enumerating the members of the lattice

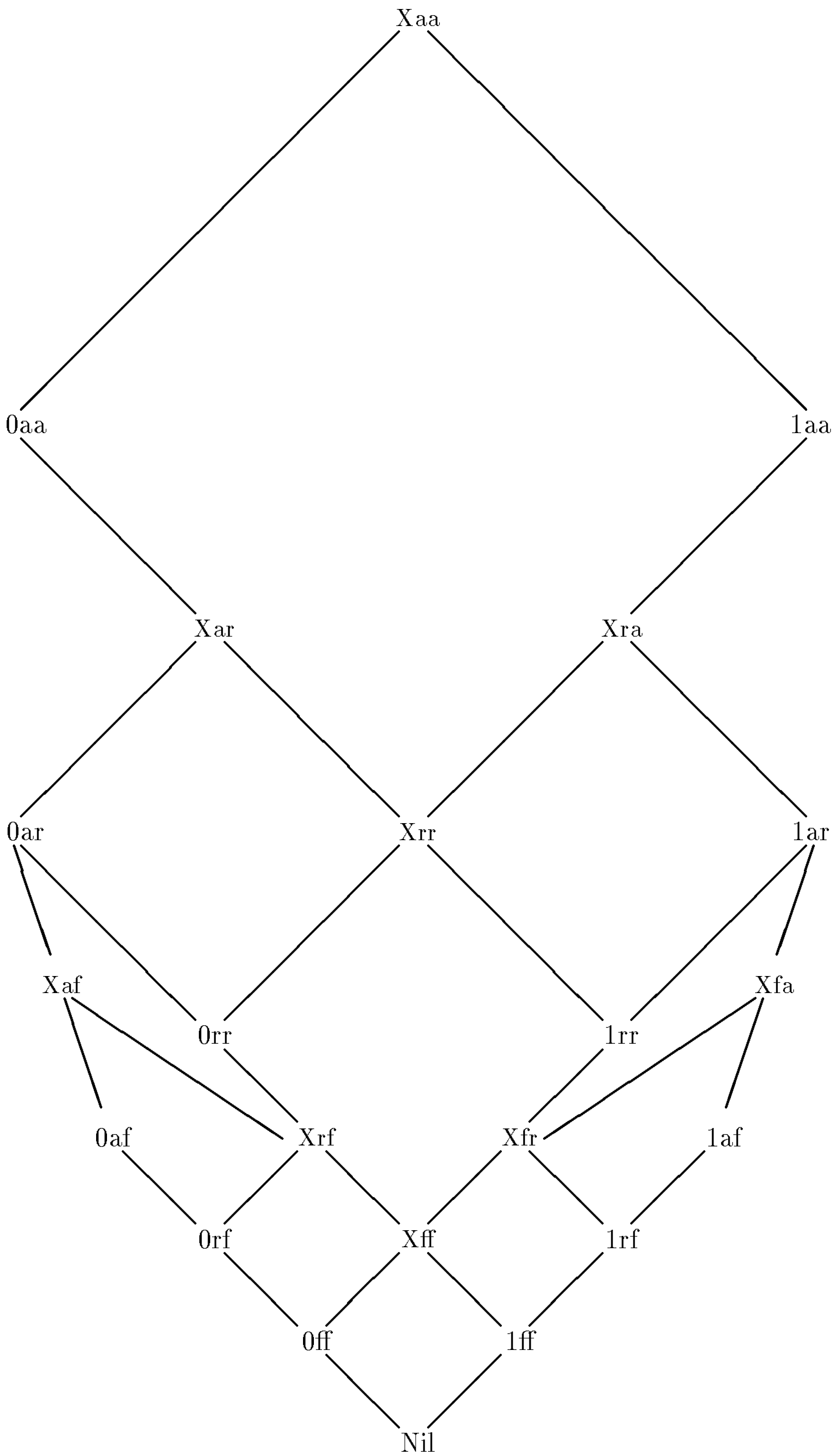


Figure 2: STATE Lattice for NOVA (N=4)

and their covers. Using these enumerations, we define the binary function `join` to be the least upper bound of two input arguments. In the circuit, the `join` function represents the result of interconnecting the two input signals.

To enable reasoning about the `STATE` signal lattice, we also prove the following theorems:

1. Idempotence. $\forall a.$ `join` $a\ a = a$.
2. Commutativity. $\forall a\ b.$ `join` $a\ b =$ `join` $b\ a$.
3. Associativity. $\forall a\ b\ c.$ `join` a (`join` $b\ c$) $=$ `join` (`join` $a\ b$) c.
4. Existence of bottom. $\forall a.$ `join` a `Nil` $= a$.

5.1 STATES Abstraction Function

At some level in a proof hierarchy, we wish to state the properties of the circuit in terms of boolean data types and show that the circuit based on the `STATE` data type correctly implements it. We define a function that embodies that abstraction:

```
⊢def STATES_ABS sig =
      (((sig=laa)∨(sig=lar)∨(sig=lrr)∨
        (sig=laf)∨(sig=lrf)∨(sig=lff)) →  T |
      (((sig=0aa)∨(sig=0ar)∨(sig=0rr)∨
        (sig=0af)∨(sig=0rf)∨(sig=0ff)) →  F |
                                         ARB))
```

The unknown `STATE`s are assigned an arbitrary boolean value `ARB`.

