Formal Verification of Behavioural VHDL Specifications : a Case Study

Félix NICOLI, Laurence PIERRE
LIM - URA CNRS 1787
Université de Provence
3, Place Victor Hugo
13331 Marseille cedex 3, FRANCE

Abstract

This paper addresses the problem of formally verifying VHDL descriptions. More precisely, we check the correctness of a VHDL *architecture* w.r.t. another *architecture* of the same *entity*. Both of them are translated into recursive functional forms, and the proof of their equivalence is realized by means of the Boyer-Moore theorem prover. Our methodology is illustrated by a significant example that has been proposed by THOMSON. This benchmark presents interesting characteristics and its proof is not obvious.

1 Introduction

Since the correctness of VLSI devices must be efficiently verified before manufacturing, formal verification is becoming an interesting alternative to simulation. In the formal proof approach, the circuit implementation and its expected behaviour (called below its "specification") are expressed using a mathematical model (propositional calculus, first-order or higher-order logic, temporal logic, ...) and appropriate methods are applied to these representations. The concept of formal verification includes various aspects [12], here we focus on verifying that a given realization is functionaly equivalent to its specification. To that goal, we must be certain that the given specification itself is correct.

Our system PREVAIL, that is being developed in cooperation with Imag/ Artemis, is a prototype proof environment which includes a set of verification tools [3]. It takes as input VHDL descriptions [1] and verifies the equivalence (or the implication) of two different *architectures* of the same *entity* (for instance an implementation and its specification, or two descriptions given at different abstraction levels, or a description and its optimization, ...). To that goal, the most appropriate verification tool is selected with the help of the designer, the original descriptions are automatically translated into the formalism of this proof system, which is finally invoked. The VHDL descriptions must be given using a particular language subset and description style, both of them have been chosen such that usual constructs (register transfers, finite state machines, ...) can be given a functionnal semantics

In the case where we have to compare an implementation and its specification, both given as VHDL *architectures* by the designer, we may need to verify that the specification includes no error, with respect to the behaviour that it is supposed to express. Indeed, it happens that the specification is so close to the realization that it is more easy to verify that they are equivalent than to prove that the specification is correct. In this paper, we address this second issue on the basis of a significant case study. More precisely, we verify the correctness of the complex specification of a circuit which inputs a bit vector and outputs the index of the first bit equal to '1', starting from the most significant bit. This benchmark has been proposed by THOMSON in the framework of a JESSI project. Since this proof requires a inductive technique, it will be performed by means of the Boyer-Moore Theorem Prover (Nqthm), which is one of the proof tools included in PREVAIL.

The architecture "recursive" of entity "prem" in Figure 1 below corresponds to the specification that is to be verified. The algorithm expressed by function "*first_rec*" computes the index of the first bit equal to '1' using a dichotomy-based technique. The architecture "iterative" of "prem" realizes the same computation using an iterative function which simply goes through the bit-vector and stops when a '1' is found. The correctness of the architecture "recursive" will be checked with respect to the architecture "iterative".

In other words, the verification process consists in

proving the equivalence of the functions "*first_loop2*" and "*first_rec*". To that goal, we will :

- translate these VHDL descriptions into equivalent recursive Boyer-Moore functions,
- use the induction principle of Nqthm to verify their equivalence.

For the validation of combinational as well as sequential VLSI devices, significant BDD-based methodologies have recently been developed [11], [13], [8] ... However, the major drawbacks of these techniques are :

- the size of the circuit must be fixed to 8, 16, 32 ... bits, and the proof system verifies the correctness of each output independently,
- the "specification" of the circuit must be another implementation of the same arithmetical function, also considered at the boolean level.

Conversely, using the inductive methodology of the Boyer-Moore prover brings several advantages :

- we no longer have to care about the size of the device, we reason on n-bit architectures and we prove generic properties,
- the specification can be given at the "integer" level, and we can use conversion functions between bit-vectors and integers. This feature is very important for the hierarchical verification of large arithmetic devices built from interconnected elementary modules.

2 The Boyer-Moore system

2.1 Overview

We simply give a brief overview of the Boyer-Moore theorem prover. For more details, see [14], [15]. This prover is based on a quantifier-free first-order logic with equality. Its syntax is a Lisp-like syntax, combining functions and variable names in a prefix notation.

In that logic, terms are variables or expressions of the form $(fa_1 \ldots a_n)$ where f is a n-ary function and $a_1 \ldots a_n$ are terms. There are two constants $(true)$ and $(false)$ denoted T and F.

