{ This is a revised version of the paper that appeared in the proceedings CHDL'93 **April 26-26, 1993}**

SPECIFYING HARDWARE SYSTEMS IN LOTOS

Mohammed Faci and Luigi Logrippo

University of Ottawa, Protocols Research Group, Dept. of Computer Science, Ottawa, Ontario, Canada K1N 6N5, Email: mfaci{luigi}@csi.uottawa.ca

Abstract

This paper reports on some initial results in using LOTOS as a hardware description language. LOTOS, the *Language Of Temporal Ordering Specifications*, is a language that has been conceived in the framework of Open Systems Interconnection (OSI) standardization as a tool for the formal description of OSI services and protocols. We present some examples to show how LOTOS can be applied to the specification of hardware systems. The resulting specifications are obtained by *mapping* hardware components into LOTOS processes and connection wires into LOTOS interaction points. Much of the power of this mapping mechanism is based on the *parallel composition operator*, which is *central* to the design of highly concurrent systems, across many levels of abstraction.

Keyword Codes: D.1.3; D.2.1; D.3.1
Keywords: Concurrent Description languages, Formal Specification, LOTOS**.**

1. Introduction and Background

S*tructural hierarchy* is the most widely used method for describing highly complex and concurrent systems such as VLSI systems. In this methodology, the designer begins by a high level description of the system's functionality. Then a top down approach is used to decompose the system into several interacting subsystems. One of the advantages of such an approach is that it allows the designer to confess ignorance about the internal structure of the system components and postpone dealing with the details. The designer is able to concentrate on the global view of the system and the interfaces between the modules. The lack of appropriate languages which support the requirements of this methodology in a natural way has motivated researchers and organizations to develop new languages, such as CIRCAL[Miln91] and VHDL[Aylor86].

LOTOS[LFH92, BoBr87], *Language Of Temporal Ordering Specifications*, is a language which has been conceived in the framework of OSI standardization as a tool for the formal description of OSI services and protocols. We claim, however, that the concepts of LOTOS are general enough to make the language useful for a wide range of applications. Our research group has applied LOTOS to a number of application areas such as distributed algorithms[HL91] and telephone systems[FLS91]. In this paper, we address the issue of using

LOTOS for the formal specification of digital systems.

There are several advantages for using LOTOS in the domain of hardware specifications.

- First, LOTOS has a *formally defined syntax, static semantics, and dynamic semantics* which are described operationally in terms of inference rules [BoBr87].
- Second, it is based on a set of sound mathematical theories, due mostly to Milner [Mil80] and Hoare [Hoa85], which allow designers to prove a rich set of algebraic equivalence properties, based on several types of equivalence relations.
- Third, LOTOS semantics are defined operationally. This means that LOTOS specifications can be directly *simulated* on an interpreter [LOBF88] as well as, with some user supplied information, *translated* into an executable program [MM89].
- Fourth, LOTOS supports many levels of abstractions and favors stepwise decomposition of processes. By using the LOTOS *parallel composition operator* and the *hide* operator, systems can be described at many levels of abstractions.
- Last, but not least, LOTOS offers a facility that very few hardware specification languages offer*: Multiway synchronization.* Multiway synchronization allows many processes to synchronize their actions simultaneously. This is a very powerful tool for specifying, for example, connections between a component which drives many other components, a very common characteristic of hardware systems. Other languages which support the multiway synchronization include CIRCAL[Miln91], a language for hardware specification, which predates and shares many of its concepts with CCS and CSP.

2. Formal Specifications of Small Digital Circuits

In this paper, we are interested in showing how the concepts of LOTOS can be used in specifying small hardware modules. Therefore, we have chosen to walk the reader through a series of simple examples, at the gate level, so that we do not divert our attention to explaining details about the functionalities of our examples. The simplicity of the examples also allows us to introduce relevant LOTOS concepts as the need arises.