6 Lattice Structure for VHDL Standard Logic Package

While we have selected BOLT for this research, it seems appropriate to examine how the lattice theory extension applies to the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL). Approved by the Institute of Electrical and Electronic Engineers (IEEE) in December, 1987, as IEEE Standard 1076-1987, VHDL has been adopted for design tools throughout the electronics industry. The U.S. Department of Defense MIL-STD-454L dictates the use of VHDL to specify and document all application specific integrated circuits (ASICs). Dozens of software tools based on VHDL are available [21]. The VHDL Standard Logic Package (SLP) defines a signal value data type, **t_wlogic**, containing 46 unique values [8]. There are five strengths represented in the SLP:

- Forced (F).
- Strong resistive (R).
- Weak resistive (W).
- High impedance (Z).
- Disconnect (D).

The SLP data type maps onto a $N = 5$ STATE lattice as defined in Section 3.1 with two additional extensions:

1. The U value is included to represent a *valid but not known* (uninitialized) value.[4]
2. The strengths are allowed to range over *all* possible strengths, including the Nil strength, D.[5]

The nomenclature of the **t_wlogic** values is different than that defined for STATEs in Section 3.1. The **t_wlogic** value representations include either one, two, or three alpha-numeric characters. The three character 1 and 0 values represent signals with a finite range of possible strengths; the first character represents the strongest possible strength, the second is the weakest possible strength, and the third is the logic state. These values are the same as those defined for STATEs except the order of the triple is changed to **bdK**. For example, FR1 represents a 1 state that may be as strong as forced, but may be as weak as resistive. The three character X values are also the same as those defined for STATEs, except the order is changed to **pqX**. In the STATE cases where $\mathbf{b} = \mathbf{d}$ or $\mathbf{p} = \mathbf{q}$ so there is no strength indeterminacy, the **t_wlogic** value is truncated to two characters; **b**1, **b**0, or **p** X. There are two single character values; the uninitialized value, U, and the `Nil` or disconnect value, D.

Figure 3 shows a Hasse diagram for the VHDL Standard Logic Package **t_wlogic** data type. As an indication of the expressive power of this lattice treatment, the information expressed in Figure 3 requires a 46 X 46 look-up table, covering nearly five pages, in *The VHDL Handbook* [8].

[4]It may be useful while a design is being debugged for a simulator to be able to distinguish uninitialized nodes from nodes that have an invalid state due to device conflicts. However, combining the cases of invalid logic level and valid but not known simplifies the simulation algorithm. A rigorous modeling of the effects of unknown but valid states would require determining Boolean satisfiability, an NP-complete problem [5].

[5]While the structure of a circuit clearly distinguishes between a node that is connected and one that is disconnected, it may be considered to be more mathematically rigorous to include all possible strength ranges.

