The validation of a bidirectional serial/parallel shift register by Evolving Algebras*

Giampaolo Bella** Elvinia Riccobene***

Abstract. We develop a new approach to perform hardware *validation* by the *formal verification* methodology and apply it to a bidirectional serial/parallel shift register. In such process, the formalism *Evolving Algebras* is used and its great simplicity in modelling the temporal evolution of real systems is stressed.

KEYWORDS: hardware validation, formal verification, Evolving Algebras.

1 Introduction

The recent progress of VLSI technology has caused an increasing level of complexity in the area of design and fabrication of hardware systems, and consequently testing their validity has become a harder task.

When hardware systems were made up of small-scale integrated circuits, *simulation* was used to check for their correct operation. In particular, the validation was achieved simply by showing that the circuit gave the expected outputs on all possible inputs.

Due to the complexity of modern systems, simulation is now inadequate: it is not practically feasible to simulate all possible input patterns to verify a hardware design because this process should be measured in man-years!

In fact, only a very reduced subset of the exhaustive set of patterns is typically simulated after the design system, in the hope that no bugs are overlooked in this process, but this is not exactly the case in practice. There exist many cases where errors have been discovered too late in the design cycle, sometimes even after the commercial production and marketing of a product. Thus, the cost of finding a design error at such a stage is very high as it often requires the redesign of large sections of the chip. Ideally, one would like to be able to detect errors at the earliest opportunity for the sake of minimising the amount of wasted design effort.

This pregnant necessity has given a deep impulse to research, so that an alternative to simulation has been proposed: the use of *formal methods* within the *formal verification* methodology for determining hardware correctness.

* **Original full paper**

** Computer Laboratory - University of Cambridge (UK)
e-mail: Giampaolo.Bella@cl.cam.ac.uk

*** Dipartimento di Matematica, Università di Catania, Viale A.Doria 6, 95125 Catania
e-mail: riccobene@dipmat.unict.it - tel: 095 330533 int. 640 - fax: 095 330094

Formal verification can be seen as a *mathematical proof* of a design correctness. In fact, "as correctness of a mathematically proven theorem holds regardless of particular values that it is applied to, correctness of a formal verified hardware design holds regardless of its input values. Thus, consideration of *all cases* is implicit in a methodology for formal verification" [6].

Lots of formalisms have been used for the formal verification of VLSI circuits. This paper is intended to show the suitability of *Evolving Algebras* [8] ("EAs" will be used to refer to the formalism, "ealgebra" to refer to a particular evolving algebra) to this approach, thanks to their simplicity of abstraction and expressiveness in modelling the temporal evolution of *real systems*. These qualities have already arisen with a very simple device in [1], where the case was stressed for the syntactical simplicity of EAs by a direct comparison with the *Circal* formalism. We now go beyond by using the same methodology to validate a real VLSI device, the bidirectional serial/parallel *shift register*, which is part of the material used by Integrated Silicon Design Pty Ltd of Adelaide in the conduct of training courses (*cMOS project 2* in [9]). This is the first success of EAs in the validation of VLSI design.

The paper is organised as follows: in Section 2 the formal verification methodology is briefly introduced. Section 3 contains the formal verification of one implementation of the bidirectional serial/parallel shift register. Section 4 concludes the work and mentions former and further findings. The guidelines of the EAs formalism are presented in Appendix.

2 Formal verification methodology

Hardware formal verification may be defined as the process to *formally establish that an Implementation is "equivalent" to a Specification.*

Specification is the name of a high-level, macroscopic, description of a device operation, whereas *Implementation* is the name of a low-level (microscopic) description of the operation of the same device. The above mentioned notion of equivalence is exhaustively explained in Appendix B. It broadly relies on the proof that certain mathematical relations hold between Specification and Implementation.

Lots of formalisms have been used to describe Specifications and Implementations (broadly categorised under *logic-based*, *automata-based* and *hybrid* formalisms) but almost all seem to suffer of syntactical complexity in order to represent the temporal evolution of real systems and of restricted flexibility in abstraction [5, 6, 4, 10].

By contrast, EAs reveal a good syntactical simplicity, thanks to which our descriptions may result quickly understandable and are so flexible that they can be tailored to any levels of abstraction. (See Appendix A for a brief introduction to the formalism).

3 The formal verification of the bidirectional serial/parallel Shift Register

Shift registers play an important role in applications implying information *temporary storage* and data *transfer* in a digital system.