2.1 Specifying Combinational Circuits

In this section, we specify a full adder, using the bottom up approach, to demonstrate that LOTOS is well suited for specifying combinational circuits. The LOTOS constraint-oriented style[VSV88] is well suited for expressing the constraints that each component must satisfy. This reflects the fact that, for hardware systems, the system's behaviour is expressed by the composed behaviours of its sub-components. To specify the full adder shown in Fig. 1, we begin by specifying the components at the lowest level of abstraction. This means the specifications of the *Or* gate and the gates which define the half adder, namely, *And* and *Xo*r. The specification of the full adder is obtained by composing two half adders and an *Or* gate.

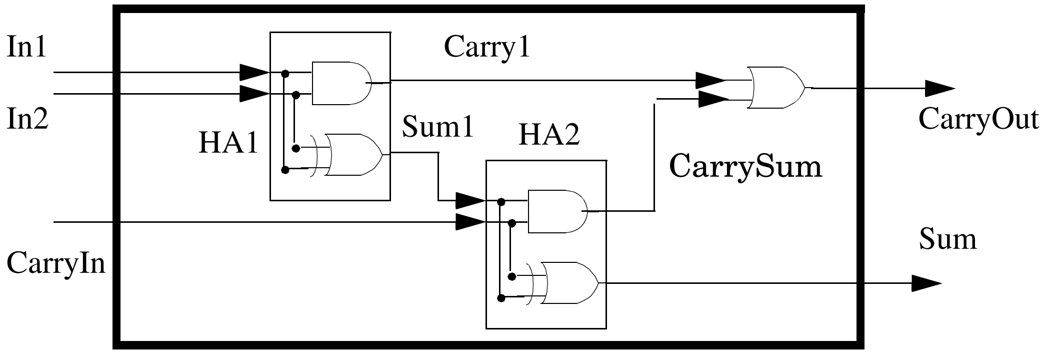

Fig. 1. Structure of a Full Adder

We represent a two-input logic gate by a LOTOS process which interacts with its environment through three interaction points. A process in LOTOS is defined by its name, a list of interaction points, a list of parameters, a functionality and a behaviour expression. For example, the specification of a two-input logical *And* is given in Fig. 2. It takes two inputs *x* and *y* and produces an output *(x AND y)*. The *AND* function is defined as an abstract data type which implements the functionality of the logical *And*. The interleaving operator (|||) is used to express the desired behaviour of this component, meaning that *x* and *y* can be accepted in any order. Other logical functions such as *OR, XOR, NOR, NOT, NAND* are described in a similar fashion, by replacing the abstract definition of the *AND* function by the respective functions.

```
process AndGate [In1, In2, Out]: exit:=
(     In1 ?x: Bit;          exit(x, any Bit)
      |||
      In2 ?y:Bit;           exit(any Bit, y)
) >> accept x, y: Bit in ( Out ! (x AND y); exit )
endproc
```

Fig. 2. Specification of Logic And Gate in LOTOS

The next level of abstraction, bottom up, concerns the specification of the half adder's behaviour. It is the composed behaviours of an *And* gate and an *Xor* gate, as shown in Fig. 3. Process AndGate has three observable interaction points, In1, In2, and Carry. Process XorGate has three interaction points as well, In1, In2, and Sum. Note that two inputs, In1 and In2, are shared by the two processes. This implies that the two processes must synchronize on these two inputs. This is expressed by using the selective parallel composition operator |[In1, In2]|.

```
process HalfAdder [In1, In2, Carry, Sum]: exit:=
(
    AndGate[In1, In2, Carry] |[In1, In2]| XorGate[In1, In2, Sum]
)
endproc
```

Fig. 3. Specification of Half Adder in LOTOS.

The structure of a full adder, as shown in Fig. 1, is expressed in terms of two half adders and an *Or* gate. While the *Or* gate is considered to be a basic building block, the half adder is composed of two other basic building blocks. The full adder's behaviour can now be expressed by composing two half adders and an *Or* gate. The result is shown in Fig. 4. Note that Sum1, Carry1, and CarrySum are specified as hidden interaction points. These interaction points are not observable from the environment.

```
process FullAdder[In1, In2, CarryIn, CarryOut, Sum]: noexit:=

 hide Sum1, Carry1, CarrySum in
 (
        (HalfAdder[In1, In2, Carry1, Sum1] |[Sum1]| HalfAdder[CarryIn, Sum1, CarrySum, Sum] )
        |[Carry1, CarrySum]|
        OrGate[Carry1, CarrySum, CarryOut]
)
>> FullAdder[In1, In2, CarryIn, CarryOut, Sum]
 endproc
```