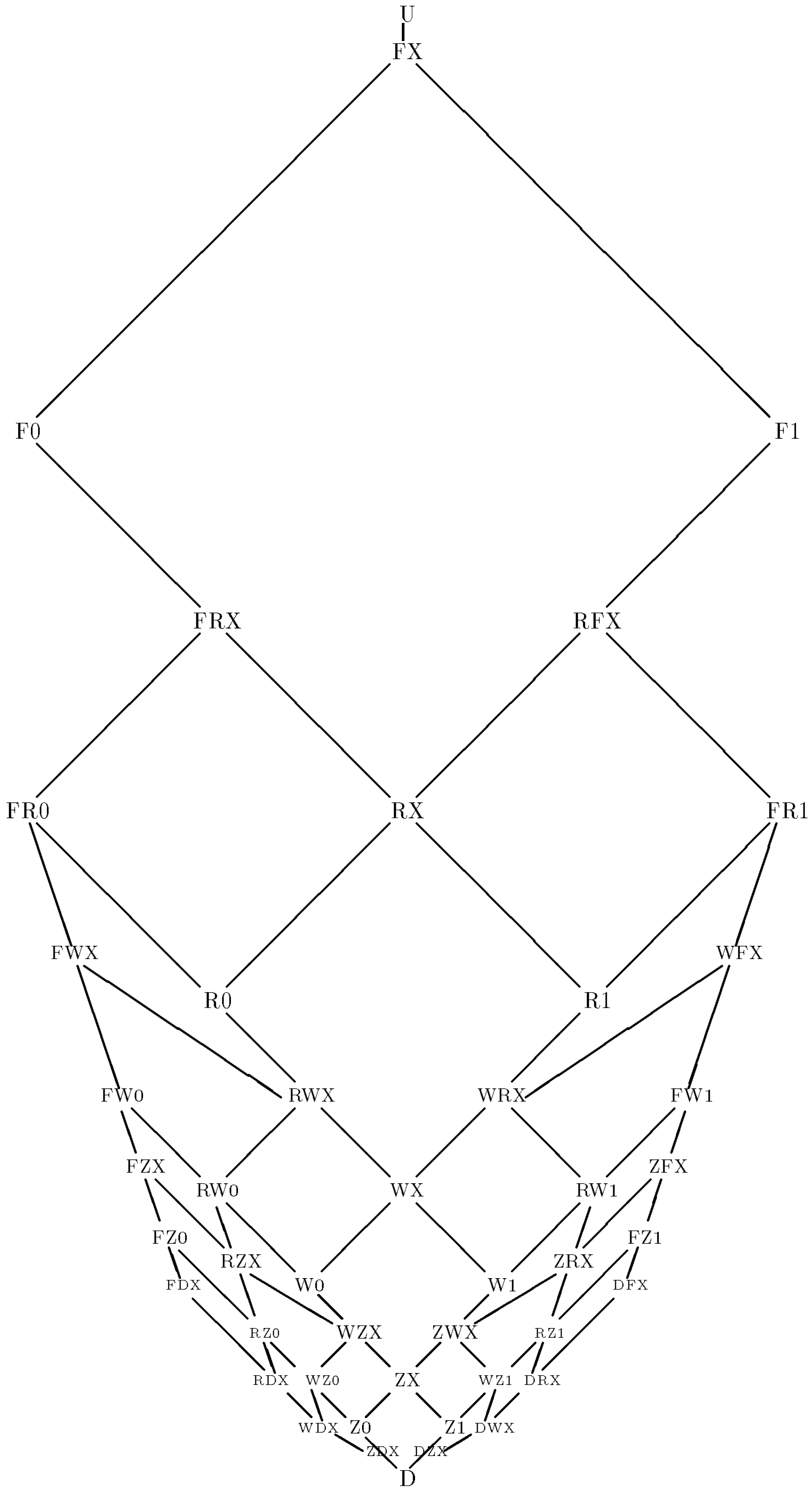


Figure 3: Signal Lattice for VHDL Standard Logic Package

7 The BOLT to HOL Translator

The BOLT to HOL translator is comprised of a syntax parser built using a parser-generator tool included with version 2.0 of the HOL system and a set of ML functions that construct the semantics of the parsed BOLT syntax into HOL definition terms.

7.1 The Syntax Parser

The parser-generator takes as input a grammar representing the formal syntax of BOLT given in a modified Backus Naur Form (BNF) notation similar to Prolog's definite clause grammar (DCG) [19]. The output of the generator is a ML program that recognizes the HDL syntax and makes appropriate calls to the ML semantic construction functions, whose names are included as action symbols in the input grammar. The BOLT syntax is defined in [2] and was converted into the appropriate BNF form. The HOL parser-generator library developed at the University of Cambridge was found to be very useful in building the syntax recognizer portion of the translator.

As an example of the required syntax conversion, the syntax of a statement-body from the BOLT manual is:

```
BEGIN

[ { Module-Invocation | Case-Statement } ... ]

END ;
```

Upper-case words are keywords, the expression `[ ]` contains an optional item that may be omitted, { } indicates choose one of the enclosed items, and ... indicates that an item may be repeated any number of times. This syntax is entered into the parser-generator input grammar as the following recursive production:

```
statement_body  --> [BEGIN] invocation_list [END] [;]

invocation_list --> mod_invocation invocation_list |
                    case_statement invocation_list |
                    [].
```

7.2 The Semantic Construction Functions

No formal semantics exists for BOLT. We defined semantic construction functions using the built–in metalanguage of HOL called ML. Our ML functions are activated by the parser and translate a BOLT description of a circuit into the equivalent HOL representation of the circuit.

In order to translate BOLT modules into HOL, the following data is required:

1. The module name from the BOLT module declaration. The same name is used for the HOL definition.

2. The set of ports declared to be external to the top module in the BOLT declaration. The signals on these ports must be universally quantified in the HOL definition.

3. The names of the component modules that are invoked inside the BOLT module. Each of these module invocations will cause an identically named HOL predicate to be conjoined in the HOL definition predicate.

4. The set of ports declared to be outputs of any invoked module. This set will be used to determine the signals that are driven by more than one device. A `JOIN` predicate must be added to the HOL definition to resolve all interconnected outputs. This set will also be used in the identification of the internal ports that are to be hidden.

   In BOLT, the ports are not explicitly declared to be module input or outputs. By *convention* the outputs are listed before the module name and the inputs are listed following the module name. **Our translator relies on conformance to this convention**.

5. The set of ports declared to be an input to any invoked module. The union of this set and the set of invoked module output ports defines the set of all ports in the module. The set difference of the set of all ports minus the set of externally declared ports is used to define the set of internal signals that must be existentially quantified in the HOL definition.

6. The set of module paramaters for each module invocation. The only BOLT module parameter with any meaning in the definition produced by the translator is the `STR` parameter, which the circuit designer uses to define the output strength of an invoked module. If no `STR` parameter is included in the module invocation then the default strength is active.

During parsing, the ML functions defined as action symbols int he parser construct a data structure representing the abstract syntax. Once the BOLT module `END;` statement is parsed, the abstract syntax is used to the create the HOL definition corresponding to the BOLT module.

As the BOLT module is parsed, each new module invocation adds the module name, output node names, input node names, and optional output strength parameters to the abstract syntax data structure. After parsing is finished, the abstract syntax is subjected to a validity check which determines if the output nodes of the invoked module are already members of the set of

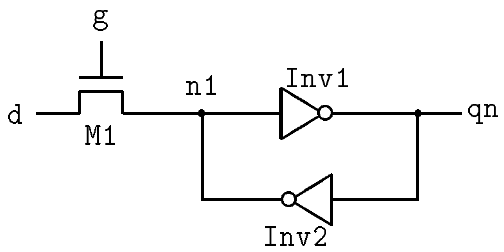


Figure 4: Latch Schematic

internal output nodes for the current module. If they are, then two outputs are connected to drive the signal value on that node and a `join` resolution function from the `STATE` theory is inserted. The `join` function is added by renaming the first instance of that output node name to the decorated (primed) variation of the name and the current invocation output is given the double-decorated node name variation. An invocation of `JOIN` is then added to the end of the data structure where the output of the `JOIN` is the original node name and the inputs are the new decorated and double-decorated nodes.

The set of external output nodes unioned with the external input nodes are universally quantified in the resulting HOL definition. The variables in the set difference of the internal nodes and the external nodes are existentially quantified (hidden).

