Hardware Synthesis from a Restricted Class of LOTOS Expressions

Teruo Higashino, Keiichi Yasumoto, Junji Kitamichi and Kenichi Taniguchi

Department of Information and Computer Sciences,
Osaka University, Toyonaka, Osaka 560, Japan
Email : higashino@ics.es.osaka-u.ac.jp

Abstract

Recently, some studies for developing hardware circuits using LOTOS as a hardware description language have been proposed. In this paper, we introduce a LOTOS-like language called LOTOS/HD. Although LOTOS/HD can treat I/O parameters, LOTOS/HD expressions are functionally closed to Basic LOTOS expressions whose LTS's are finite. Then, we propose a technique for synthesizing hardware circuits semi-automatically from LOTOS/HD expressions. As the target circuits, synchronous sequential circuits are considered. In the proposed technique, first, the designers describe a specification *S* of a synchronous sequential circuit in LOTOS/HD where they only describe which values should be calculated and how such values are calculated using some functions implemented as combinational logic circuits. Next, from the data dependency relations and the temporal ordering of the events in *S*, a candidate *C* of sequential circuits implementing *S* is derived where *C* is also written in LOTOS/HD. The derived circuit *C* is a correct implementation of *S* if *C* satisfies some conditions. If *C* does not satisfy the conditions, the designers must modify *C* so that the conditions hold. The variables and functions in *C* are allocated to the registers and combinational logic circuits, respectively. Using a data path allocation technique, the connections between their components are decided automatically.

Keywords Codes : C.2.2, D.1.2, F.4.3
Keywords : Network Protocols, Automatic Programming, Formal Languages

1. Introduction

Many specifications of communication protocols and distributed systems have been described in LOTOS [ISO 89]. The systems implementing such specifications have been developed as the communication software. Since recently hardware technologies have been improved, such systems may be implemented as the hardware circuits. Recently, some studies for developing hardware circuits using LOTOS as a hardware description language have been proposed [FaLo 93, Turn 93a, Turn 93b]. In this paper, first, we introduce a LOTOS-like language called LOTOS/HD where LOTOS/HD expressions are functionally closed to Basic LOTOS expressions whose LTS's are finite although it treats I/O parameters. Then, we propose a technique for synthesizing hardware circuits semi-automatically from LOTOS/HD expressions. As the target circuits, synchronous sequential circuits are considered. In hardware research areas, many hardware description languages such as VHDL [IEEE 88], Verilog HDL [Veri 91] and SFL [Naka 87] have been used. Also many hardware synthesis techniques have been proposed (for survey, see [McPa 88]). One may think that those synthesis techniques can be applied for LOTOS expressions directly. However, in most cases, those hardware synthesis techniques use procedural languages as the hardware specification languages [DiPa 81, Naka 87, WaTh 89]. In such languages, the abstract level's specifications are described as procedural programs where the parallelism in the calculation is not considered explicitly in this level although such a parallelism is considered in the derived circuits. On the other hands, in LOTOS, the parallelism and interruption are very common in the abstract level's specifications. Therefore, the proposed hardware synthesis techniques based on the procedural languages cannot be applied directly for LOTOS expressions. Some specific ideas are needed.

In our approach, first, the designers describe a specification *S* of a synchronous sequential circuit in LOTOS/HD. In this level of abstraction, the designers do not consider its hardware architecture. They only describe which values should be calculated and how such values are calculated using some primitive functions which can be implemented as combinational logic circuits. Here, we say that a LOTOS/HD expression *C* is a correct implementation of another LOTOS/HD expression *S* if and only if the simulation relation given in Section 3 holds between *C* and *S* where *C* may be more deterministic than *S*. From the data dependency relations and the temporal ordering of the events in the abstract specification *S*, a candidate *C* of sequential circuits implementing *S* is obtained where *C* is also written in LOTOS/HD. The derived sequential circuit *C* is correct if *C* satisfies some conditions. If *C* does not satisfy the conditions, the designers must modify *C* so that the conditions hold. The variables and functions in *C* are allocated to the registers and combinational logic circuits, respectively. Using a data path allocation technique (for survey, see [TsSi 86]), the connections between the registers and combinational logic circuits are decided automatically.

The paper is structured as follows. In Section 2, we introduce LOTOS/HD and give an example hardware specification. In Section 3, we define the correctness of the implementation formally. A technique for deriving sequential circuits from given LOTOS/HD expressions is explained in Section 4. The conclusion and future works are discussed in Section 5.

2. Hardware Descriptions in LOTOS/HD

2.1 LOTOS/HD

In this section, we will introduce our LOTOS/HD language.

[Definition 2.1]

LOTOS expressions satisfying the following conditions are called LOTOS/HD expressions.

(1) If an observable gate is used as an input (output) gate, then the gate must not be used for an output (input) gate.

(2) An observable input event must be represented as g?x:type, i.e., the number of the variables x must be one.

(3) An observable output event must be represented as g!y where y is a defined variable.

(4) Each choice statement must be described in a form of ([f(y1,...,yn)] -> α) [] ([not(f(y1,...,yn))] -> β) where f(y1,...,yn) is a total function and the type of the function f(y1,...,yn) is Boolean. The choice statements without the guards are not permitted. This means that each choice must be deterministic.

