Two Approaches to the Formal Proof of Replicated Hardware Systems using the Boyer-Moore Theorem Prover

Eric GASCARD, Laurence PIERRE
Laboratoire d'Informatique de Marseille
CMI / Université de Provence
39, rue Joliot-Curie
13453 Marseille Cedex 13 - FRANCE
e-mail : laurence@gyptis.univ-mrs.fr

I. INTRODUCTION.

Due to their high level of reliability, formal methods are gaining acceptance in academia as well as in industry [Ha,90], [BH,95], [Gu,92]. In particular, the use of first-order theorem provers based on mechanisms such as resolution, rewriting, or induction is very promising. Many research teams are conducting work in this field, using provers like OTTER [Mc,91], LP [GG,91], RRL [KZ,87], Nqthm [BM,88],... In the framework of the PREVAIL system [BP,92], [BB,96], which proposes various proof tools for validating hardware descriptions written in VHDL [Ie,88], we focus on the application of the Boyer-Moore theorem prover, Nqthm, to the verification of the equivalence (or implication) between a circuit implementation and its specification, i.e. its expected behaviour. Nqthm is essentially based on the principles of recursion and induction. Thus, it is well-adapted to categories of circuits that can naturally be modelled recursively, in particular replicated parameterized architectures, where recursion expresses the regularity of the structure.

Here we consider one of the most widespread distributed architectures, the four neighbor torus of elementary processors [Le,91]. Due to their replicated nature, such hardware systems can easily be represented in VHDL by means of iterative or recursive "generic" descriptions, i.e. their size (relatively to the number of elementary processors) can be associated with a "generic" parameter N. This feature is particularly well-suited to the powerful induction mechanism of Nqthm. We propose two different approaches to formally reason about these devices using this prover. The first one makes use of recursive functions that faithfully translate the structural replication, and which are close to the usual VHDL descriptions. The second one is based on the notion of Cayley graphs [BM,76], [He,97], a representation of mathematical groups, in particular permutation groups. Here we consider the fact that a torus is the representation of a permutation group with four generators. In the former case, we propose a specific modelling technique and an associated proof methodology which extend previous results [Pi,95] to toruses. In the latter, we give a Nqthm definition of toruses as Cayley graphs by means of their generators, and we build a library of formally verified fundamental properties of these graphs. Proving the correctness of instances of toruses makes use of these properties. The advantages of these two approaches are respectively the simplicity of the specification, and the efficiency of the proof.

II. NQTHM AND ITS APPLICATION TO PARAMETERIZED ARCHITECTURES.

The Boyer-Moore logic is a quantifier-free first order logic with equality [BM,88]. The syntax of Nqthm is a Lisp-like syntax. Its three main principles are :

- *the "shell" principle*. Inductive abstract data types - called "shells" - can be built by means of a bottom object, a constructor, and one or more accessors. A boolean function, called a recognizer, recognizes if an object belongs to the shell.
- *the definition principle*. Prior to accepting a recursive definition, the system verifies that there exists a measure which decreases according to a "well-founded" relation.
- *the induction principle*. It allows to prove inductive theorems over recursive functions. Induction variables are automatically detected and induction schemes are automatically generated accordingly.

Some ideas about the verification of parameterized hardware with the Boyer-Moore theorem prover were already proposed in [GW,85]. More recently, the application of this prover to parameterized architectures described in Hilarics was studied at IMEC [VV,92].

This methodology applies to combinational as well as sequential devices with one-way propagated carries. The same problem has also been investigated with other provers, for instance in [KS,96], RRL is used to specify and reason over generic hardware components (the example of a carry save adder, used in the Wallace tree multiplier, is developed).

III. FOUR NEIGHBOR TORUSES.

Figure 1 depicts a NxN four neighbor torus. This interconnection network is an array of four neighbor processors where the output data on the right hand side are input to the first column $P_{0,0}$, ..., $P_{N-1,0}$, the output data on the left hand side are input to the last column $P_{0,N-1}$, ..., $P_{N-1,N-1}$, the output data on the bottom are input to the first row $P_{0,0}$, ..., $P_{0,N-1}$, and the output data at the top are input to the last row $P_{N-1,0}$, ..., $P_{N-1,N-1}$. Usually, each processor implements an elementary boolean or arithmetic function and can store data.