The translator generates HOL terms by matching each invoked module with a previously defined HOL constant whose name and type both match the structure built by the parser and construction functions. The terms from all of the invoked modules are conjoined to complete the HOL definition for the current module. A stack is maintained for current module data structures so that embedded BOLT modules can be properly defined and translated.

8 Example: A Static RAM Cell

A static RAM cell or data latch (Figure 4), implemented with gate level and pass transistor primitives, provides a simple example of how our system can be used apply verification tools and simulators to the same problem. Although a circuit this small would not normally be the subject of a formal verification effort, this circuit provides an interesting example because without a signal value representation and resolution function that supports output dominance, this circuit cannot be correctly modeled. For this circuit to work correctly, the output strength of pass-transistor `M1` must dominate the output of inverter `Inv2` to force node `n1` to the state of the input `d` while the gate `g` is 1 (high voltage). The feedback inverter `Inv2` acts to store the state by dominating the pass-transistor after the gate goes to 0, thereby turning the transistor off.

8.1 The BOLT Structural Description

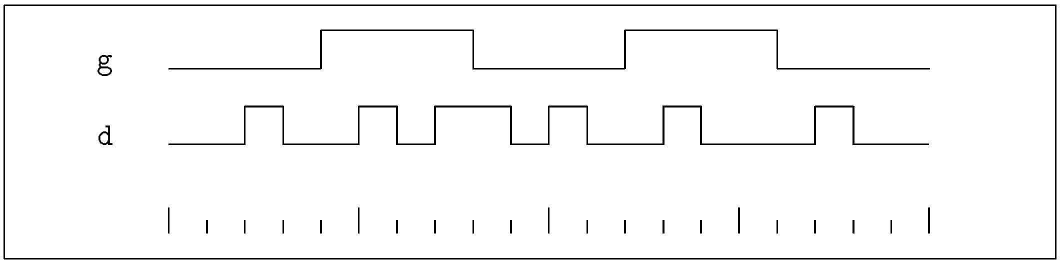
The BOLT description for the latch shown in Figure 4 is:

```
MODULE qn .LATCH g d;
  BEGIN
    n1  .NTRAN g d;
    qn  .INVR  n1;
    n1  .INVR  qn  (STR='RR');
  END;
```

The STR='RR' parameter in the second .INVR invocation defines the output strength of inverter Inv2 as resistive. The default value, used for Inv1, is active.

8.2 Simulating the Latch

The operation of the latch defined in the last section can be tested using the NOVA simulator. The LATCH module was run in NOVA with the following waveforms on the g and d inputs:



The resulting simulator output is shown in Table 1, where the symbol 1 represents 1aa, 0 represents 0aa, and X represents Xaa.

8.3 The HOL Circuit Primitives

The latch structure includes three predicate definitions; an inverter, a NMOS-transistor, and the signal resolution operation, JOIN. These primitive elements must be defined in HOL. This section presents the behavioral specifications for the primitives used in our example circuit.

In HOL, time is represented as a stream of natural numbers (num) and circuit signals are defined to be functions of type (num → STATE). The rest of the notation is that of standard second–order logic except that conditional statements are written a → b | c, representing if a then b else c.

```
        q
    g d n
    * * *
00001>0 0 X
00002>0 0 X
00003>0 1 X
00004>0 0 X
00005>1 0 1
00006>1 1 0
00007>1 0 1
00008>1 1 0
00009>0 1 0
00010>0 0 0
00011>0 1 0
00012>0 0 0
00013>1 0 1
00014>1 1 0
00015>1 0 1
00016>1 0 1
00017>0 0 1
00018>0 1 1
00019>0 0 1
00020>0 0 1
```

Table 1: Latch Simulation Data

8.3.1 INVR Definition

The predicate specification of the inverter has five arguments. The first three arguments are of type STATE and define the possible inverter output strengths. The first is the output STATE for a true output, the second for a false output, and the third the unknown state output. The unknown output value is derived from the strongest true and false strengths. The fourth and fifth arguments are signal functions of type num → STATE. The fourth is the inverter output and the fifth is the input.

```
⊢def INVR 1s 0s Xs (out,in_a) =
     ∀ t. let a_is_0 = ((in_a t =0aa)∨(in_a t =0ar)∨(in_a t =0rr)∨
                        (in_a t =0af)∨(in_a t =0rf)∨(in_a t =0ff)) in
          let a_is_1 = ((in_a t =1aa)∨(in_a t =1ar)∨(in_a t =1rr)∨
                        (in_a t =1af)∨(in_a t =1rf)∨(in_a t =1ff)) in
        (out t = a_is_1 →  0s |
                 a_is_0 →  1s |
                           Xs )
```

The let...in... terms in the definition are local variable assignments used to improve the readability of the definition.

8.3.2 NTRAN Definition

The transistor model has one output port (s), and two input ports (g and d). According to the behavioral abstraction, the value of the source, s is computed relative to three possible classes of input on the gate, g:

1. If the signal value at the gate, g, is equal to any one of the six true STATEs, the source, s is assigned to be equal to the signal at the drain, d.

2. If the gate is equal to any of the six false STATEs, the transistor is not conducting, and the output strength of the source will be a floating state equal to the state value of the previous source.