The main principles implemented in that prover are :

```
entity prem is
               generic (n :  Natural; log_N : Natural)
               port (i :  in UNSIGNED (N-1 downto 0);
                     o :  out UNSIGNED (N-1 downto 0));
end prem;

architecture iterative of prem is
---------------------------------

function first_loop2 (X: UNSIGNED) return NATURAL is
         variable retour :  NATURAL range 0 to X'length;
begin
      retour := 0;
      for i in X'range loop
          retour := retour + 1;
          if X (i) = '1' then return retour;
          end if;
      end loop;
      return 0;
end;

begin
      o <= CONV_UNSIGNED (first_loop2 (i), o'length);
end iterative;

architecture recursive of prem is
---------------------------------

function first_rec (X: UNSIGNED) return NATURAL is
         constant medium :  NATURAL range X'low to X'high :=
                            (X'length / 2) + X'low;
         constant half :  NATURAL range X'low to X'high :=
                          X'length - (X'length / 2);
         variable temp1, temp2 :  NATURAL range X'low to X'high;

begin
      if X'length = 1 then
         if X (X'high) = '1' then return 1;
         else return 0;
         end if;
      else
           temp1 := first_rec (X (X'high downto medium));
           temp2 := first_rec (X (medium-1 downto X'low));
           if temp1 /= 0 then return temp1;
           elsif temp2 /= 0 then return temp2 + half;
           else return 0;
           end if;
      end if;
end;

begin
      o <= CONV_UNSIGNED (first_rec (i), o'length);
end recursive;
```

Figure 1: Behavioral specifications of entity "prem"

- *Shell principle* : it allows to define inductive types. They are defined by a *bottom* object, a *constructor function*, one or more *destructor function(s)* and a *type recognizer predicate*.
- *Definition principle* : it allows to introduce new recursive functions in the logic. Prior to accepting a new recursive definition, the system verifies that it can find a *measure* which decreases according to a *well-founded relation*.
- *Induction principle* : a property $\mathcal{P}$ is proved if the proof checker succeeds in rewriting it into T. To that goal, it uses various heuristics, among them the structural induction. Roughly speaking, proving $\mathcal{P}(x)$ consists in showing :
  - $\mathcal{P}(x)$ is true for the base case;
  - if $d_1 \cdots d_n$ are destructor functions for x, $\forall i /$ $d_i(x)$ and x have the same type, $\mathcal{P}(d_i(x)) \Rightarrow$ $\mathcal{P}(x)$.

2.2 Hardware verification and Nqthm

The scope of the Boyer-Moore theorem prover covers various areas. For instance, it has already been involved in the verification of theorems in the field of theoretical computer science or mathematics [10, 18, 4], and in the validation of algorithms [9]. Since 1985, significant results have been obtained in the framework of hardware verification :

- Warren Hunt realized the mechanical proof of the FM8501 microprocessor [17], and more recently he verified the ALU of the FM8502 [16],
- at Stanford University, a "String-Functional Semantics" has been implemented in the Boyer-Moore logic, and devices such as pipelined architectures have been verified [2],
- at the University of Newcastle, this prover has been included in a synthesis environment for DSP devices, [6],
- at IMEC (Belgium), this system has been applied to parameterized architectures described in Hilarics [5].

In all these approaches, the Boyer-Moore code has to be directly written by the designer. To avoid this user-interaction with the prover, our system takes as input VHDL circuit descriptions and translates them automatically in the appropriate formalism.

2.3 The bit-vector shell

To represent bit-vectors in the Boyer-Moore logic, we use the bit-vector shell defined by W.Hunt (see [17]). This shell consists of :

- *(btm)* : the bottom object (empty bit vector);
- *bitv* : the two-arguments constructor function;
- *bit* : the one-argument destructor function returning the first bit;
- *vec* : the one-argument destructor function returning the end of the vector;
- *bitvp* : the recognizer function.

In the following, we will also use bit-vector functions that have been given by W.Hunt : "*all–zerosp*" which checks whether every bit of a bit vector equals F, "*v–append*" which catenates two bit-vectors, "*size*" which returns the length of a bit-vector.