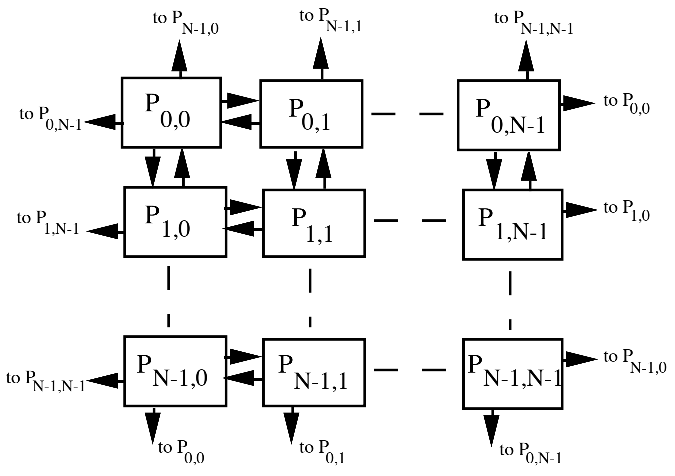

Figure 1. NxN four neighbor torus

```
entity row_max is port(ck:in bit; load:in bit;
                        V:in nat_vector(0 to N-1); C1,C2:in natural;
                        A,C:in nat_vector(0 to N-1); R:out nat_vector(0 to N-1));
end row_max;

architecture struct of row_max is
component cell_max port(ck:in bit; load:in bit;
                        m_in:in natural; a,b,l,r:in natural; m_out:out natural);
end component;
signal Carry1, Carry2, S:nat_vector(0 to N-1);
begin
   cellblock: block
   for all : cell_max use entity work.cell_max(behav);
   begin
     Row: for I in 0 to N-2 generate      -- regular series of cells
        R1:cell_max port map(ck,load,V(I),A(I),C(I),Carry2(I),Carry1(I),S(I));
        Carry2(I+1) <= S(I);      -- data propagation
        Carry1(I) <= S(I+1); R(I) <= S(I);
     end generate;
     R2:cell_max port map(ck,load,V(N-1),A(N-1),C(N-1),    -- last cell
                         Carry2(N-1),Carry1(N-1),S(N-1));
   end block;
   Carry2(0) <= C2;  R(N-1) <= S(N-1);  Carry1(N-1) <= C1;  -- initializations
end struct;
```

Figure 2. The basic row in VHDL

Our illustrative example [Bo,84] is a NxN torus where each processor stores an integer value in its internal register. Its expected behaviour is the following : after a constant number of steps, which is equal to the "diameter" of the array, i.e. N (the maximum "distance" between any two nodes, where the "distance" is the number of edges in the shortest path joining the nodes), the maximum of the array is transmitted to every processor. At each step, every processor compares its own value with the values of its four neighbors and updates its register with the maximum of these five values. We give below one of the main VHDL descriptions for this system. A VHDL description is composed of an *entity* specification, which describes the interface of the system, and one or several associated *architectures* that give particular views of the system structure or behaviour. Usually, an *architecture* contains a set of concurrent instructions which handle *signals*. Hierarchical interconnections of components can be described, using the "`for...use`" and "`port map`" constructs.

We give a structural view of the basic row of this circuit with the entity `row_max` and its architecture `struct` (Figure 2). The parameters `A` and `C` are two vectors of natural numbers (the one above the row and the one below it), `C1` and `C2` are two natural numbers (resp. the right and left input data of $P_{i,N-1}$ and $P_{i,0}$), and the vector `V` is used to initialize the registers each time a new computation starts. The structural description shows that each row consists of N instances of `cell_max` (N-1 in the "`for I in 0 to N-2 generate...`" instruction plus the last instance) and it clearly expresses data propagation between the processors via the signals `Carry1` and `Carry2`. Similarly, we describe the whole array as the interconnection of N identical rows with data propagation between them.