3. If the value of the gate is unknown, then the state of the source will also be unknown. The strength of the source may be as weak as floating (i.e. the unknown gate value is a false, so the transistor is off) but may be as strong as the drain (i.e. the unknown gate value is a true, so the transistor is passing the value on the drain). The strength of the drain can only be driving toward the state of the drain. The maximum strength toward the opposite boolean state is always float. For example, if the gate is unknown and the drain is an active true, then the source value is unknown, but could be driven toward true

as strongly as active. The strongest drive toward false is the float of a non-conducting transistor. The STATE value in this case is Xfa. Both the NOVA and HOL models fail to correctly model the eventual discharge of a 1ff STATE to a Xff or even Off due to the leakage current at the substrate to source pn junction.

The complete specification of the NTRAN is:

```
⊢def NTRAN (s,g,d) =
     (∀t.
          let g_is_0 = ((g t =0aa)∨(g t =0ar)∨(g t =0rr)∨
                        (g t =0af)∨(g t =0rf)∨(g t =0ff))  in

          let g_is_1 = ((g t =1aa)∨(g t =1ar)∨(g t =1rr)∨
                        (g t =1af)∨(g t =1rf)∨(g t =1ff))  in

         let s_was_0 = ((s(t-1)=0aa)∨(s(t-1)=0ar)∨(s(t-1)=0rr)∨
                        (s(t-1)=0af)∨(s(t-1)=0rf)∨(s(t-1)=0ff))  in

         let s_was_1 = ((s(t-1)=1aa)∨(s(t-1)=1ar)∨(s(t-1)=1rr)∨
                        (s(t-1)=1af)∨(s(t-1)=1rf)∨(s(t-1)=1ff))  in

         let d_is_active_1 = ((d t =1aa)∨(d t =1ar)∨(d t =1af))  in

         let d_is_resistive_1 = ((d t =1rr)∨(d t =1rf))  in

         let d_is_float_1 = (d t =1ff)  in

         let d_is_active_0 = ((d t =0aa)∨(d t =0ar)∨(d t =0af))  in

         let d_is_resistive_0 = ((d t =0rr)∨(d t =0rf))  in

         let d_is_float_0 = (d t =0ff)  in

     (s t = g_is_1       →         d t    |
            g_is_0       →
              (s_was_1     →       1ff |
               s_was_0     →       0ff |
                                  Xff )  |
              (d_is_active_1    →  Xfa |
               d_is_resistive_1 →  Xfr |
               d_is_float_1     →  Xff |
               d_is_active_0    →  Xaf |
               d_is_resistive_0 →  Xrf |
               d_is_float_0     →  Xff |
                                  d t)))
```

8.3.3 JOIN

The `JOIN` predicate performs two tasks.

1. It determines the resulting signal value of a combination of circuit outputs by applying the `join` function.

2. It models sequential behavior of a charge storage node. The capacitance of a node may result in a time delay when the node is driven to a new signal level. The delay increases as the capacitance increases or as the strength of the driving signal decreases. This sequential behavior is modeled as having a variable delay, where the length is based on the strength of the `join` function result [9, 13]. The Hasse diagram shows the relative strength of `STATE`s and can be used to abstract the delay values for individual `STATE`s by segregating them into horizontal bands on the diagram. All `STATE`s within a common band have the same delay and the delay is longer for lower bands. For cases where it is desired to model different delays for rise and fall times the diagram can be segregated right from left also.

We have modeled the `JOIN` as having two possible delays (i.e. we divide the Hasse diagram into two bands). When the pass-transistor is turned on, the storage node at the join is driven by an active strength and the delay is defined to be zero. When the pass-transistor is turned off, the storage node is driven by the resistive strength of the feed-back inverter and the delay is defined to be one. The HOL specification of `JOIN` is given as follows:

```
⊢def JOIN (s,s′,s″) =
    (∀ t. let sig = join (s′ t) (s″ t) in
          (((sig = 0aa) ∨
            (sig = 1aa) ∨
            (sig = Xaa) ∨
            (sig = Xar) ∨
            (sig = Xra)) →  (s t = sig)     |
                             (s (t+1) = sig)))
```

8.4 Translating BOLT Structural Specifications

In our system, HOL structural specifications are obtained by translating BOLT descriptions. The translator may be invoked to operate on a file containing the BOLT description or on BOLT text included between the keywords `BEGIN_BOLT` and `END_BOLT` within the HOL operating environment.

The BOLT description of the latch is given as:

```
BEGIN_BOLT
  MODULE qn .LATCH_IMP g d;
    BEGIN
      n1  .NTRAN g d;
      qn  .INVR  n1;
      n1  .INVR  qn  (STR='RR');
    END;
END_BOLT
```

When the HOL system reads this description the translator produces the following HOL definition for the latch specification:

```
⊢def LATCH_IMP (qn,g,d) =
    (∃ n1 n1′ n1″.
       NTRAN (n1′,g,d) ∧
       INVR laa Oaa Xaa (qn,n1) ∧
       INVR lrr Orr Xrr (n1″,qn) ∧
       JOIN (n1,n1′,n1″))
```

This definition is usable in the HOL system in the same way as any native definition.

8.5 The Behavioral Description

The behavioral specification is written in terms of boolean signal values, not the more complex STATE data type since we desire to make the description as abstract as possible at each level. The HOL behavioral description is:

```
⊢def LATCH_SPEC (qn,g,d) =
    (∀ t.
      (g t →  (qn t = ¬d t)     |
              (qn (t+1) = qn t)))
```

When the gate of the pass-transistor is true, the latch is enabled and the output, qn, follows as the inverse of d. When the gate is false, the latch stores the previous data.

8.6 The Latch Verification

The verification of the latch entails proving that the latch structural description, along with two validity conditions, logically imply the behavioral specification.