We are going to consider the n-cell, bidirectional, serial/parallel shift register project presented in [9]. A *two-phase non-overlapping clock* is considered for defining the times during which data is allowed to move into and through successive processing stages of the device. The two clock phases are two signals that have the same period, are high by turns and never overlap. One clock period is enough for the two phases to be both high by turns [9].

This project has already been fabricated according to the cMOS technology and tested by simulation, but this is the first time for it to be formally verified.

Under appropriate clock signals, the device can shift data either in serial mode (throughout the cells to right or to left) or in parallel mode (from outside to the cells and viceversa). Therefore, we take into consideration the following seven non-trivial operation modes:

1. *serial right in/out*;
2. *serial left in/out*;
3. *serial right in, parallel out*;
4. *serial left in, parallel out*;
5. *parallel in/out*;
6. *parallel in, serial right out*;
7. *parallel in, serial left out.*

One clock period is needed for a bit to shift from one cell to the next one.

3.1 The Specification

We define an ealgebra $\mathcal{H}$ to carry out the Specification of the device.

Remember that any ealgebra may be viewed as a sequence of static algebras (or *states*), each formalising an instantaneous configuration of the device. Every application of one or more transition rules, starting from the initial state, causes the passing from the current state to the next one (see Appendix A for more). In the sequel, the state of any ealgebra $\mathcal{A}$ at instance of time j is indicated by A_j, and the interpretation of a given term g in a such state by $[g]_{A_j}$.

In fig. 1 we can recognise four *external signals* by which the main circuit exerts its influence on the device: *right*, which drives the bit flow through *rightin* and *rightout* channels; *left*, which drives the bit flow through *leftin* and *leftout* channels; *parallelin*, which controls *parallelindata* channels; *parallelout*, which controls *paralleloutdata* channels. They must be viewed as oracle functions, externally updated by the main circuit. In any state they must satisfy the integrity constraints described further.

In serial right mode, the out-gate of the cell $i-1$ is the same as the in-gate of the cell i, whereas in serial left mode, the out-gate of the cell i is the same as

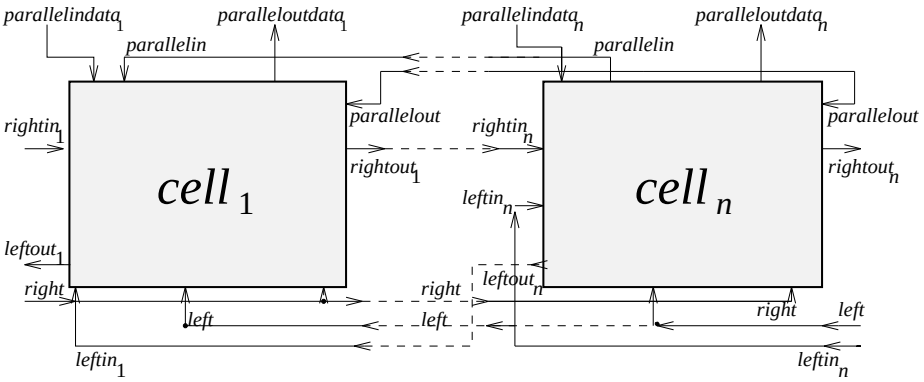

Fig. 1. Device macroscopic view

the in-gate of the cell $i-1$, $i = 2, 3, \ldots, n$. It should be observed that the four signals mentioned above are connected to each cell so that, at a given instance of time, all cells perform the same operation according to the signals.

To represent the two clock phases ϕ_1 and ϕ_2 (which are two signals that never overlap), we introduce two external functions *phi1* and *phi2*.

In order to guarantee the real working conditions of the device, we impose the following integrity constraints, holding in any state H_i ($i = 0, 1, \ldots$), amongst some external functions:

Constraint 1 (Existence of a clock phase). At any instance of time, the circuit is at least in one clock phase:

$$(phi1 \vee phi2) = 1.$$

Constraint 2 (Non-overlap of clock phases). At any instance of time, only one of the two clock phases is at logic level 1:

$$(phi1 \wedge phi2) = 0.$$

Constraint 3 (Shift right operation). Right in-gates (driven by *right*) are open only on clock phases ϕ_1, whereas all the other in-gates (driven by *left* and *parallelin*) are closed:

$$right = (phi1 \wedge \neg left \wedge \neg parallelin).$$

Constraint 4 (Shift left operation). Left in-gates (driven by *left*) are open only on clock phases ϕ_1, whereas all the other in-gates (driven by *right* and *parallelin*) are closed:

$$left = (phi1 \wedge \neg right \wedge \neg parallelin).$$

Constraint 5 (Parallel-in operation). Parallel in-gates (driven by *parallelin*) are open only on clock phases ϕ_1, whereas the other in-gates (driven by *left* e *right*) are closed:

$$parallelin = (phi1 \wedge \neg right \wedge \neg left).$$

Constraint 6 (Parallel-out operation). Parallel out-gates (driven by *parallelout*) are open only on clock phases ϕ_2:

$$parallelout = phi2.$$

The integrity constraints guarantee that:

$$phi2 \rightarrow (\neg right \wedge \neg left \wedge \neg parallelin)$$

so, neither serial shift nor parallel-in occur on clock phases ϕ_2: parallel-out only occurs. To represent the gates of the device, we define the universe *Gates* with the following elements, for $i = 1, 2, \ldots, n$:

- $rightin_i$: the serial right in-gate of cell i
- $rightout_i$: the serial right out-gate of cell i
- $leftin_i$: the serial left in-gate of cell i
- $leftout_i$: the serial left out-gate of cell i
- $parallelindata_i$: the parallel in-gate of cell i
- $paralleloutdata_i$: the parallel out-gate of cell i

Fig. 1 shows the validity of the following semantic equalities for $i = 2, 3, \ldots, n$:

$$rightout_{i-1} \equiv rightin_i \qquad\qquad leftin_{i-1} \equiv leftout_i$$

We formalise the in-out gates of the device by the universe $ExternalGates = \{IN, OUT\}$ The interpretations of its elements vary according to the type of operation (serial or parallel) of the device, so that the following semantic equalities hold:

$$IN \equiv \begin{cases} rightin_1 & \text{in right operation} \\ leftin_n & \text{in left operation} \\ [parallelindata_1, \ldots, parallelindata_n] & \text{in parallel operation} \end{cases}$$

$$OUT \equiv \begin{cases} rightout_n & \text{in right operation} \\ leftout_1 & \text{in left operation} \\ [paralleloutdata_1, \ldots, paralleloutdata_n] & \text{in parallel operation} \end{cases}$$

Nevertheless, we have to point out that *IN*, both as a single gate and as an array of gates, receives values from an external, non-deterministic oracle, while *OUT* values are updated by the transition system.

To abstractly represent the cells of the shift register, we introduce the universe:

$$Cells = \{cell_1, cell_2, \ldots, cell_n\}$$

with obvious interpretation for $cell_i$, $i = 1, 2, \ldots, n$ [4].

Finally, to take the values on cells and on gates, we define the function:

$$val : Cells \cup Gates \cup ExternalGates \longrightarrow Bits \cup \{undef\}$$

where $Bits = \{0, 1\}$ is the universe of the feasible values. *val* is an *oracle function* on *IN*.

[4] The n cells of the device could be formalised by an array, but the given formalization should be considered as a generalisation of that presented in [1].

3.1.1 The Transition system. The transition system of $\mathcal{H}$ comprises four rules[5]:

R_1) Shift-right rule

```
if right then
    val(cell_1) := val(IN);
    val(cell_{i+1}) := val(cell_i);
    val(OUT) := val(cell_n);
endif
where 1 ≤ i ≤ n − 1
```

R_2) Shift-left rule

```
if left then
    val(cell_n) := val(IN);
    val(cell_i) := val(cell_{i+1});
    val(OUT) := val(cell_1)
endif
where 1 ≤ i ≤ n − 1
```

R_3) Parallel-in rule

```
if parallelin then
    val(cell_i) := val(IN[i]);
endif
where 1 ≤ i ≤ n
```

R_4) Parallel-out rule

```
if parallelout then
    val(paralleloutdata_i) := val(cell_i);
endif
where 1 ≤ i ≤ n
```

3.1.2 The Specification Correctness. Reminding that there are only seven non-trivial operation modes, we now intend to prove that the ealgebra $\mathcal{H}$ correctly simulates the whole operation of the shift register. The sequel of the section proves the following correctness Theorem.

Theorem 1 (Correctness of the Specification). *$\mathcal{H}$ correctly simulates a bidirectional serial/parallel shift register.*

We need to prove that the input values are output after a delay depending on the length of the device and on the particular mode of operation. This is achieved by the propositions described below.

Of course each operation mode is **uniquely determined** by the **precise sequence of external signals** holding on phases ϕ_1, because on phases ϕ_2 only *parallelout* holds.

It should be stressed that the two clock phases are high by turns, without overlapping from an instant to another, as guaranteed by Constraints 1 and 2.