Fig. 4. LOTOS specification of a full adder

The first instance of the process *HalfAdder (*HA1) in Fig. 1 takes In1 and In2 as inputs and produces Carry1 and Sum1 as outputs. Sum1 is then used as an input, along with CarryIn, to the second instance (HA2) of the half adder. This relation between the two half adders implies that the second half adder can only proceed when the first half adder has produced its output Sum1, even if the CarryIn input is already present. This type of information sharing is presented by making the two half adders synchronize on the interaction point Sum1. This structure guarantees that the two half adders accept their inputs and produce their outputs independent of each other except for Sum1, which is used as output from *HA1* and input to *HA2*. The same reasoning is also applied between the two half adders, taken as one component, and the *Or* Gate. The signals on Both Carry1 and CarrySum must be present before the *Or* Gate can be activated.

2.2 Specifying Sequential Circuits

In this section, we turn our attention to the more general model of switching circuits, *sequential systems*. In combinational circuits, the present value of the outputs is a function of the present values of the inputs. In sequential circuits, the present values of the outputs are a function of both the present values of the inputs as well as the values of the previous state of the system.

Sequential circuits are classified into two categories: Synchronous circuits and asynchronous circuits. In synchronous circuits the state of the circuit changes only in response to a clock pulse. In asynchronous circuits, a change in any signal may produce an immediate change in the state of the circuit. The approach taken in specifying both categories is similar. The difference is that, for synchronous circuits, the clock must be explicitly modelled by an extra interaction point. Because of space restrictions, we will deal with the specification of asynchronous circuits only.

2.2.1 Specification of a Set Reset flip flop

In this section we consider the specification of a set reset flip-flop. The proper functionality of this flip-flop requires that the two outputs are complements of each other. One output is labelled Qp and the other one Qb. A pulse on the Set input drives the output Qp to 1; it sets the flip-flop. A pulse on the Reset input drives the Qp to 0; it resets the flip-flop. The normal operation of the flip-flop will be violated if both inputs are allowed to be set to 1.

Our task is to first capture the structure of the set reset flip flop, then augment it with some parameters, which are necessary for expressing the behaviour of the flip-flop. From an observational point of view, the flip flop can be modelled by the diagram shown in Fig. 5. This diagram identifies only the

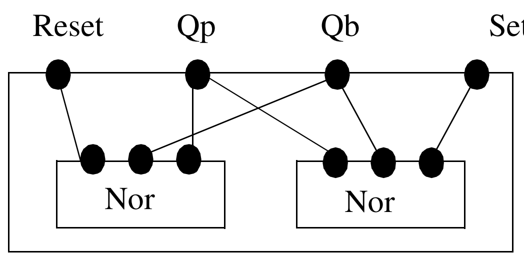

Fig. 5. Flip flop interaction with environment.

observable interaction points through which the flip flop interacts with its environment. It does not define the order of events that are exchanged, with the environment, at these interaction points. For example, the diagram expresses the fact that Qp is an interaction point shared between the environment and the two Nor gates. In other words, both Nor gates and the environment must synchronize on Qp.

For sequential circuits, it is assumed that only one signal can change at a time. This particular characteristic of sequential circuits has strongly influenced our style for structuring LOTOS specifications. Where the interleaving operator was dominant in the case of combinational circuits, the choice operator '[]' becomes dominant here instead. Fig. 6 shows the representation of the flip flop in terms of a LOTOS specification. When the process SrFlipFlop is first instantiated, a set of initial values representing the inputs *Set* and *Reset* and the outputs *Qp* and *Qb* are provided to it by the environment. This is represented by the list of parameters, Ps, Pr, Pqp, Pqb: Bit, at the header of the process, meaning that the process expects four values, of type Bit, which define its initial state. The actual behaviour of the flip flop is expressed by the composition of two Nor gates, line 2. The body of the Nor process, lines 4 to 13, expresses the behaviour of the Nor gate. It shows that at any instance, a choice exists between the following alternatives: First, change the *Set* signal. This is shown at line 5, where the environment may supply a new value for the *Set* signal resulting in another state.