Validity Condition 1: Input is Boolean Active The proper operation of the latch requires that the output of the pass-transistor dominate the resistive strength output of Inv2. The pass-transistor is not an amplifier so there is a validity condition that the signal applied to input d must be stronger than resistive.

```
⊢def Is_bool_active (d) =
    (∀ t. (d t = 1aa ) ∨ (d t = 0aa))
```

Validity Condition 2: Gate is Boolean Because the behavior of the latch is defined only for boolean value signals at the gate, there is a validity condition for the gate that it be either a 1 or 0 state. This condition yields a 12 way case analysis in the proof that is easily reduced to considering only the two cases of enabled and latching.

```
⊢def Is_bool (g) =
    (∀ t.
      (g t = 1aa) ∨ (g t = 1ar) ∨
      (g t = 1rr) ∨ (g t = 1af) ∨
      (g t = 1rf) ∨ (g t = 1ff) ∨
      (g t = 0aa) ∨ (g t = 0ar) ∨
      (g t = 0rr) ∨ (g t = 0af) ∨
      (g t = 0rf) ∨ (g t = 0ff))
```

Using the two validity conditions, we can show that the latch structural specification implies its behavioral specification. Because the latch behavioral specification is defined in terms of boolean values the signal functions must be composed with the STATE abstraction function, STATE_ABS, in order to obtain boolean parameters for the specification predicate. The goal to be proven is represented by the following circuit correctness statement:

```
  ((Is_bool_active (d) ∧
    Is_bool (g) ∧
     LATCH_IMP (qn,g,d))   ⇒
     LATCH_SPEC (STATES_ABS o qn,
                 STATES_ABS o g,
                 STATES_ABS o d))
```

The Proof One problem preventing the acceptance of verification into VLSI design methodology is that proofs can be extremely tedious. One solution to this problem is to use different theorem provers for different levels in the design hierarchy. For example, we do *not* advocate the use of HOL for proofs of latches such as the one in this example. As we stated before, its size is convenient for expository purposes. We and others [17] are conducting research aimed at allowing more automatic theorem provers to used at this level in the abstraction hierarchy.

That is not to say, however, that we think that HOL and theorem provers like it have no place in the verification of computer systems. The more abstract the specification, the more likely one is to use a theorem prover like HOL. Even at these levels, multi–level signals are possible. We have, therefore, also begun researching tactics for in HOL that hide some of the intricacies of higher-order logic and allow circuit verification to proceed in a fashion that is more natural for design engineers. Using these tactics, the steps required to prove the goal are similar to the steps that an engineer would go through to analyze the circuit. The proof steps involve unfolding definitions, case splits, and symbolic execution. The verification of out latch example in raw HOL ran to 40 steps. With our new tactics, the proof has been reduced to 15 steps. Most of the work required to verify the circuit is producing the desired case splits. These case splits follow the same lines of reasoning that a circuit designer would take to analyze this latch circuit. HOL provides a convenient way of keeping track of the proof and ensuring that no steps are missed.

The complete description of these compound tactics can be found in [10]. We are currently working on tactics which produce verifications of simple circuits completely automatically [1]. The source code of the complete latch verification session and step by step descriptions of both the current goal and assumption list for the proof is documented in [10].

9 Trying to Verify an Incorrect Circuit

Assigning a single logic state, X, to represent both *invalid* and *valid but not known* signal values can cause difficulties when debugging incorrect circuits using simulation. For example, the NOVA simulator assigns the `STATE Xaa` to both of these cases. In this section we will demonstrate, using an incorrectly designed latch circuit, how the output of the simulator and the verification system compare for incorrect designs.

A BOLT description that incorrectly lets the strength of the feedback inverter, `Inv2`, default to active instead of specifying a resistive strength is described in BOLT as follows:

```
let LATCH_IMP =
BEGIN_BOLT
  MODULE qn .LATCH g d;
    BEGIN
      n1  .NTRAN g d;
      qn  .INVR  n1;
      n1  .INVR  qn;
    END;
END_BOLT
```

This description is the same as the one given earlier except that the specification of the strength has been left out for the second inverter, a common mistake.

The translation of the above BOLT description into HOL is shown below:

```
                q                   q
            g d n               g d n
            * * *               * * *
      00001>0 0 X         00001>0 0 X
      00002>0 0 X         00002>0 0 X
      00003>0 1 X         00003>0 1 X
      00004>0 0 X         00004>0 0 X
      00005>1 0 1         00005>1 0 X
      00006>1 1 0         00006>1 1 X
      00007>1 0 1         00007>1 0 X
      00008>1 1 0         00008>1 1 X
      00009>0 1 0         00009>0 1 X
      00010>0 0 0         00010>0 0 X
      00011>0 1 0         00011>0 1 X
      00012>0 0 0         00012>0 0 X
      00013>1 0 1         00013>1 0 X
      00014>1 1 0         00014>1 1 X
      00015>1 0 1         00015>1 0 X
      00016>1 0 1         00016>1 0 X
      00017>0 0 1         00017>0 0 X
      00018>0 1 1         00018>0 1 X
      00019>0 0 1         00019>0 0 X
      00020>0 0 1         00020>0 0 X
```

Table 2: Correct vs Incorrect Latch Simulation Data

```
LATCH_IMP =
⊢def LATCH (qn,g,d) =
   (∃ n1 n1′ n1″.
      NTRAN (n1′,g,d) ∧
      INVR 1aa 0aa Xaa (qn,n1) ∧
      INVR 1aa 0aa Xaa (n1″,qn) ∧
      JOIN (n1,n1′,n1″))
```

The NOVA simulation output data for this incorrect circuit is compared to the simulation data for a correctly designed circuit in Table 2. The `X` states indicated for the `qn` output during the first four time instances of the correct latch simulation data are the result of the value initially stored in the latch being assumed to be valid but unknown. The value of `qn`, for the correct latch, changes to `1` as soon as `g` first goes true during the fifth time instance. In the case of the incorrect latch, the `qn` output remains `X` during the fifth, and *all* following time instances; the result of the active strength feedback inverter not being overdriven by the value of `d` being passed by the transistor.

