AUTOMATED RTL VERIFICATION BASED ON PREDICATE CALCULUS

M. Langevin
Département d'IRO, Université de Montréal, C.P. 6128, Succ. A
Montréal, CANADA, H3C 3J7
e-mail: langevin@iro.umontreal.ca

Abstract

This paper presents a technique for formally verifying synchronous circuits modelled at the Register Transfer Level of abstraction. The circuit's behavior, specification, and hypotheses on its clock sequencing are modelled using a special kind of predicate calculus formulas, named transfer formulas. The proof process consists in applying two general rules of the predicate calculus in a specific order. The automated verification process uses an acyclic graph for representing each transfer formula; in this way, the application of the rules is similar to the manipulation of boolean functions represented by acyclic graphs.

1 Introduction

An error-free design can be assured using good test vectors for simulation, formal verification tools, or silicon compilation. Automated synthesis is the easiest way to design a circuit; unfortunately, no existing tool can handle a wide range of circuit designs. For example, the VTI CAD system can be used to generate the layout of specific components like ROMs, RAMs, PLAs, State Machine, and DataPath, from a high level description [16]; but the compilation of a complete synchronous circuit (datapath and control parts) is not yet available. The validation of these circuits has to be performed with simulation or formal verification tools. Since the complete simulation of a large circuit may be impossible, the designer needs other tools to ascertain the validity of his design; formal verification has been introduced to fill this gap. Much progress has been made in formal verification until now [7]; for example, BULL's PRIAM system has proven the functional correctness of circuits of up to 20000 transistors [11], while the formal verification of a high level description of the RSRE's VIPER processor has been carried out with the HOL system [8].

This paper presents a formal model to prove that the specification of a synchronous circuit is realized by the interconnexion of its components; the circuit specification and structural description are model at the RT Level, but memory component can be latches (asynchronous parts), i.e. the circuit description is more detailed than the circuits with registers only for memory components. Some formalisms have been proposed to perform automated RTL verification [4,15], but these tools do not handle the presence of latches in the design. The systems HOL [6] and VERIFY [3], are accurate formalisms to model the specification and component behavior of the proposed kind of circuit; however, the proofs of correctness in these systems have to be completely or partially human directed, because of the high flexibility of their underlying formalisms. An important feature of a formal verification system is its automation because the circuit designer is not interest to manage the proof; this goal is also pursued in [17,18], where the Boyer-Moore theorem prover is used to verify synchronous design.

The proposed method is based on a subset of predicate calculus for modelling circuit behavior. The proof of correctness of a circuit consists in always applying certain logical rules in a specific order, which means that the process can be automated. Direct acyclic graphs (DAGs) are used to represent logical formulas since the manipulation of these formulas is reduces to the graph-based manipulation of boolean functions [5].

2 Level of modelling

The proposed formalism is intended for modelling the behavior of synchronous circuit driven with a multiphase clock, where memory elements could be latches or registers [1]. Carriers are used to represent the nodes of the circuit [2]: there are clock carriers, input carriers, output carriers, and internal carriers for a particular circuit. A carrier represents a bit vector with a certain length.

The circuit behavior is modelled at Register Transfer Level [2]; at this level, the combinational parts are instantaneous. The time granularity used to model the behavior of the circuit components corresponds to the clock phases; a time unit thus corresponds to the interval between the middle point of two adjacent active clock phases. For example, Figure 1 presents the meaning of the time for a circuit which operates in a 3-phase clock mode. This time representation is called the phase-time representation. The clock frequency and sequencing is assumed to assure correct timing.