IV. OUR BOYER-MOORE ORIENTED APPROACHES.

IV.1 First approach : replication represented by recursive functions.

In this approach, we use two kinds of functions : a recursive function associated with the row, which expresses the regular interconnection of processors, and a recursive function associated with the array, which expresses the regular interconnection of rows. Vectors and arrays of natural numbers are modelled using specific "shells". The general patterns for these functions are given in [Pi,97]. For instance, the instanciation of our pattern for the description of Figure 2 gives :

```
(defn Row (P A C Carry1 Carry2)
   (if (and (natvp P) (natvp A) (natvp C)
            (equal (nsize A) (nsize P)) (equal (nsize C) (nsize P)))
       (if (equal P (nbtm))
           (nbtm)  ; bottom element of the shell of vectors of natural numbers
           (if (equal (nvec P) (nbtm))
               (natv (Cell_max (nat A) (nat C) Carry2 Carry1 (nat P)) (nbtm))
               (natv (Cell_max (nat A) (nat C) Carry2 (nat (nvec P)) (nat P))
                     (Row (nvec P) (nvec A) (nvec C) Carry1 (nat P)))))
       (nbtm)))
```

where `natv` is the constructor of the "shell" of vectors of natural numbers, `nat` is the accessor that extracts the first element of the vector, and `nvec` returns the rest of the vector. The function `nsize` returns the size of a vector of natural numbers, `Cell_max` gives the behavior of the elementary processor, and `P` represents the previous value of the row.

Because of the lack of place, we briefly recall the proof task, which considers two sub-problems : the validation of the behaviour of the basic row, and the verification of the correctness of the whole system. A usual way of designing or validating this kind of arrays is based on the notion of invariant (see for instance [CM,86],[GV,93]). The reasoning we adopt is also very close to this notion, we verify properties that are similar to invariants. The expected behaviour of this system is that, after at most `N` iterations (where `N` is the size of the array), the maximum element of the initial torus is propagated to every processor (less than `N` iterations are required if there are several occurrences of the maximum in the array when the algorithm starts). In fact, the maximum needs at most `N/2` iterations to reach every row, and once a row contains it, at most `N/2` iterations are needed for propagating it to every processor in the row. Thus, we prove a theorem which states that once the maximum element entered a row `V`, the number of its occurrences in `V` is incremented by 2, at least, at each computation step (except in some special cases). Similarly, we have to verify a second lemma which states that the number of rows containing the maximum is incremented by 2, at least, at each iteration (except in some special cases). Finally, it is clear that the maximum element will be propagated to every processor after at most N/2 + N/2, i.e. N iterations.

IV.2 Second approach : Cayley graphs.

Interconnection networks can be modelled by finite graphs.The vertices represent the nodes and the edges are associated with communication lines. Such a modelling is commonly used when reasoning about problems like the degree/diameter problem, the development of communication algorithms (broadcasting, gossiping,...), fault tolerance,... Here, one of our aims is the evaluation of its usefulness in the framework of formal verification.