In a larger circuit containing such an incorrectly specified latch, the unchanging stream of

Xs in the simulation data would leave few clues as to whether the problem were caused by something such as an incorrect strength device or by some other problem. Attempting to verify the incorrect latch circuit using the proof shown in Section 8.6 results in the message:

```
evaluation failed     REWRITING DID NOT RESOLVE GOAL
```

This message indicates failure of tactic. Examination of the assumption list (part of the proof state printed out as the proof proceeds) reveals a non-equational assumption.

```
[ "(((join laa Oaa = Oaa) ∨
     (join laa Oaa = laa) ∨
     (join laa Oaa = Xaa) ∨
     (join laa Oaa = Xar) ∨
     (join laa Oaa = Xra)) →
    (n1 t = join laa Oaa) |
    (n1(t + 1) = join laa Oaa))" ]
```

During the verification of the correct circuit the assumption resulting from this step is:

```
[ "n1 t = laa" ]
```

The tactics have been written in such a way that they only results in an equational assumption when the strength of one of the arguments to the `join` function is such that it can overdrive the other input signal. In the case of the incorrect circuit, both inputs to the `join` function are active. Because this may cause the `join` to introduce a new `unknown` signal value, the tactic has been written so that it does not add an equational assumption to the assumption list, instead the new assumption remains a conditional. This incompletely resolved assumption gives a clear indication that the fault is due to an incorrectly specified strength for one of the devices driving node `n1`.

10 Conclusion

10.1 Future Work

The theory of signal lattices and BOLT to HOL translator represent an important step in integrating formal verification with CAD tool environments. Future steps include:

1. Expanding and validating the library of HOL definitions corresponding to the primitive components in the BOLT library.

2. Continuing to examine circuit verification proof styles in order to develop even more efficient and natural compound tactics.

3. Developing an abstract syntax and denotational semantics for the circuit structure level of HDLs.

4. Integrating the circuit structure level translation work with the results of on-going research aimed at HDL behavioral model levels to create a complete link between HDL's and verification logics.

10.2 Summary

The goal of our work is to improve techniques for ensuring that VLSI devices are functionally correct by making verification using theorem-proving accessible to design engineers. Before verification is accepted by design engineers, stand-alone verification tools used in the academic research arena must be integrated with the CAD tools used by VLSI designers. The HDLs used by the CAD tools are not appropriate for verification work because they lack clear semantic definitions. On the other hand, logic descriptions lack some of the features needed by the CAD tools. Automatic translation from HDL to verification logic solves this dilemma. Our work has focused on the BOLT HDL, but is applicable to other hardware description languages as well. We have presented a representation for the VHDL multi–signal data type.

In this paper, we have described a method for formal reasoning about the signal values required for the accurate simulation of transistor logic and tri-state bus circuits. This is important because it provides for the interface between verification through theorem-proving and lower level methods to possibly extend below the point where all signals are represented strictly as booleans. We have described a translator for moving circuit structure descriptions from the realm of CAD design tools to formal verification.

The use of formal verification as part of a circuit design methodology faces another problem—formal verification is always tedious and sometimes difficult. We have begun to address this problem by writing several compound tactics that hide some of the intricacies of manipulating the theorem prover and allow circuit verification proof to proceed in a fashion that is more natural for design engineers. Use of these tactics reduces the proof of the example latch circuit from over 40 steps down to 1. The names of these new tactics have been selected to be more meaningful to electronic circuit designers. We have also shown that the verification process can be helpful in locating functional design faults that may be difficult to find using a simulator.

10.3 Acknowledgments

This work was sponsored by the National Science Foundation under NSF grant MIP–9109618, the Department of Defense under contract MDA904–91-C–7054, and NASA under Space Engineering Research Center grant NAGW-1406.

References

[1] M. AAgard, M. Leeser, and P. J. Windley. "Towards a Superduper Hardware Tactic". In J. J. Joyce and C. Seger, editors, *The Proceedings of the International Tutorial and Workshop on the HOL Theorem Proving System and its Applications*, August 1993. Also Technical Report LAL-91-08, University of Idaho Laboratory for Applied Logic, October 1991.

[2] AMI: A Subsidiary of Gould Inc. *BOLT Users Manual.*

[3] G. Birkhoff. *Lattice Theory.* American Mathematical Society, 1948.

[4] R. Boulton, M. Gordon, J. Herbert, and J. van Tassel. "The HOL Verification of ELLA Designs". In *1991 International Workshop on Formal Verification in VLSI Design*, Miami, January 1991.

[5] R. E. Bryant. "A Switch-Level Model and Simulator for MOS Digital Systems". *IEEE Transactions on Computers*, Vol. C-33:160–177, February 1984.

[6] K. B. Cameron and J. C. Shovic. "Calculating Minimum Logic State Requirements for Multi-Strength Multi-Value MOS Logic Simulators". In *1987 IEEE International Conference on Computer Design: VLSI in Computers & Processors*, pages 672–675, Rye Brook, New York, October 1987. IEEE Computer Society Press.