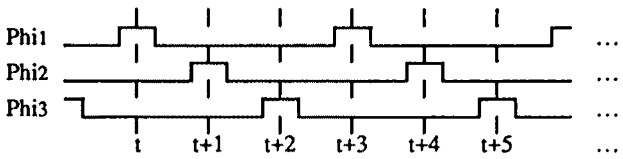

Figure 1

The specification of each circuit's component is a set of transfer functions; a transfer function specifies the possible value that can take a component's output port (an internal or output carrier of the circuit). The components are supposed to be already verified (bottom-up verification), or they will be verified later (top-down verification). Their verification could be performed with other tools of the CAD system: simulators or formal verification tools; in the ideal case, a silicon compiler could be used to generate the components' layout from their specification. A transfer function is defined for any time t, i.e. for any clock phase. A carrier can have memory or not [5]. If a carrier is memoryless, its value at time t+1 depends only on the value of other carriers at time t+1; whereas the value of a memory carrier at time t+1 also depends on the value of carriers at time t (at the previous clock phase).

The specification of a circuit consists in providing transfer functions that specified the possible values for each output carrier and for certain internal carriers; these carriers are the observable carriers of the circuit. The time representation used for these transfer functions corresponds to the clock cycle; this is the cycle-time representation. The circuit's specification is a structural and a temporal abstraction of its behavior [13]. In the specification, the value of an observable carrier V at the end of cycle t+1 (at the last phase of this cycle) must depend only on the values of the input carriers during the cycle and the values of the observable carriers at the end of the previous cycle (the cycle t). These transfer functions must be defined for any clock cycle t. To be able to perform the verification of the circuit, the correspondence between the cycle-time representation and the phase-time representation has to be established. Consider the case of a d-phase clock: the nth phase of cycle $t + 1$ ($1 \leq n \leq d$) occurs at phase-time $d.t + n$, i.e. the end of each cycle occurs at a phase-time which is a multiple of d. Hypotheses on the value of clock carriers have to be made when the circuit behavior is specified. A circuit description is the set of its transfer functions which represent the circuit's behavior, the specification, and the hypotheses on the clock sequencing.

3 An example

Let the behavior of a k-bit latch with input D, output Q, and control line 'load' be modelled. The carriers D and Q are described by time function returning bit vectors of length k, while the 'load' carrier is described by a function returning a one bit value. The latch behavior can be modelled with the following first-order formula:

$$\mathrm{load}(t+1) * \mathrm{EQU}(Q(t+1),D(t+1)) + \overline{\mathrm{load}(t+1)} * \mathrm{EQU}(Q(t+1),Q(t))$$

where '*' stands for logical 'and', '+' for logical 'or', 'EQU' is the equality predicate, and the variable t represents a clock phase (t is considered to be universally quantified). This formula, called transfer formula, specifies the possible value that can take Q at any phase t+1; if load is true at this phase then the latch is transparent, otherwise the latch keeps its last value.

Two of this latches and one multiplexor are interconnected to form the circuit of Figure 2; after a clock cycle, the value of B is a new value iff 'we' (Write Enable) is high during the second phase. The formulas H1, H2, H3 and H4 constrain the possible values for the clock carriers while the circuit specification is the formula S. To prove that the structure of the circuit meets its specification, it must be shown that the formula S is a logical consequence of the formulas H1, H2, H3, H4, and the formulas F1, F2 and F3, extracted from the circuit structure; F1 and F2 are latch specification while F3 is the multiplexor specification.

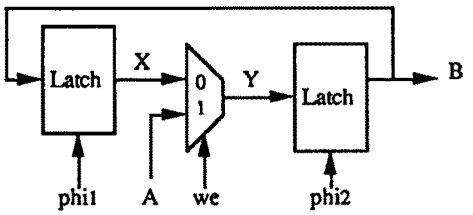

Figure 2

$$\mathrm{we}(2t+2) * \mathrm{EQU}(B(2t+2),A(2t+2)) + \overline{\mathrm{we}(2t+2)} * \mathrm{EQU}(B(2t+2),B(2t)) \qquad (S)$$

$$\mathrm{EQU}(\mathrm{phi}_1(2t+1),T) \qquad (H_1)$$

$$\mathrm{EQU}(\mathrm{phi}_2(2t+1),F) \qquad (H_2)$$

$$\mathrm{EQU}(\mathrm{phi}_1(2t+2),F) \qquad (H_3)$$

$$\mathrm{EQU}(\mathrm{phi}_2(2t+2),T) \qquad (H_4)$$

$$\mathrm{phi}_1(t+1) * \mathrm{EQU}(X(t+1),B(t+1)) + \overline{\mathrm{phi}_1(t+1)} * \mathrm{EQU}(X(t+1),X(t)) \qquad (F1)$$

$$\mathrm{phi}_2(t+1) * \mathrm{EQU}(B(t+1),Y(t+1)) + \overline{\mathrm{phi}_2(t+1)} * \mathrm{EQU}(B(t+1),B(t)) \qquad (F2)$$

we(t) * EQU(Y(t),A(t)) + $\overline{\text{we(t)}}$ * EQU(Y(t),X(t)) (F3)