The effects of this signal, which leads to a new state, is expressed by the recursive instantiation of the Nor process at line 6. Second, change the *Reset* signal. This is similar to the previous case, except that the change affects the *Reset* signal and the values which depend on it. And third, output the *Nor* of the current input values and keep the present state of the flip flop unchanged.

```
1     process SrFlipFlop[S, R, Qp, Qb](Ps, Pr, Pqp, Pqb: Bit) : noexit :=
2         Nor [S, Qp, Qb](Ps, Pqp, Pqb) |[Qp, Qb]| Nor [R, Qb, Qp](Pr, Pqb, Pqp)
3         where
4             process Nor [In1, In2, Out](Pin1, Pin2, Pout: Bit) : noexit :=
5                 In1 ?x: Bit;
6                 Nor [In1, In2, Out](x, Pin2, Not(x Or Pin2))
7                 []
8                 In2 ?y: Bit;
9                 Nor [In1, In2, Out](Pin1, y, Not(Pin1 Or y))
10                []
11                Out !Pout;
12                Nor [In1, In2, Out](Pin1, Pin2, Pout)
13            endproc
14    endproc
```

Fig. 6. LOTOS specification of the set reset flip flop.

The reader may have observed that we do not consider the restrictions imposed on the inputs (i.e., both *Set* and *Reset* being 1 at the same time) as part of the specification itself. This is consistent with the view that this restriction is really a constraint which must be enforced by the environment in which the flip-flop is expected to be used.

3. Validating the Specification

Validation of circuit designs has been the subject of much research [CaPr88]. One of the advantages of specifying circuits in LOTOS is that the language has well-established theories for formal verification and testing, on which formal disciplines of circuit verification and testing can be based.

The simplest validation method is simulation. Its objective is to detect design and functional errors. Using the simulation based approach, the designer concentrates on specifying the behaviour of the system first. Then, once the system is completely specified, the validation of the behaviour begins. Basically, there are two modes of simulation: *value-based* simulation and *symbolic* simulation. In value-based simulation, a batch of test cases is selected, based on ad hoc methods and using the specifier's intuition and knowledge. Then, each of these value test cases are applied to the circuit under test. If the stimuli produce the expected behaviour of the circuit, the test cases are recorded as a success. The specifier iterates through this process until he/she feels that the design is robust. In symbolic simulation, the designer uses variables to cover a complete range of values rather than trace the control sequence of the circuit for each value. Simulation can be automated by running the system in parallel with testing processes, also specified in LOTOS.

During the past few years, we have been using a LOTOS simulator, called *ISLA (Interactive System for LOTOS Applications),* designed and developed at the University of Ottawa[LOBF88], to validate LOTOS specifications. Among other functions, the simulator supports both value and symbolic simulations. The reader is referred to [LOBF88] for further details concerning the simulation of LOTOS specifications.

As mentioned earlier, the theoretical basis for LOTOS are CCS and CSP, so CCS and CSP proof techniques can be applied (with little adaptation) to LOTOS. However, formal verification of anything but small examples depends on the availability of suitable software tools, and, because of the relative novelty of the language, few tools to do this are available. CAESAR/ALDEBARAN/ CLEOPATRE [Fetal] is a complex tool for performing a number of validation activities, such as proofs of weak bisimulation equivalence and model checking. Garavel [Gar91] discusses an experience where, by using this system, he verified a number of different systolic architectures to compute convolution products. [KBG92] present a system with similar functionality.

Testing is another area where formal methodologies are being developed for LOTOS. There exists quite a bibliography on the subject, one of the main references being [Br88]. Because of the fact that LOTOS's main application area has been protocol specification, the main problem envisaged has been testing equivalence between a protocol specification and its implementation. However, much of the theory is general, and can be applied to other testing problems such as testing whether a circuit implementation provides the behavior prescribed by the specification.

4. Conclusions and Research Directions

We have shown that LOTOS, a language initially designed for the formal description of communications protocols, is well suited for the description of digital systems. We have given small examples of both combinational and sequential circuits specifications.