3 Definitions

3.1 VHDL functions translated in Nqthm

The definition of the function "*first_rec*" is recursive. Thus, its translation in Boyer-Moore is rather straightforward. Conversely, the iterative form of "*first_loop*2" must be transformed into a "pseudo-iterative" function, i.e. a recursive function with an accumulator. To that goal, the "for" loop is replaced by recursion and `retour` becomes the accumulator.

Remarks :
- We have seen that the bit-vector shell encodes bit-vectors as lists of booleans. We will consider that the first element of the list represents the most significant bit;
- Bit-vectors are implicitly normalized in our representation, i.e. `x'low = 0` and `x'high = x'length-1` . In particular, this is the reason why `medium` corresponds to `x'length/2` instead of `x'length/2 + x'low` .

3.1.1 first_loop2

As mentioned above, the iterative definition becomes recursive with an accumulating parameter which corresponds to the variable `retour` of the VHDL description. This definition is given in Figure 2.

```
first_loop2(x,retour)
= if (bitvp(x) and x ≠ (btm)) then
      if bit(x) then (retour + 1)
      else first_loop2(vec(x),(retour + 1))
  else 0
```

Figure 2: Definition of $first\_loop2$

3.1.2 first_rec

Here, the Boyer-Moore definition faithfully translates the VHDL description. However, since they depend on x, the constants $half$ and $medium$ are transformed into functions :

```
medium(x)                    half(x)
= quotient(size(x), 2)       = size(x) - medium(x)
```

The dichotomy-based algorithm splits the bit vector x into two parts :
$x[x'high \ \ downto \ \ medium]$ and $x[medium - 1 \ downto \ x'low]$. In order to represent in the Boyer-Moore system the expressions mentioned above, we use two functions that we have defined and stored in a hardware verification-oriented package. These functions are :

- $MSB(x, n)$ which extracts the n most significant bits of the vector x;
- $LSB(x, n)$ which gives the vector x without its n most significant bits.

Their Boyer-Moore definitions are :

```
MSB(x,n)
= if bitvp(x) then
    if (n = 0) then (btm)
    else bitv(bit(x),MSB(vec(x),(n-1)))
  else (btm)

LSB(x,n)
= if bitvp(x) then
    if (n = 0) then x
    else LSB(vec(x),(n-1))
  else (btm)
```

Thus, we have the following correspondences between Boyer-Moore expressions and VHDL terms :

- $MSB(x, size(x) - medium(x)) \equiv$ $x[x'high \ downto \ medium]$;
- $LSB(x, size(x) - medium(x)) \equiv$ $x[medium - 1 \ downto \ x'low]$.

```
first_rec(x)
= if (bitvp(x) and x ≠ (btm)) then
    if (size(x)=1) then
      if bit(x) then 1
      else 0
    else
        if first_rec(MSB(x,size(x)-medium(x)))≠0
        then first_rec(MSB(x,size(x)-medium(x)))
        else
            if first_rec(LSB(x,size(x)-medium(x)))≠0
            then first_rec(LSB(x,size(x)-medium(x)))+half(x)
            else 0
  else 0
```

Figure 3: Definition of $first\_rec$

The Boyer-Moore definition of $first\_rec$, given by Figure 3, uses the expressions mentionned above.

3.2 Function acceptance

In the definition of $first\_loop2$, it is clear that the measure of x decreases, since x becomes $vec(x)$ (i.e. the bit-vector without its most significant bit) in the recursive call. Therefore, this definition is directly accepted by the Boyer-Moore system.

As far as the definition of $first\_rec$ is concerned, the system does not accept it unless we give the hint $(lessp \ \ (size \ \ x))$ at the end of the definition. This hint means that the $measure$ which decreases is $size(x)$. The system verifies this assertion using the following lemmas, already proved and stored with the definitions of LSB and MSB :

$(\alpha) \ \ n < size(x) \Rightarrow size(MSB(x, n)) < size(x)$

$$(\beta) \quad \left.\begin{array}{l} n < size(x) \\ n \neq 0 \end{array}\right\} \Rightarrow size(LSB(x, n)) < size(x)$$

4 Formal proof

4.1 Strategy

```
flr(x)
= if (bitvp(x) and x ≠ (btm)) then
    if all-zerosp(x) then 0
      if bit(x) then 1
      else (flr(vec (x)) + 1)
    else 0
  else 0
```

Figure 4: Definition of flr

Once the definitions have been accepted, we can verify the equivalence between the two specifications, that is :

(1) $bitvp(x) \Rightarrow first\_rec(x) = first\_loop2(x, 0)$

This problem is particularly interesting because it combines two difficulties :

- one of the functions involves an accumulator and the other one does not;
- they do not recurse according the same scheme.