The following proof demonstrates that the circuit's behavior meets its specification. It is straithforward for this small example, but could be tedious for larger circuits (note that this proof holds for any width of the datapath). An algorithm to perform automatically this kind of proof is proposed in this paper.

1.	phi_2(2t+2) * EQU(B(2t+2),Y(2t+2)) + $\overline{\text{phi}_2\text{(2t+2)}}$ * EQU(B(2t+2),B(2t+1))	F2,$[t \mid 2t+1]$)
2.	EQU(B(2t+2),Y(2t+2))	(1,H4)
3.	we(2t+2) * EQU(Y(2t+2),A(2t+2)) + $\overline{\text{we(2t+2)}}$ * EQU(Y(2t+2),X(2t+2))	(F3,$[t \mid 2t+2]$)
4.	we(2t+2) * EQU(B(2t+2),A(2t+2)) + $\overline{\text{we(2t+2)}}$ * EQU(B(2t+2),X(2t+2))	(2,3)
5.	phi_1(2t+2) * EQU(X(2t+2),B(2t+2)) + $\overline{\text{phi}_1\text{(2t+2)}}$ * EQU(X(2t+2),X(2t+1))	(F1,$[t \mid 2t+1]$)
6.	EQU(X(2t+2),X(2t+1))	(5,H3)
7.	we(2t+2) * EQU(B(2t+2),A(2t+2)) + $\overline{\text{we(2t+2)}}$ * EQU(B(2t+2),X(2t+1))	(4,6)
8.	phi_1(2t+1) * EQU(X(2t+1),B(2t+1)) + $\overline{\text{phi}_1\text{(2t+1)}}$ * EQU(X(2t+1),X(2t))	(F1,$[t \mid 2t$)
9.	EQU(X(2t+1),B(2t+1))	(8,H1)
10.	we(2t+2) * EQU(B(2t+2),A(2t+2)) + $\overline{\text{we(2t+2)}}$ * EQU(B(2t+2),B(2t+1))	(7,9)
11.	phi_2(2t+1) * EQU(B(2t+1),Y(2t+1)) + $\overline{\text{phi}_2\text{(2t+1)}}$ * EQU(B(2t+1),B(2t))	(F2,$[t \mid 2t]$)
12.	EQU(B(2t+1),B(2t))	(11,H2)
13.	we(2t+2) * EQU(B(2t+2),A(2t+2)) + $\overline{\text{we(2t+2)}}$ * EQU(B(2t+2),B(2t))	(10,12)

4 Transfer formulas

In predicate calculus it is possible to use functions that return bit vectors of fixed length. The value taken by a carrier of the circuit at a given time can be modelled as a time function that returns bit vectors. This kind of function is called a signal, as in [10].

Functions can be used also to model high level operators similar to the ones used in RT level HDL's [2]. An operator is modelled as a function that returns a bit vector, and takes as parameters a (possibly empty) list of bit vectors. The operators have only a syntactic meaning, like in [15], i.e. the function of an operator used in the circuit specification has to be realized by a component of the circuit. Hierarchical verification is favored because the functionality of the operator is verified when the corresponding component is verified.