[7] A. Camilleri, M. Gordon, and T. Melham. "Hardware Verification Using Higher Order Logic". In D. Borrione, editor, *From HDL Descriptions to Guaranteed Correct Circuit Designs*, pages 43–67. Elsevier Scientific Publishers (North-Holland), 1987. Also Technical Report No. 91, University of Cambridge Computer Laboratory, September, 1986.

[8] D. R. Coelho. *The VHDL Handbook.* Kluwer Academic Publishers, Boston, 1989.

[9] I. S. Dhingra. "Formal Validation of An Integrated Circuit Design Style". In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 293–321. Kluwer Academic Publishers, Boston, 1988. Also Technical Report No. 115, University of Cambridge Computer Laboratory, August, 1987.

[10] J. W. Gambles. Incorporating formal verification into VLSI design methodology. Master's thesis, University of Idaho, May 1992.

[11] M. J. C. Gordon. "Why Higher Order Logic is a Good Formalism for Specifying and Verifying Hardware". In G. J. Milne and P. A. Subrahmanyam, editors, *Formal Aspects of VLSI Design*, pages 153–177. Elsevier Scientific Publishers (North-Holland), 1986. Also Technical Report No. 77, University of Cambridge Computer Laboratory, 1985.

[12] M. J. C. Gordon. "HOL: A Proof Generating System for Higher-Order Logic". In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 73–128. Kluwer Academic Publishers, Boston, 1988. Also Technical Report No. 103, University of Cambridge Computer Laboratory, August, 1987.

[13] J. P. Hayes. "A Unified Switching Theory with Applications to VLSI Design". *Proceedings of the IEEE*, Vol. 70(No. 10):1140–1151, October 1982.

[14] T. F. Melham. "Abstraction Mechanisms for Hardware Verification". In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 267–291. Kluwer Academic Publishers, Boston, 1988. Also Technical Report No. 106, University of Cambridge Computer Laboratory, May, 1987.

[15] T. F. Melham. "Using Recursive Types to Reason About Hardware Verification". In G. Milne, editor, *Design for Behavioural Verification*, Glosgow, July 1988. IFIP WG 10.2. Also Technical Report No. 135, University of Cambridge Computer Laboratory, May, 1988.

[16] L. Miles, P. Prins, and K. Cameron. "NOVA: A New Multi-Level Logic Simulator". In *2nd NASA Symposium on VLSI Design*, pages 4.1.1–4.1.13, Moscow, Idaho, October 1990. NASA Space Engineering Research Center, University of Idaho.

[17] C.-J. Seger and J. J.Joyce. A two-level formal verification methodology using hol and cosmos. Technical Report 91-10, University of British Columbia Department of Computer Science, June 1991.

[18] J. D. Ullman. *Computational Aspects of VLSI.* Computer Science Press, 1984.

[19] J. P. van Tassel. *The HOL Parser Library.* University of Cambridge Computer Laboratory, July 1991.

[20] J. P. van Tassel and D. Hemmendinger. "Toward Formal Verification of VHDL Specifications". In L. Claesen, editor, *Applied Formal Methods For Correct VLSI Design*, pages 261–270, Houthalen, Belgium, November 1989. IMEC-IFIP WG 10.2/WG 10.5, Elsevier Scientific Publishers (North-Holland).

[21] R. Waxman, L. Saunders, and H. Carter. "VHDL links design, test, and maintenance". *IEEE Spectrum*, Vol. 26(No. 5):40–44, May 1989.

[22] G. Winskel. "A Compositional Model of MOS Circuits". In G. Birtwistle and P. A. Subrahmanyam, editors, *VLSI Specification, Verification and Synthesis*, pages 323–347. Kluwer Academic Publishers, Boston, 1987. Also Technical Report No. 105, University of Cambridge Computer Laboratory, 1987.

Contact Author

Until May 31, 1993:

Phillip J. Windley
Laboratory for Applied Logic
Department of Computer Science
University of Idaho
Moscow ID 83843

208.885.6501 (voice)
208.885.6645 (fax)

After May 31, 1993:

Phillip J. Windley
Laboratory for Applied Logic
Department of Computer Science, 3361 TMCB
Brigham Young University
Provo UT 84602

801.378.3027 (voice)
801.378.7775 (fax)

Affiliation of Authors

Jody W. Gambles

Microelectronics Research Center, Department of Electrical and Computer Engineering, University of New Mexico, Albuquerque NM 87131

Phillip J. Windley

Laboratory for Applied Logic, Department of Computer Science, Brigham Young University, Provo UT 84602

The work reported in this paper was performed while both authors were at the University of Idaho.

Notes

1. The lattice join function is the least upper bound of elements.

2. $N = 1$ is a trivial case of one single STATE, `Nil`.

3. That is, it is proven that the data type elements and covers do truly form an upper lattice.

4. It may be useful while a design is being debugged for a simulator to be able to distinguish uninitialized nodes from nodes that have an invalid state due to device conflicts. However, combining the cases of invalid logic level and valid but not known simplifies the simulation algorithm. A rigorous modeling of the effects of unknown but valid states would require determining Boolean satisfiability, an NP-complete problem [5].

5. While the structure of a circuit clearly distinguishes between a node that is connected and one that is disconnected, it may be considered to be more mathematically rigorous to include all possible strength ranges.

6. The validity conditions and structural description rewritten with the device primitive definitions.

Keywords: circuit verification, signal values, tri-state busses, Standard Logic Package, HDL translation, HOL