Our strategy consists in splitting the problem in two sub-problems, by introducing an intermediate function which avoids the first difficulty. To eliminate the problem of having to deal with an accumulating parameter, the solution consists in producing an equivalent pure recursive definition. This is a well-known technique, in particular significant results have been proposed in [7].This new function is flr given in Figure 4.

Then, the new sub-goals to be verified are :

(2) $bitvp(x) \Rightarrow first\_loop2(x, 0) = flr(x)$

(3) $bitvp(x) \Rightarrow first\_rec(x) = flr(x)$

4.2 Proof

Subgoal (2) can not be directly proved. We verify the corresponding generalized form :

$$(4)\quad \left.\begin{array}{l} bitvp(x) \\ numberp(n) \end{array}\right\} \Rightarrow \\ if\ all\text{–}zerosp(x)\ then\ flr(x) = first\_loop2(x, n) \\ else\ flr(x) + n = first\_loop2(x, n)$$

That proof is immediate and does not require any sub-lemma . Proposition (2) is clearly obtained from (4) where n equals 0.

The proof of (3) is less obvious. Three subgoals have to be verified to achieve this proof. They are directly related to the three subcases which appear in the definition of "$first\_rec$" :

(5)
$$\left.\begin{array}{l} bitvp(x) \\ first\_rec(MSB(x, size(x) - medium(x))) \neq 0 \\ flr(MSB(x, size(x) - medium(x))) = \\ first\_rec(MSB(x, size(x) - medium(x))) \\ (induction\ hypothesis) \end{array}\right\} \\ \Rightarrow first\_rec(x) = flr(x)$$

(6)
$$\left.\begin{array}{l} bitvp(x) \\ first\_rec(MSB(x, size(x) - medium(x))) = 0 \\ first\_rec(MSB(x, size(x) - medium(x))) \neq 0 \\ flr(MSB(x, size(x) - medium(x))) = \\ first\_rec(MSB(x, size(x) - medium(x))) \\ flr(LSB(x, size(x) - medium(x))) = \\ first\_rec(LSB(x, size(x) - medium(x))) \\ (induction\ hypotheses) \end{array}\right\} \\ \Rightarrow first\_rec(x) = flr(x)$$

(7)
$$\left.\begin{array}{l} first\_rec(MSB(x, size(x) - medium(x))) = 0 \\ first\_rec(LSB(x, size(x) - medium(x))) = 0 \\ flr(MSB(x, size(x) - medium(x))) = \\ first\_rec(MSB(x, size(x) - medium(x))) \\ flr(LSB(x, size(x) - medium(x))) = \\ first\_rec(LSB(x, size(x) - medium(x))) \\ (induction\ hypotheses) \end{array}\right\} \\ \Rightarrow first\_rec(x) = 0$$

In fact, in order to guide the prover in using the induction hypotheses, we prove the following properties :

$$(8)\quad first\_rec(MSB(x, size(x) - medium(x))) \neq 0 \\ \Rightarrow flr(x) = flr(MSB(x, size(x) - medium(x)))$$

$$(9)\quad \left.\begin{array}{l} first\_rec(MSB(x, size(x) - medium(x))) = 0 \\ first\_rec(LSB(x, size(x) - medium(x))) \neq 0 \end{array}\right\} \\ \Rightarrow flr(x) = flr(LSB(x, size(x) - medium(x))) \\ + size(x) - medium(x)$$

$$(10)\quad \left.\begin{array}{l} first\_rec(MSB(x, size(x) - medium(x))) = 0 \\ first\_rec(LSB(x, size(x) - medium(x))) = 0 \end{array}\right\} \\ \Rightarrow flr(x) = 0$$

Once again, the proof of each of these propositions has to be decomposed into sub-proofs. For instance, proving the two theorems below is sufficient for deducing (8). A similar reasoning is used for (9) and (10).

$$(11)\quad \left.\begin{array}{l} bitvp(x) \\ bitvp(y) \end{array}\right\} \Rightarrow flr(v\text{–}append(x, y)) = \\ \left|\begin{array}{l} if\ (not(all\text{–}zerosp(x)))\ then\ flr(x) \\ else \\ \quad if\ (not(all\text{–}zerosp(y)))\ then\ size(x) + flr(y) \\ \quad else\ 0 \end{array}\right.$$

(12) $first\_rec(x) \neq 0 \Rightarrow (not(all\text{–}zerosp(x)))$

From (8) and the definition of "$first\_rec$", the system deduces (5).