A signal or an operator that returns a bit vector of length 1 can be considered a predicate; it is true if the returned bit is high, and false if the returned bit is low. Special kind of formulas, called transfer formulas, are used to model the transfer functions. The syntax used for the description of a given circuit is similar to the predicate calculus syntax, and corresponds to a subset of first-order predicate calculus (with equality) [12]. The syntax uses temporal expression with the form (c.t+n) to represent a phase-time; the constant c and n are non-negative integers while the variable t can represent a clock phase or a clock cycle. Also, the terms have the type Quad_n defined as $\{0,1,\Phi,\Delta\}^n$; this type represents a vector of n values defined as 0, 1, undefined or high impedance. Here is the syntax:

A) Signals:
A function V: (time) $\rightarrow$ Quad_n, is a signal. Each carrier is represented by a signal.

B) Operators:

1) The function M^k: $(Quad_{n1} \times \ldots \times Quad_{nk}) \rightarrow Quad_n$, is associated with each operator M of the circuit,
2) Special operators:
   i) The constant T: $() \rightarrow Quad_1$ models the truth value "true", and the bit high;
   ii) The constant F: $() \rightarrow Quad_1$ models the truth value "false", and the bit low;
   iii) The constant UD_n: $() \rightarrow Quad_n$ models a vector of n undefined values;
   iv) The constant HZ_n: $() \rightarrow Quad_n$ models a vector of n high impedance values;
   v) The operator BUS_n: $(Quad_n \times Quad_n) \rightarrow Quad_n$ models the connection of two sources (defined like the primitive JOIN in [3]).

C) Terms are defined recursively as:
   1) If I is a temporal expression and S is a signal then S(I) is a term,
   2) If $V_1,\ldots,V_n$ are terms and M^n an operator, then $M^n(V_1,\ldots,V_n)$ is a term (the type of V_i has to be the same as the specified type of the ith parameter).

D) Predicates:
   A term P of type $Quad_1$ is considered a predicate, except for HZ_1 and UD_1.

E) Conditions (propositional formulas):
   1) A predicate is a condition,
   2) If G and H are conditions then so are
      i) $\overline{G}$ (logical not)
      ii) $G * H$ (logical and)
      iii) $G + H$ (logical or).

F) Transfers formulas (TFs) for the term U are defined recursively as:
   1) If V is a term then EQU(U,V) is a TF for the term U (this special predicate is called atomic TF (ATF)). It means that the term U takes on the value of the term V; U is the destination while V is the source (the types of U and V have to be the same).
   2) If P is a condition, and F_1 and F_2 are TF for the term U, then $P * F_1 + \overline{P} * F_2$ (IF P THEN F_1 ELSE F_2) is a TF for the term U.

In the following, we will assume that the type restrictions are respected in all TFs. The only variable that could appear in a TF is the time variable t; it is considered to be universally quantified. Therefore, a TF for the term U defines the values that U can take for all t. This is the general form of the TF:

$$cond_1 * EQU(U,U_1) + cond_2 * EQU(U,U_2) + \ldots + cond_n * EQU(U,U_n)$$

where the $cond_i$ are composed from a set $P_1,\ldots,P_m$ of predicates. This formula specifies that term U takes the value of the term U_i if $cond_i$ is satisfied; U can take one and only one value because all the $cond_i$ are mutually disjoint and their union is a tautology. These formulas can be represented with direct acyclic graphs (DAGs) if the predicates are matched to the nodes of the graph and the atomic TFs EQ(X,Y) are matched to the leaves (Figure 3). The difference of TFs with boolean functions is that more than two values can be transferred in the TFs. A boolean TF is a TF where each Ui is T or F; its representation is the same as for a boolean function. The reduction algorithm for the DAGs that represent boolean functions proposed in [5] can be applied in the same way as for the DAGs that represent TFs. Moreover, as in the case of boolean functions, there exists a canonical form for these DAGs.