Without loss of generality, we impose the additional:

Constraint 7 (Initial phase). Phase ϕ_1 holds on the initial state:

$$[phi1]_{H_0} = 1$$

Thus, starting from the initial state H_0, even states represent device configurations on phases ϕ_1, whereas odd states correspond to device configurations on which ϕ_2 holds. It follows that:

Remark 1. *Algebras $2j$ and $2(j+1)$ represent system configurations which differ in one clock period.*

[5] In general, the rule `if` *cond* `then` $a_{f(i)} := b_{g(i)}$ `endif where` $1 \leq i \leq m$ is an abbreviation for the rule `if` *cond* `then` $a_{f(1)} := b_{g(1)}$; $a_{f(2)} := b_{g(2)}; \ldots; a_{f(m)} := b_{g(m)}$ `endif`.

In serial in/out mode, our Specification must guarantee that the output is delayed of n clock periods. Therefore, according to Remark 1, each input bit should be given as output in $2n+1$ transitions (the last one transfers the bit from the final cell to the output variable). Remember that in serial right operation IN is interpreted as $rightin_1$ and OUT as $rightout_n$, while in serial left operation IN is interpreted as $leftin_n$ and OUT as $leftout_1$. As said above, in the following we can indicate only the signals holding on phases ϕ_1.

Proposition 1 (Correctness in serial right (left) in/out mode). *If, starting from the $2j$-th state, the sequence of signals*

$$\underbrace{right(left), \ldots, right(left)}_{n+1}$$

occur, the following correctness relation holds:

$$[val(OUT)]_{H_{2j+2n+1}} = [val(IN)]_{H_{2j}}, \quad j \geq 0.$$

Proof. By induction on j. Starting from the state H_{2j}, we simulate the device operation by firing (see A2 for details) the transition rules. □

Corollary 1 (Shift Time). *In case of serial shift, the value stored in a cell p, at any instance of time t, is shifted into a successive cell q, in accordance with the fixed shift direction, in $2d$ transitions, being $d = |p - q|$; that is:*

$$[val(cell_p)]_{H_t} = [val(cell_q)]_{H_{t+2d}} \quad t = 0, 1, \ldots$$

Proof. The Corollary is trivially true, in both directions, if $p = q$. Otherwise, the result comes from the relation

$$[val(cell_k)]_{H_t} = [val(cell_{k+1})]_{H_{t+2}} = \ldots = [val(cell_{k+d})]_{H_{t+2d}} \quad t = 0, 1, \ldots$$

driven by the transition system, where $p < q$ and $k = p$ in right mode, and $p > q$ and $k = q$ in left mode. □

In case of parallel input and serial output, one transition must load all the cells. After that, the bit of the i-th cell must be shifted right ($n - i + 1$ cells to cross) or left (i cells to cross) along the device.

Proposition 2 (Correctness in parallel in, serial right (left) out mode). *If, starting from the $2j$-th state, the sequence of signals*

$$parallelin, \underbrace{right(left), \ldots, right(left)}_{n}$$

occur, the following correctness relation holds for $k = n - i + 1$ (resp. $k = i$):

$$[val(OUT)]_{H_{2j+2k+1}} = [val(IN[i])]_{H_{2j}}, \quad j \geq 0, \ i = 1, 2, \ldots, n.$$

Proof. By application of R_3, $[val(cell_i)]_{H_{2j+1}} = [val(IN[i])]_{H_{2j}}$ yields; combining it together with the result of the Corollary 1, applied to cells i and n (resp. 1 and i), it follows $[val(cell_h)]_{H_{2j+2k-1}} = [val(IN[i])]_{H_{2j}}$, for $k = n - i + 1$ and $h = n$ (resp. $k = i$ and $h = 1$), from which the thesis comes out by applications of R_4 and then of R_1 (resp. R_2). □

Lemma 1 (Cell contents). *During a serial right (left) operation, the contents of the i-th cell, $1 \leq i \leq n$, in a fixed state H_t, are related to the input as follows:*

$$[val(cell_i)\,]_{H_t} = \begin{cases} [val(IN)]_{H_{t-2k+1}} & odd(t) \wedge t \geq 2k-1 \\ [val(cell_i)]_{H_{t-1}} & even(t) \wedge t \neq 0 \end{cases}$$

where $k = i$ (resp. $k = n - i + 1$).

Proof. For even t, the result is trivially true because only rule R_4 can be fired, without updating the contents of any cells.
Let t be an odd value. By application of R_1 (resp. R_2) it follows:

$$[val(IN)]_{H_{t-2k+1}} = [val(cell_h)]_{H_{t-2k+2}}$$

with $h = 1$ (resp. $h = n$). The result comes out from the application of Corollary 1 to cells 1 and i (resp. i and n). □

In case of serial input and parallel output, we expect $2n - 1$ transitions to load the n cells and the $2n$-th one (on ϕ_2) to give the contents of the whole device as output array.