Let G and S be a group and a subset of this group, the Cayley digraph of G and S is such that its vertices are the elements of G and its arcs are all ordered pairs (g, $g \otimes s$) where $g \in G$, $s \in S$ and $\otimes$ is the law of the group. If S is a generating set of G then the Cayley digraph is strongly connected. If S is unit free and closed under inverses then the digraph is a simple graph. Cayley graphs have the property of vertex symmetry [AK,89], hence this model is well-suited to the representation of many symmetric interconnection networks. The N$\times$N torus of Figure 1 is the cartesian product of two "cycles" of length N, a cycle is a connected graph with N vertices and degree 2. If we denote (x_1, x_2, ... x_N) the permutation σ such that $\sigma(1)=x_1$, $\sigma(2)=x_2$, ... $\sigma(N)=x_N$, then the generators of such a cycle are g_1=(N,1,2,...,N-1) and g_2=(2,3,...,N,1). Similarly, the generators of the N$\times$N torus are g_1=(N,1,2,...,N-1, N+1,...2N), g_2=(2,3,...,N,1,N+1,...2N), g_3=(1,2,...,N,2N,N+1,...2N-1) and g_4=(1,2,...,N,N+2, ...2N,N+1). Intuitively, the generators g_1 and g_2 correspond to horizontal moves along the edges, and g_3 and g_4 correspond to vertical moves. We denote g_1(i,n), resp. g_2(i,n), the image of i by the generator g_1, resp. g_2, of the cycle of length n, and $g_1$prime(i,n), resp. $g_2$prime(i,n), $g_3$prime(i,n), $g_4$prime(i,n), the image of i by the generator g_1, resp. g_2, g_3,g_4, of the n$\times$n torus. We define these functions in Nqthm as follows :

```
(defn g1 (i n)                                    (defn g2 (i n)
  (if (and (not (zerop n))                          (if (and (not (zerop n))
        (not (zerop i)) (leq i n))                          (not (zerop i)) (leq i n))
      (add1 (remainder                                  (add1 (remainder i n))
              (plus (sub1 i) (sub1 n)) n))            0))
      0))

(defn g1prime (i n)   ; similar definition for g2prime, with g2 instead of g1
  (if (and (not (zerop n)) (not (equal n 1)) (not (zerop i)) (leq i (plus n n)))
      (if (leq i n) (g1 i n) i)
      0))

(defn g3prime (i n)   ; similar definition for g4prime, with g2 instead of g1
  (if (and (not (zerop n)) (not (equal n 1)) (not (zerop i)) (leq i (plus n n)))
      (if (leq i n) i (plus (g1 (difference i n) n) n))
      0))
```

Using these definitions, we verify a set of simple but fundamental properties about the cycle and the torus. Among the torus properties, we can cite :

- $\forall$ p, $g_1^p = g_1^{p \bmod n}$, and similarly for g_2, g_3 and g_4
- $\forall$ p,q $\in$ [0,n-1], $p \neq q$, $g_1^p \neq g_1^q$, and similarly for g_2, g_3 and g_4
- $\forall$ p $\in$ [0,n-1], $g_1^p = g_2^{n-p}$, and $g_2^p = g_1^{n-p}$, $g_3^p = g_4^{n-p}$, and $g_4^p = g_3^{n-p}$
- $g_1g_3 = g_3g_1$, and $g_2g_4 = g_4g_2$
- the diameter of the n$\times$n torus is equal to 2*[n/2]

Then, the verification of the correctness of our example is straightforward since we have proven that the diameter of the torus is equal to 2*[n/2], and we know that the number of iterations needed to broadcast a value in such an interconnection network is at most equal to its diameter. It remains to verify that the value which is broadcasted in the torus is actually the maximum. To that goal, we prove that the value propagated by a processor to each of its neighbors is the maximum of the five values that it considered during the previous iteration. Thus, a value which is not the maximum can no longer be propagated.

V. CONCLUSION.

We have proposed two different approaches to the application of a first-order theorem prover, Nqthm, to the formal verification of a particular kind of distributed hardware system. Both of them take into account the replicated nature of the architecture, and they present different advantages. The Nqthm encoding for the first one is close to the VHDL descriptions and can easily be obtained from them, the proof strategy for the second one can be determined so that it takes advantage of pre-proven fundamental properties.

An important aspect is the link between VHDL and Nqthm. As far as our first approach is concerned, we have defined a translation method under the assumption that some "design for verifiability" rules are respected. A recommended modelling style guarantees that iterative VHDL descriptions can be mapped to our recursive patterns. With respect to the Cayley graphs approach, we have not yet considered this aspect. Our example has demonstrated that the iterative VHDL descriptions are useless in that case, only the description of the elementary processor is required. We can reasonably imagine an interactive user interface which proposes to the designer a choice of typical interconnection networks (torus, butterfly, hypercube,...) associated with pre-proven Nqthm libraries, so that the user only has to select one of them and to give the description of the basic processor, and the system could semi-automatically generate the Nqthm events.

REFERENCES.