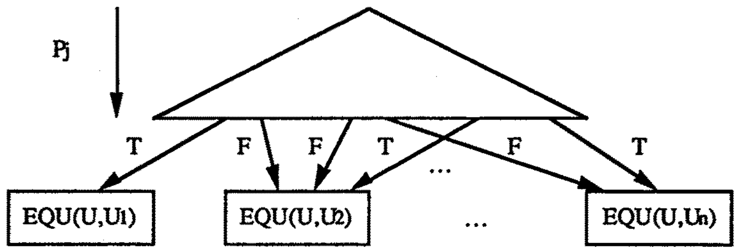

Figure 3

For a given circuit operating in d-phase mode, the description is modelled with specific TFs. For the specification, a TF is defined for each observable carrier at time dt+d (the end of the cycle); its value depends on the value of input carriers at a time between dt+d and dt, and on the value of the observable carriers at the time dt. For the hypotheses on the clock sequencing, a TF is defined for each clock carrier at time dt+n, $1 \leq n \leq d$; its value is either true or false depending on whether or not the clock carrier is active at the nth phase. For the behavior, a TF is defined for each internal and output carrier; if the carrier is a memory carrier, its value at time t+1 depends on the value of the other carriers at time t+1 and t, otherwise, its value at time t depends only on the value of the other carriers at time t.

Example of a circuit description: The circuit NextPC (Figure 4), operating with a 2-phase clock, computes the future value of the PC. At the end of a cycle if the flags' value satisfies the jump condition (JC) then the PC takes on the value of the jump address (JA); otherwise the last value of the PC (the one at the end of the previous cycle) is incremented by 1. The increment operation is modelled with the operator INC while the test condition is modelled with the operator SATISFY; it is assumed that these operators have a verified implementation.

The specification of NextPC is modelled with the transfer formula S while the formulas F1, F2, F3, F4 and F5, extracted from the circuit component, modelled the circuit behavior. The hypothesis on the clock sequencing are the same as for the circuit of Figure 2.

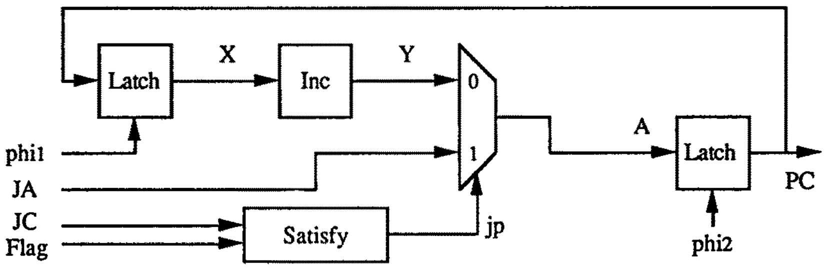

Figure 4

$$SATISFY(JC(2t+2),Flag(2t+2)) * EQU(PC(2t+2),JA(2t+2)) + \overline{SATISFY(JC(2t+2),Flag(2t+2))} * EQU(PC(2t+2),INC(PC(2t))) \quad (S)$$

$phi_2(t+1) * EQU(PC(t+1),A(t+1)) + \overline{phi_2(t+1)} * EQU(PC(t+1),PC(t))$ (F_1)

$phi_1(t+1) * EQU(X(t+1),PC(t+1)) + \overline{phi_1(t+1)} * EQU(X(t+1),X(t))$ (F_2)

$jp(t) * EQU(A(t),JA(t)) + \overline{jp(t)} * EQU(A(t),Y(t))$ (F_3)

$EQU(jp(t),SATISFY(JC(t),Flag(t)))$ (F_4)

$EQU(Y(t),INC(X(t)))$ (F_5)

5 Behavioral verification

With the proposed formalism, the proof of correctness of a circuit consists in applying some rules of predicate calculus with the goal of deriving the transfer formulas of the specification from the transfer formulas of the circuit behavior. Let $F_1,\dots,F_n$ be the TFs of the circuit behavior, $H_1,\dots,H_m$ be the hypotheses on clock sequencing, and $S_1,\dots,S_p$ be the TFs of the circuit specification. The goal is to prove that each of the formulas S_i is a theorem when the formulas $F_1,\dots,F_n,H_1,\dots,H_m$ are taken as assumptions. A transfer formula S_i specifies the values that an observable carrier V at time dt+d can take as a function of the values of the input carrier between the times dt and dt+d, and the values of the observable carriers at the time dt (the constant d is the number of phases). This kind of TF is called an observable TF, and is defined as:

Def 1: A term U is observable iff one of the following conditions is true:
- i) U has the form S(I) where S is an input signal and I a temporal expression,
- ii) U has the form S(dt) where S is an observable signal,
- iii) U has the form $M(A_1,\dots,A_n)$ where M is an n-ary operator and each A_i is observable.