Proposition 3 (Correctness in serial right (left) in, parallel out mode). *If, starting from the $2j$-th state, the sequence of signals*

$$\underbrace{right(left), \ldots, right(left)}_{n}$$

occur, the following correctness relation holds for $k = n$ (resp. $k = 1$) and $j \geq 0$:

$$[val(OUT)]_{H_{2j+2n}} = [[val(IN)]_{H_{2j+2|k-1|}}, [val(IN)]_{H_{2j+2|k-2|}}, \ldots, \\ \ldots, [val(IN)]_{H_{2j+2|k-n+1|}}, [val(IN)]_{H_{2j+2|k-n|}}], \quad j \geq 0.$$

Proof. By application of Lemma 1, for $t = 2j + 2n - 1$, to all n cells and then of rule R_4. □

In full parallel operation both input and output are n-length arrays. One transition only should load the device and another one should unload it.

Proposition 4 (Correctness in parallel in, parallel out mode). *If, starting from the $2j$-th state, the signal*

$$parallelin$$

occurs, the following correctness relation holds:

$$[val(OUT)]_{H_{2j+2}} = [val(IN)]_{H_{2j}}, \quad j \geq 0$$

Proof. By applications of rule R_3 and next of R_4. □

Propositions 1, 2, 3, 4 prove the Theorem 1.

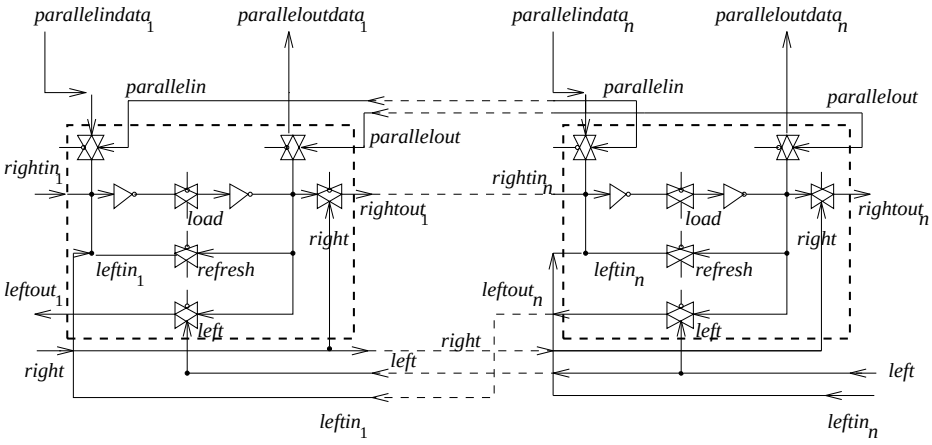

Fig. 2. Device microscopic view

3.2 The Implementation

We now examine the design of the bidirectional serial/parallel shift register in which each cell is made up of two inverters serially connected. This design is discussed in [9] and is not to be intended as the most efficient; our aim is indeed to show that our approach is suitable to validate a given implementation.

We opportunely modify universes and functions of the ealgebra $\mathcal{H}$, the Specification, in order to get a new ealgebra $\mathcal{L}$, the Implementation, desciribing the shift register at a low level of abstraction.

The signature of the ealgebra $\mathcal{L}$ differs from the signature of the ealgebra $\mathcal{H}$ in the presence of some new functions, described below, instead of the universe *Cells*[6].

In fig. 2, we can see that each cell contains six *pass transistors*, one per each of the four signals mentioned in the Specification and two others driven by the signals:

- *load*, which isolates the two inverters in each cell, allowing the cells to be loaded;
- *refresh*, which allows the output of the second inverter to be the input of the first, in case all the input gates (*leftin, rightin, parallelindata*) are closed, the respective signals being low;

which are to be considered two other oracle functions. Constraint 7 must be rearranged to be imposed on the initial state L_0, while constraints $1, \ldots, 7$ must be imposed on any state L_i $(i = 0, 1, \ldots)$ together with:

[6] The domain of the function *val* is restricted.