[AK,89] S.AKERS, B.KRISHNAMURTHY : "A Group-Theoretic Model for Symmetric Interconnection Networks". IEEE Trans. on Computers, Vol. 38 (4), April 1989.

[BB,96] D.BORRIONE, H.BOUAMAMA, D.DEHARBE, C.LE FAOU, A.WAHBA : "HDL-based Integration of Formal Methods and CAD Tools in the PREVAIL Environment". Proc. FMCAD'96, Palo Alto (Ca), Nov. 96.

[BH,95] J.BOWEN, M.HINCHEY : "Seven More Myths of Formal Methods". IEEE Software, July 1995.

[BM,76] J.BONDY, U.MURTY : "Graph theory and applications". The MacMillan Press (UK), 1976.

[BM,88] R.S.BOYER, J S.MOORE : "A Computational Logic Handbook". Academic Press, Inc. 1988.

[Bo,84] S.H.BOKHARI : "Finding Maximum on an Array Processor with a Global Bus". IEEE Trans. on Computers, Vol. c-33, n°2. February 1984.

[BP,92] D.BORRIONE, L.PIERRE, A.SALEM : "Formal Verification of VHDL Descriptions in the PREVAIL Environment". IEEE Design&Test Magazine, Vol. 9, n°2. June 1992.

[CM,86] K.CHANDY, J.MISRA : "Systolic Algorithms as Programs". Distributed Computing, vol.1, 1986.

[GG,91] S.GARLAND, J.GUTTAG : "A guide to LP, the Larch Prover". Report DEC Systems Research Center n°82, Palo Alto (Ca), December 1991.

[Gu,92] A.GUPTA : "Formal Hardware Verification Methods : a Survey". Formal Methods in System Design, Vol.1, 1992.

[GV,93] P.GRIBOMONT, V.VAN DONGEN : "Generic Systolic Arrays : a Methodology for Systolic Design". Proc. TAPSOFT'93, LNCS n°668, Springer-Verlag, 1993.

[GW,85] S.M. GERMAN, Y. WANG : "Formal Verification of Parameterized Hardware Designs". Proc. IEEE Int. Conf. on Computer Design : VLSI in Computers, ICCD-85, October 1985.

[Ha,90] A.HALL : "Seven Myths of Formal Methods". IEEE Software, September 1990.

[He,97] M.C.HEYDEMANN : "Cayley Graphs and interconnection networks". Internal Notes, LRI (Orsay), March 1997.

[Ie,88] IEEE Standard VHDL Language Reference Manual, IEEE, 1988.

[KS,96] D.KAPUR, M.SUBRAMANIAM : "Mechanically Verifying a Family of Multiplier Circuits". Proc. CAV'96, in R.Alur & T.A.Henzinger eds., Computer Aided Verification, LNCS n°1102, Springer-Verlag.

[KZ,87] D.KAPUR, H.ZHANG : "RRL : Rewrite-Rule Laboratory User's Manual". Dept. of Computer Science, State University of New-York at Albany, June 1987.

[Le,91] F.LEIGHTON : "Introduction to parallel algorithms and architectures". Morgan Kaufmann, 1991.

[Mc,91] W.McCUNE : "What's new in OTTER 2.2". Technical Report ANL/MCS-TM-153, Mathematics and Computer Science Division, Argonne National Laboratory (Il), July 1991.

[Pi,95] L.PIERRE : "An Automatic Generalization Method for the Inductive Proof of Replicated and Parallel Architectures". In R.Kumar & T.Kropf eds., Theorem Provers in Circuit Design, LNCS n°901, Springer-Verlag.

[Pi,97] L.PIERRE : "VHDL Description, Boyer-Moore Specification and Formal Verification of a Parallel System for Finding a Maximum". Proc. Workshop "Formal Methods for Parallel Programming : Theory and Applications" (FMPPTA'97), Geneva, April 1997.

[VV,92] D.VERKEST, J.VANDENBERGH, L.CLAESEN, H.DE MAN : "A description methodology for parameterized modules in the Boyer-Moore logic". In "Theorem provers in circuit design", V.Stavridou, T.Melham & R.Boute ed., North Holland, 1992.