Def 2: A condition C is observable iff each predicate that occurs in C is observable.

Def 3: An atomic transfer formula EQU(U,V) is observable iff the term V is observable.

Def 4: A TF is observable iff each of its conditions and atomic TFs is observable.

An observable TF for a term U specifies the values that U can take as a function of only the values of the observable carrier at time dt and the values of the input carriers at some arbitrary time; the TF S_i is an observable TF for the term V(dt+d). Another TF for V(dt+d) can be extracted from the circuit's behavior since the variable t of the TF for the carrier V can be replaced by any temporal expression. Now, suppose that this TF is transformed in an observable form using logical rules; if this observable formula is identical with S_i then S_i is deduced from the circuit behavior. This is the verification technique proposed in this paper. Two specific rules are used to transform a TF in an observable form:

1) The substitution rule:

Suppose that we have the two following TFs:

$cond_1 * EQU(U,U_1) + cond_2 * EQU(U,U_2) + \dots + cond_n * EQU(U,U_n)$ (1.1)

$G * EQU(P,T) + \overline{G} * EQU(P,F)$ (1.2)

The substitution of 1.2 in 1.1 consists in replacing all the occurrences of P in the conditions of the TF 1.1 by the condition G of the TF 1.2.

2) The transition rule:

Suppose that we have the two following TFs:

$C_{1,1} * EQU(U,U_1) + ... + C_{1,n} * EQU(U,U_n)$ (2.1)

$C_{2,1} * EQU(V,V_1) + ... + C_{2,m} * EQU(V,V_m)$ (2.2)

The transition of 2.1 with 2.2 consists in replacing all the occurrences of V in the terms U_i of 2.1 by its corresponding value specified in 2.2. First, the logical 'and' is applied between the two TFs and the following formula is obtained:

$$C_{2,1} * C_{1,1} * EQU(U,U_1) * EQU(V,V_1) + ... + C_{2,1} * C_{1,n} * EQU(U,U_n) * EQU(V,V_1)$$
$$\cdots$$
$$C_{2,m} * C_{1,1} * EQU(U,U_1) * EQU(V,V_m) + ... + C_{2,m} * C_{1,n} * EQU(U,U_n) * EQU(V,V_m)$$

This transitive rule for equivalence is applied to each product of atomic TFs:

EQU(G,H) * EQU(X,Y) => EQU(G,H') where H' is the term H in which the occurrences of X are replaced by Y.

The algorithm used to deduce the observable TF for V(t') (t' = ct+n) consists in applying these two rules in a specific order; this algorithm is presented in Figure 5. The initial TF is taken from the list of TFs of the circuit behavior (if the signal represents an output or an internal carrier), or from the list of TFs of the hypotheses on the clock sequencing (if the signal represents a clock carrier). Each non-observable predicate P in the conditions of this TF are replaced by its value specified by the observable TF for P; this transformation is performed with the application of the substitution rule. After, each non-observable source term X of the atomic TF is replaced by its value specified by the observable TF for X; this transformation is performed with the application of the transition rule. The resulting formula is the observable TF for V(t').

The application of the two deduction rules is similar to boolean function graph manipulation when the TFs are represented with a DAG. The substitution rule consists in replacing a predicate P of a TF E by the value specified in the TF D for P. Since the TF D is in the boolean form, this rule can be applied in the same way as the composition of boolean function. This algorithm generates a graph by applying the IF...THEN...ELSE (ITE) function between three graphs: the graph of D, the graph of E where P is restricted to the value true, and the graph of E where the value of P is restricted to false. When the algorithm reaches the leaves of these graphs, the function ITE is applied on three leaves: a leaf that represents a truth value for P, a leaf of E that represents an atomic TF when P is true, and a leaf of E that represents an atomic TF when P is false. Like for boolean functions, the result leaf depends only on the value of the leaf of P.