Constraint 8 (Cell loading). Cells are loaded on ϕ_2:

$$load = phi2$$

Constraint 9 (Refresh activation). If all the input gates of the cells are closed, *refresh* is activated so that the cell contents are not lost:

$$refresh = (phi1 \wedge \neg left \wedge \neg right \wedge \neg parallelin)$$

We extend the universe *Gates* by the following set of in-out gates of the $2n$ inverters:

$$\{IN1_1, IN1_2, \ldots, IN1_n, OUT1_1, OUT1_2, \ldots, OUT1_n,\\ IN2_1, IN2_2, \ldots, IN2_n, OUT2_1, OUT2_2, \ldots, OUT2_n\}$$

on which the following semantic equalities hold:

- for $i = 1, 2, \ldots, n$
  - $IN1_i \equiv \begin{cases} rightin_i & \text{in right operation} \\ leftin_i & \text{in left operation} \\ parallelindata_i & \text{in parallel operation} \end{cases}$
  - $OUT1_i \equiv IN2_i$
  - $OUT2_i \equiv \begin{cases} rightout_i & \text{in right operation} \\ leftout_i & \text{in left operation} \\ parralleloutdata_i & \text{in parallel operation} \end{cases}$
- for $i = 2, 3, \ldots, n$, $OUT2_{i-1} \equiv IN1_i$

The universe *ExternalGates* remains unchanged.

3.2.1 The Transition system. The transition system of $\mathcal{L}$ consists of the following six rules:

R'_1) Shift-right rule
```
if right then
   val(OUT1_1) := ¬val(IN);
   val(OUT1_{i+1}) := ¬val(OUT2_i);
   val(OUT) := ¬val(OUT1_n);
endif
where 1 ≤ i ≤ n − 1
```

R'_2) Shift-left rule
```
if left then
   val(OUT1_n) := ¬val(IN);
   val(OUT1_i) := ¬val(OUT2_{i+1});
   val(OUT) := ¬val(OUT1_1);
endif
where 1 ≤ i ≤ n − 1
```

R'_3) Parallel-in rule
```
if parallelin then
   val(OUT1_i) := ¬val(IN[i]);
   endif
where 1 ≤ i ≤ n
```

R'_4) Parallel-out rule
```
if parallelout then
   val(parralleloutdata_i) := ¬val(OUT1_i);
endif
where 1 ≤ i ≤ n
```

R'_5) Loading rule
```
if load then
   val(OUT2_i) := ¬val(OUT1_i);
endif
where 1 ≤ i ≤ n
```

R'_6) Refresh rule
```
if refresh then
   val(OUT1_i) := ¬val(OUT2_i);
endif
where 1 ≤ i ≤ n
```

Rules R'_4 and R'_5 will be always applied simultaneously because both signals *parallelout* and *load* are high on phases ϕ_2.

3.3 The equivalence of the two levels

To suitably define a *verification function* in order to prove the *operational equivalence* of $\mathcal{H}$ and $\mathcal{L}$ (see Appendix B), the following technical lemma gives us a useful intuition.

Lemma 2. *Let H_t be a state of $\mathcal{H}$ and L_t be the state of $\mathcal{L}$ at the same instance of time t. The following relation holds for all $t \geq 0$:*

$$[val(cell_i)]_{H_t} = \neg[val(OUT1_i)]_{L_t},\ 1 \leq i \leq n$$

The thesis has been easily proved by simulation, taking into account the three possible way of loading a cell: left, right and parallel.

Theorem 2 (Correctness and Completeness). *Let $\mathcal{F} : \mathcal{L} \longrightarrow \mathcal{H}$ be a function such that:*

a) $\mathcal{F}$ maps a compound structure of $\mathcal{L}$ into a nullary function of $\mathcal{H}$ as follows:[7]

$$(IN1_i, OUT1_i \equiv IN2_i, OUT2_i) \stackrel{\mathcal{F}}{\longmapsto} cell_i, \qquad i = 1, 2, \ldots, n;$$

b) $\mathcal{F}$ is the identity function on the rest of the $\mathcal{L}$-signature;

c)
- *$\mathcal{F}(R'_i) = R_i$, being R'_i ($i = 1, 2, 3$) an $\mathcal{L}$-rule and R_i the corresponding $\mathcal{H}$-rule;*
- *$\mathcal{F}(R'_4 \| R'_5) = R_4$,*[8] *being R'_4 and R'_5 $\mathcal{L}$-rules and R_4 the $\mathcal{H}$-rule;*
- *$\mathcal{F}([\bar{R}', R'_6, \bar{\bar{R}}']) = [\mathcal{F}(\bar{R}'), \mathcal{F}(\bar{\bar{R}}')]$, with $\bar{R}', \bar{\bar{R}}' \in \{R'_1, R'_2, R'_3, R'_4, R'_5\}$*

d) $[val(cell_i)]_{\mathcal{F}(L_t)} = \neg[val(OUT1_i)]_{L_t}$, for $i = 1, 2, \ldots, n$, $t \geq 0$[9]