5 Results - Conclusion

We have demonstrated that validating VHDL specifications is not straightforward but is feasible and can be mechanized using an automatic theorem prover.The statistics of this benchmark are summarized in the table below :

number of definitions	5
number of intermediate lemmas	8
total CPU time (in seconds)	48.5

The CPU times have been measured on a SPARC station 10. The number of definitions and lemmas

corresponds to the functions and theorems devoted to the benchmark. General ones are not considered.

One of the main interests of this benchmark is that it combines two difficulties :

- comparing a pseudo-iterative function with a pure recursive one;
- dealing with two different recursion schemes.

We have shown that decomposing the problem by introducing a well-chosen intermediate function can be a satisfying solution. Future work will aim at re-using this methodology in other comparable cases :

- hardware verification problems specified in terms of bit-vectors;
- proofs of programs (algorithms for sorting, searching, ...).

Acknowledgements

This work has been supported by the EEC under CHARME2 ESPRIT 6018 working group.

References

[1] *IEEE Standard VHDL Language Reference Manual*, 1988.

[2] A. Bronstein, C. L. Talcott. Formal verification of pipelines based on string-functional semantics. In L.Claesen, editor, *Formal VLSI Correctness Verification*. IFIP WG 10.2 Int. Workshop Nov. 1989, North Holland, 1990.

[3] D. Borrione, L. Pierre, A. Salem. Formal verification of VHDL descriptions in the PREVAIL environment. *IEEE Design&Test magazine*, 9(2), June 1992.

[4] D. Russinoff. A mechanical proof of quadratic reciprocity. *Journal of Automated Reasoning*, 8(1), 1992.

[5] D. Verkest, J. Vandenbergh, L. Claesen, H. De Man. A description methodology for parameterized modules in the Boyer-Moore logic. In North Holland V.Stavridou, T.Melham & R.Boute, editor, *Theorem provers in circuit design*, 1991.

[6] F. P. Burns, D. J. Kinniment, A. M. Koelmans. Correct interactive transformational synthesis of DSP hardware. In *Proc. European Design Automation Conference. Amsterdam (The Netherlands)*, 1991.

[7] G. Huet, B. Lang. Proving and applying program transformations expressed with second-order patterns. *Acta Informatica*, (11), 1978.

[8] G.Cabodi, P.Camurati, F.Corno, S.Gai, P.Prinetto, M.Sonzareorda. A new model for improving product machine traversal. In *29th ACM/IEEE Design Automaton Conference*, 1992.

[9] M. Kaufmann. Generalization in the presence of free variables : A mechanically-checked correctness proof for one algorithm. *Journal of Automated Reasoning*, 7(1), 1991.

[10] N. Shankar. A mechanical proof of the Church-Rosser theorem. Technical Report 45, Institute for Computing Science, University of Texas, Austin, 1985.

[11] O.Coudert and C. Berthet and J.C. Madre. Verification of sequentiel machines using boolean functional vectors. *Formal VLSI Correctness Verification, North-Holland*, 1989.

[12] P. Camurati, P.Prinetto. Formal verification of hardware correctness : Introduction and survey of current research. *IEEE Computer*, 21(7), July 1988.

[13] P.Ashar, A.Ghosh, S.Devadas, A.Newton. Combinational and sequential logic verification using general binary decision diagrams. In *Proc. Int. Workshop on Logic Synthesis, Research Triangle Park*, 1991.

[14] Robert S. Boyer, J S. Moore. *A Computational Logic*. Academic Press Inc (London), 1979.

[15] Robert S. Boyer, J S. Moore. *A Computational Logic Handbook*. Academic Press Inc (London), 1988.

[16] W. A. Hunt, B.C.Brock. The verification of a bit-slice alu. In NY (USA) Cornell University, Ithaca, editor, *Formal VLSI Correctness Verification*. Workshop on Hardware Specification, Verification and Synthesis : Mathematical Aspects, 1989.

[17] Warren A. Hunt Jr. FM8501 : A verified microprocessor. Technical Report 47, Institute for Computing Science, The University of Texas at Austin, 1986.

[18] Y. Yu. Computer proofs in group theory. *Journal of Automated Reasoning*, 6(3), 1990.