The transition of a TF D for U with a TF E for V can be performed in the similar way as the application of the logical 'and'. When this function is applied between two boolean functions, the logical 'and' is applied only to the pair of leafs' value. To perform the transition rule, instead of applying the logical 'and', the equality rule is applied to the two leaves $EQU(U,U_i)$ and $EQU(V,V_j)$; the occurrences of the term V in the term U_i are replaced by the term V_j. Unlike the application of the logical 'and', the equality rule is applied in a strict order.

Closed loops in the circuit can be detected when the circuit is verified. During the recursive application of the algorithm to deduce the initial TF for U, if the term U is met, a closed loop in the circuit is detected since the value of the term U depends on the value of the term U [1]. With this algorithm, the way to prove that the specification TF for the term V(dt+d) is a theorem consists in deducing the observable TF for the term

V(dt+d) from the implementation and in showing that it is equivalent to the specified TF. This comparison is carried out by comparing the two DAGs as in [5]. Also, the inclusion of formulas can be used to show the correctness of an incomplete specification.

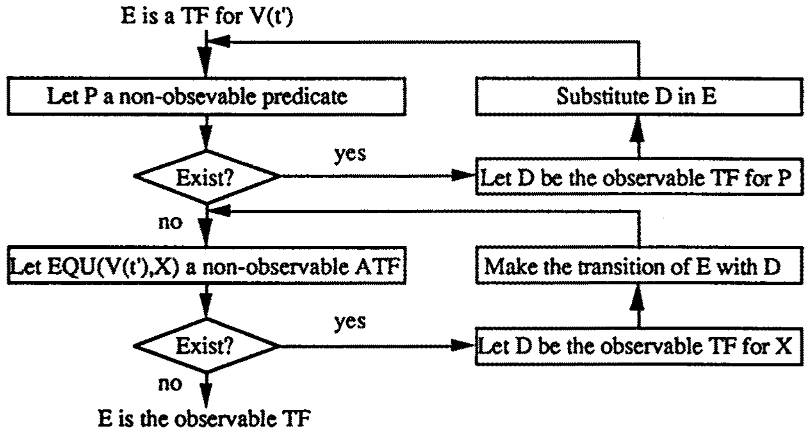
E is a TF for V(t')
Let P a non-obsevable predicate
Exist?
yes
no
Let D be the observable TF for P
Substitute D in E
Let EQU(V(t'),X) a non-observable ATF
Exist?
yes
no
Let D be the observable TF for X
Make the transition of E with D
E is the observable TF

Figure 5

6 Experimental Results

A prototype of the proposed verification technique is under development in Common Lisp. The example of section 4 was successfully verified in about 30 seconds on a Mac Plus. This tools was used in the WORP project [9] at the EPFL, Switzerland; WORP (Watch Oriented RISC Processor) is a RISC processor because it has only one level of programming and each instruction is executed in 4 phases [14]. Its general structure is of the type Harvard, i.e. data and code reside in different memories. It has a 2K of 48-bit instructions saved in a ROM, and 256 of 8-bit words of RAM that can be accessed in a relative or absolute mode.

The circuit is built from the interconnexion of a ROM, a latches bank, an ALU, some working registers, adders, multiplexors, buses, etc. Only a partial verification of this processor has been performed up to now. We specify the possible values that can be stored in the Index Latch of the processor after the execution of any instruction; these values depend on certain fields of the instruction (the ALU code operation, the two source registers for the ALU operation, etc.). A closed loop in the circuit's behavior was detected during this verification. Once this error corrected, the specification has been successfully verified; a trace of roughly 10 pages is produced during the verification and it takes about 7 minutes of real time on a Mac II.

7 Future work

Some improvements to our proposed verification system are planned. First of all, an RT level HDL language could be used to describe the circuit specification, component and hypotheses; the transfer formulas should be deduced directly from the syntax of the HDL (the partial specification of the WORP processor needs 200 lines of edit text and no syntactical analysis is performed). The type validation will be performed directly from the HDL circuit description. Secondly, like the operator BUS, it seems possible to use other predefined operators that have a specific associated meaning. This will be essential in the verification of microcode such as [10]; this requires to associate rewriting rules with the operators, as in [4]. Also to extend the range of verifiable circuits, the pipeline behavior and precharged component will be studied. The study of several circuit examples would help to establish the limitations of our approach.