The function $\mathcal{F}$ commutes the following diagram:

$$\begin{array}{ccc} H_t & \xrightarrow{\mathcal{F}(R'_i)} & H_{t+1} \\ {\scriptstyle\mathcal{F}}\uparrow & & \uparrow{\scriptstyle\mathcal{F}} \\ L_t & \xrightarrow[R'_i]{} & L_{t+1} \end{array}$$

Proof. It is straightforward to show that the function $\mathcal{F}$ satisfies the property $\mathcal{F}(L_t) = H_t$, for every instance of time t and for every possible state L_t. This guarantees the commutation of the diagram. □

[7] If a function between ealgebras maps a compound structure into an atomic object, then each component of the structure is mapped into the whole object.

[8] $R'_4 \| R'_5$ means parallel application of R'_4 and R'_5.

[9] This propriety of $\mathcal{F}$ is suggested by Lemma 2.

4 Conclusion

We have presented the formal verification of a bidirectional serial/parallel shift register using EAs. This is the first time the formalism has proved to be suitable to perform the formal verification of VLSI devices thanks to its great simplicity in modelling the temporal evolution of real systems, and its flexibility to describe a system at any level of abstraction.

This finding can be viewed as a link between the first success achieved in [1] with a very simple device and the hierarchical extension presented in [2].

In conclusion, the authors are quite optimistic about the future developments of this kind of research, as it could be consistently useful to produce error-free circuits.

References

1. G. Bella, E. Riccobene. Le Algebre Evolventi per la validazione di hardware. *Atti Congresso AICA '96*, 1996.
2. G. Bella, E. Riccobene. Hardware Hierarchical Validation by Evolving Algebras. (In preparation).
3. E. Börger. *Specification and Validation Methods*, Oxford University Press, 1994.
4. P. Camurati, P. Prinetto. Formal Verification of Hardware Correctness: Introduction and Survey of Current Research. *Formal Verification of Hardware Design.* (Ed. Michael Yoeli) IEEE Computer Society Press Tutorial, 1988.
5. B. S. Davie. *Formal Specification and Verification in VLSI Design.* Edinburgh University Press, 1990.
6. A. Gupta. Formal Hardware Verification Methods: A Survey. *Formal Methods in System Design*, vol.5. Kluwer Academic Publishers, 1995.
7. Y. Gurevich. Logic and the challenge of computer science. In E. Börger, editor, *Current Trends in Theoretical Computer Science*, Computer Science Press, 1988.
8. Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, Oxford University Press, 1994.
9. D. A. Pucknell, K. Eshraghian. *Basic VLSI Design.* Prentice Hall, 1988.
10. Hossein Saiedian. An Invitation to Formal Methods. *IEEE Computer*, vol.29, n.4, April 1996.

A Evolving Algebras

An exhaustive presentation of the formalism can be found in [8]. Only the basic concepts are introduced here.

The notion of *Evolving Algebras* has been defined in 1988 by Gurevich in an attempt to sharpen Turing's thesis by considerations from complexity theory (see [7] for more explanations). Developed as a specification method for the semantics of (complex) programming languages (C, Prolog, Parlog, Occam, VHDL, etc.), the versatility of the formalism has also allowed one to use it as verification method for implementations of programming languages, compilers (WAM), architectures (APE, PVM, Transputer), protocols (Bakery Algorithm,

Kermit, Kerberos, Group Membership Protocol), real-time algorithms (Railroad Crossing Problem) (see *Annotated bibliography on Evolving Algebras* in [3]).

These numerous real-world case studies have shown that, using the notion of EAs, one can develop elegant, powerful, simple and easily understandable specification methodology which has a huge and yet unexplored potential for industrial applications.

The basic idea of EAs consists in formalising the instantaneous configuration of a real dynamic system S by a first-order structure without relations, here called *static algebra* or *state*. Thus, the whole dynamics of S will be represented by a (finite or infinite) sequence of states.

An ealgebra $\mathcal{A}$ is a couple consisting of an *initial state* A_0 – the initial picture of the system with its objects and related functions – and a finite set of transition rules P, called *transition system*, which mirrors the dynamics of the system, i.e. its temporal evolution.

A.1 Universes

Each real system S deals with certain objects which may be classified into different categories. These objects may be atomic (*basic elements*) or they can be structured by other elements. They may have certain properties and be related with other objects. In an ealgebra modelling S, each category of objects is formalised by a corresponding *set* of elements – sets are also called *universes*–.