An interesting research direction which might be a natural extension of this work is the construction of *silicon compilers* for LOTOS. A *silicon compiler* deals with the *automatic* generation of digital circuits from their behavioral descriptions. By using a silicon compiler, the designer can produce correct circuits because the resulting designs are *correct-by-transformation.* The subject of silicon compilers has been addressed by many researchers such as Brown and Leeser[BrLe89], and Martin[Mart89], to name just a few.

References

[Aylo86] J. H. Aylor, VHDL - Feature Description and Analysis, *IEEE Design & Test of Computers*, Vol. 3, No. 2, 54-65, april 1986.

[Br88] E. Brinksma, A Theory for the Derivation of Tests. In: S. Aggarwal and K. Sabnani. Protocol Specification, Testing, and Verification 8,. North-Holland, 1988, 63-74.

[BoBr87] B. Bolognesi and E. Brinksma, Introduction to the ISO Specification Language LOTOS. Computer Networks and ISDN Systems 14 (1987) 25-59.

[BrLe89] G. Brown and M. Leeser, From Programs to Transistors: Verifying Hardware Synthesis Tools, in LNCS 408, July 1989, 129-151.

[CaPr88] P. Camurati and Prinetto, Formal Verification of hardware correctness: Introduction and Survey of Current Research, *Computer,* 21(7), July 1988, 8-19.

[FLS91] M. Faci, L. Logrippo and B. Stepien, Formal Specifications of Telephones Systems in LOTOS: The Constraint-Oriented Style Approach, *Computer Networks & ISDN Systems*, Vol. 21, No. 1, 1991, 53-67.

[Fetal] J. C. Fernandez, H. Garavel, L. Mournier, A. Rosse, G. Rodriguez, J. Sifakis, A Toolbox for the Verification of LOTOS Programs, in L. A. Clarke, ed., Proc. of the 14 International Conference on Software Engineering (ICSE'14). ACM Press, New York.

[Gar91] H. Garavel, Compilation et verification de programmes LOTOS. PhD Thesis, University of Grenoble I, 1989.

[HL91] L. Logrippo and M. Haj-Hussein, Specifying Distributed Algorithms in LOTOS, Technical Report TR-91-04, Computer Science, Univ. of Ottawa, May 1991.

[Hoa85] Hoare, C.A.R., *Communicating Sequential Processes*, Prentice-Hall, 1985.

[KBG92] G. Karjoth, K. Binding and J. Gustafsson, LOEWE: A LOTOS Engineering Workbench. To appear in Computer Networks and ISDN Systems.

[LFH92] L. Logrippo, M. Faci and M. Haj-Hussein, An Introduction to LOTOS: Learning by Examples, *Computer Networks & ISDN Systems*, Vol. 23, No. 5, 1992, 325-342. Errata sheet in *Computer Networks & ISDN Systems*, Vol. 25, 1992, 99-100.

[LOBF88] L. Logrippo, A. Obaid, J.P. Briand and M.C. Fehri, An Interpreter for LOTOS, a Specification Language for Distributed Systems. Software-Practice and Experience, 18 (1988) 365-385.

[MM89] J. A. Mañas and T. de Miguel-More, From LOTOS to C. In: K.J.Turner (ed.) Formal Description Techniques North-Holland, 1989, 79-84.

[Mart89] A. Martin, The Design of a Delay-Insensitive Miroprocessor: An Example of Circuit Synthesis by Program Transformation, in LNCS 408, July 1989, 224-259.

[Miln91] G. J. Milne, The Formal Description and Verification of Hardware Timing, *IEEE Transactions on Computers,* Vol. 40, No. 7, 811-826, Jul. 1991.

[Mil80] R. Milner, A Calculus of Communicating Systems. *Lecture Notes in Computer Science* No.92 (Springer-Verlag) 1980.

[VSV88] C.A. Vissers, G. Scollo and M. van Sinderen, Architecture and Specification Style in Formal Descriptions of Distributed Systems. In Aggarwal, S., and Sabnani, K., Protocol Specification, Testing and Verification, VIII, North-Holland, 1988, 189-204.