We try to improve the graph algorithm processing performance by using techniques such as proposed in [11]. Last but not least, the system has to be integrated in a complete CAD system in order to assure the consistency of component models with their real behavior.

Acknowledgements

We want to thank Professor Jacques Zahnd from the EPFL for his help in the formalization of the model, and for his remarks on this research in general. Part of this work was done by the author at the EPFL, Switzerland.

References

[1] F. Anceau, "The Architecture of Microprocessors", Addison-Wesley, 1986.
[2] M. R. Barbacci, "A Comparison of Register Transfer Languages for Describing Computers and Digital Systems", IEEE Trans. on Comp., Vol. C-24, No. 2, February 1975.
[3] H. G. Barrow, "Verify: A Program for Proving Correctness of Digital Hardware Designs", Artificial Intelligence, Vol. 24, 1984.
[4] A. Bartsch, H. Eveking, H.-J. Faerber, M. Kelelatchew, J. Pinder, U. Schellin, "LOVERT -A Logic Verifier of Register-Transfer Level Description", Proceedings of the IMEC IFIP International Workshop on "Applied Formal Methods for Correct VLSI Design", November 1989.
[5] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", IEEE Trans. on Comp., Vol. C-35, No. 8, August 1986.
[6] A. Camilleri, M. Gordon, T. Melham, "Hardware Verification Using Higher Order Logic", in From HDL Descriptions to Guaranteed Correct Circuit Designs, ed. D. Borrione, North-Holland, 1987.
[7] P. Camurati, P. Prinetto, "Formal Verification of Hardware Correctness: Introduction and Survey of Current Research", Computer, July 1988.
[8] A. Cohn, "A Proof of Correctness of the VIPER Microprocessor: the First Level", in VLSI Specification, Verification, and Synthesis, eds. G. Birtwistle and P. A. Subrahmanyam, Kluwer Academic Publishers, 1988.
[9] C. Iseli, "WORP: A Watch Oriented RISC Processor", B. Sc. Thesis, Laboratoire de Systèmes Logiques, EPFL, December 1987.
[10] J. Joyce, G. Birtwistle, M. Gordon, "Proving a Computer Correct in Higher Order Logic", Technical Report No. 100, Computer Laboratory, University of Cambridge, December 1986.
[11] J. C. Madre, F. Anceau, M. Currat, J. P. Billon, "Formal Verification in an Industrial Design Environment", to appear in 'The international Journal on Computer Aided VLSI Design'.
[12] Z. Manna, "Mathematical Theory of Computation", McGraw-Hill, 1974.
[13] T. F. Melham, "Abstraction Mechanisms for Hardware Verification", in VLSI Specification, Verification, and Synthesis, eds. G. Birtwistle and P. A. Subrahmanyam, Kluwer Academic Publishers, 1988.
[14] W. Stallings, "Tutorial: Reduced Instruction Set Computers", IEEE Computer Society Press, Order Number 713, 1986.
[15] R. Vemuri, "A Formal Model for Register Transfer Level Structures and Its Applications in Verification and Synthesis", Proceedings of the IMEC IFIP International Workshop on "Applied Formal Methods for Correct VLSI Design", November 1989.
[16] "VTI User's Guide", VLSI Technology, Inc, 1988.
[17] A. Bronstein, C. L. Talcott, "Formal Verification of Pipelines Based on String-Functional Semantics", Proceedings of the IMEC IFIP International Workshop on "Applied Formal Methods for Correct VLSI Design", November 1989.
[18] L. Pierre, "The Formal Proof of Sequential Circuits Described in CASCADE Using the Boyer-Moore Theorem Prover", Proceedings of the IMEC IFIP International Workshop on "Applied Formal Methods for Correct VLSI Design", November 1989.