Universes may be equipped with certain functions, providing some basic features of their elements, which can be used in the operations to be performed by the system. Properties and possible relations among objects are expressed in terms of corresponding mathematical conditions (called *integrity constraints*) which are required to be satisfied by the corresponding elements.

Universes of which cardinality and elements do not change are called *static*, but, in general, the EAs framework also permits universes to be dynamic. Since the main intention of the EAs concept is to reflect closely the dynamics of the system, there is a construct for growing of universes, which has the form:

$$\texttt{extend}\ U\ \texttt{by}\ x_1, \ldots, x_n\ \texttt{with}\ \mathit{Updates}\ \texttt{endextend}$$

where *Updates* may (and should) depend on the x_i and are used to define certain properties for (some of) the new objects x_i of the resulting universe U.

A.2 Transition rules

Elementary operations which are performed on the objects of S are represented by partial *functions* and we write $f(x) = undef$ if f is undefined at x [10].

Functions with different interpretation in different states, are called *dynamic*, while the others are said *static*.

[10] By definition $f(x) = undef$ if the argument x is undefined.

We express the evolution of the system by *function updates*:

$$f(t_1, \ldots, t_n) := t$$

where f is an arbitrary n-ary function and $t_1, \ldots, t_n, t$ are first-order terms. The meaning of the rule is that the function f yields t on parameters $t_1, \ldots, t_n$. We call *transition rule* any set of function updates. Note that no rule can change the signature of any ealgebra.

Rules can be conditioned by guards (*guarded transition rules*):

$$\texttt{if } g \texttt{ then } R_1 \texttt{ else } R_2 \texttt{ endif}$$

where g is a ground term and R_1 and R_2 are rules. If g is true, the rule is fired by simultaneous execution of all updates of the rule R_1, otherwise of the rule R_2.

The transition rules must always be constructed so that guards guarantee *consistency* of updates. The application of a transition rule R to a state A_i, produces another state A_{i+1} which differs from A_i in those functions that are updated by the rule R.

We call *transition system* any set of (guarded) transition rules.

A.3 Oracle functions

The operational behaviour of a real system may be affected by the environment. The EAs approach allows one to represent such external influence through the concept of *oracle*. If a function f is not static and has no updates of the form $f(t_1, \ldots, t_n) := t$ in any transition rule – i.e. it is not internally updated – it is called *oracle function* and its values are non-deterministically given by an oracle [8].

A.4 The operational equivalence of two ealgebras

In order to validate an Implementation, we have to prove its correctness and completeness w.r.t. a Specification, that is showing that the Implementation allows the same interpretation for certain logic names to hold as the Specification does.

In general, if $\mathcal{L}$ is an ealgebra representing a lower level description and $\mathcal{H}$ is an ealgebra representing a higher level description, we need to assert the *correctness and completeness* of $\mathcal{L}$ with respect to $\mathcal{H}$. This can be achieved by putting $\mathcal{L}$ in relation with $\mathcal{H}$ using a suitable *verification function*

$$\mathcal{F} : \mathcal{L} \longrightarrow \mathcal{H}$$

mapping $\mathcal{L}$-states L into $\mathcal{H}$-states $\mathcal{F}(L)$ and sequences of $\mathcal{L}$-rules R' into sequences of $\mathcal{H}$-rules $\mathcal{F}(R')$, in a such way that the diagram in fig. 3 commutes. The verification function $\mathcal{F}$ establish the *correctness* of $\mathcal{L}$ w.r.t. $\mathcal{H}$ if it allows the same outputs obtained in $\mathcal{L}$ under certain inputs to be obtained in $\mathcal{H}$ too,

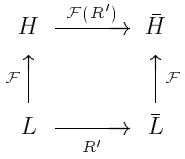

Fig. 3. Commutation Diagram between $\mathcal{L}$ and $\mathcal{H}$.

under the same inputs. If this is the case, we may view computations of $\mathcal{L}$ as implementing correctly computations of the higher level ealgebra $\mathcal{H}$.

On the other hand, we may consider the same function $\mathcal{F}$ to establish *completeness* of $\mathcal{L}$ w.r.t. $\mathcal{H}$, if every computation in $\mathcal{H}$ is the imagine under $\mathcal{F}$ of a computation in $\mathcal{L}$, since in that case we may view every *abstract* computation as implemented by a *concrete* one.

In case we establish both *correctness and completeness* in the above sense, we may speak of *operational equivalence* of the two ealgebras.

This article was processed using the LaTeX macro package with LLNCS style