(5) Each process must not invoke two processes in parallel. As the result, only one process is activated, i.e., if we treat a LOTOS/HD expression as a Basic LOTOS expression, then the size of the corresponding LTS is finite.

(6) Each unobservable event must be an input event whose form is g?y:type[y=f(x1,...,xn)] where y is a new variable and x1,...,xn are either defined variables or constants. In LOTOS/HD, only the function name and its type are defined and the axioms for f(x1,...,xn) are not specified.

(7) Each interruption must be described in a form of (α [> g?x:type ; β) where the gate name g must be observable and it must not be appeared in α. If an interruption (α [> g?x:type ; β) is contained in a process P, then the process P must not be invoked from the behavior expression β. That is, the number of possible interruptions must be finite. []

The conditions (1) to (4) represent the constraints in hardware circuits. In general, each I/O port is used either an input gate or an output gate in most hardware circuits. The condition (1) represents it. In most hardware circuits, the number of parameters which are input (output) from (to) each port is one. Such a restriction is described as the conditions (2) and (3). The condition (4) means that we do not treat non-deterministic behaviors. Here, we would like to derive circuits whose numbers of states are finite. The condition (5) is used for the purpose. The conditions (6) and (7) are used for simplifying our hardware synthesis. The details are explained in the following sections.

2.2 Example Specification in LOTOS/HD

In this section, we give an example specification in LOTOS/HD. For example, consider the LOTOS expression P(n:int) in Example 2.1. Here, we use a slightly simplified syntax and do not write the gate declarations in the process definitions.

[Example 2.1]

```
P(n:int) :=
    ( a?x:int ; a?y:int ; exit(x,y,any:int,any:int) ||| b?z:int ; b?w:int ; exit(any:int,any:int,z,w) )
    >> accept x:int, y:int, z:int, w:int in
       ( [ x+y+z+w=0 ] -> c!0 ; stop
       []
       [ not(x+y+z+w=0) ] -> d!((x*x-z*z)+(y*y-w*w))/(x+y+z+w) ; P(n+1) )          []
```

In Example 2.1, two integers x and y are given from the gate a. The other two integers z and w are also given from the gate b. Those event sequences are executed in parallel. If the sum of input data is zero, then the integer "0" is emitted to the gate c. Otherwise, the value of the expression ((x*x-z*z)+(y*y-w*w))/(x+y+z+w) is emitted to the gate d. Fig. 1 represents the relations between the gates and input data. In this specification, the designers do not describe how the output value is calculated. On general principles, we can construct a combinational logic circuit calculating the value of the expression ((x*x-z*z)+(y*y-w*w))/(x+y+z+w) for given four integers x, y, z and w. However, such a circuit is very complicated, and the cost for constructing the circuit may be expensive. Therefore, as a more general solution, we assume that we use the combinational logic circuits corresponding to the addition (+), subtraction (-), multiplication (*), division (/) and equality (=) for implementing the specification in Example 2.1. Under this assumption, for example, we can describe the specification in LOTOS/HD as follows.

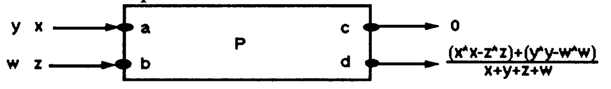


Fig. 1 Process P(n:int)

[Example 2.2]

```
P'(n:int) := hide g1,...,g9 in
    ( a?x:int ; a?y:int ; exit(x,y,any:int,any:int) ||| b?z:int ; b?w:int ; exit(any:int,any:int,z,w) )
    >> accept x:int, y:int, z:int, w:int in
       ( ( ( g1?x1:int[x1=x+z] ; g2?x2:int[x2=x-z] ; g3?x3:int[x3=x1*x2] ; exit(x3,any:int)
           ||| g4?y1:int[y1=y+w] ; g5?y2:int[y2=y-w] ; g6?y3:int[y3=y1*y2] ; exit(any:int,y3) )
               >> accept x3:int, y3:int in g7?z1:int[z1=x3+y3] ; exit(z1,any:int) )
       |[g1,g4]| g1?z2:int[z2=x+z] ; g4?z3:int[z3=y+w] ; g8?z4:int[z4=z2+z3] ; exit(any:int,z4) )
    >> accept z1:int, z4:int in
       ( [z4=0] -> c!0 ; stop
       []
       [not(z4=0)] -> g9?z5:int[z5=z1/z4] ; d!z5 ; P'(n+1) )                        []
```

We can easily show that the value of the variable z5 in Example 2.2 is equal to that of the expression ((x*x-z*z)+(y*y-w*w))/(x+y+z+w) in Example 2.1. In Example 2.2, the new gate names g1,...,g9 and new variables x1,..,x3, y1,..,y3, z1,...,z5 are introduced. We use the unobservable events g1,...,g9 only for defining the ordering of the calculation. The new variables are used for keeping the intermediate results for calculating the final result.

3. Correctness of Implementation

Here, we define the correctness of the implementation formally. One possible definition of the correctness is the weak bisimulation equivalence [Miln 89]. In general, the hardware specifications written in LOTOS/HD may be described using parallel operators. However, since we will derive synchronous sequential circuits, such circuits are more deterministic than given hardware specifications and the weak bisimulation equivalence does not hold in general. Therefore, we will introduce another relation called the simulation relation.

The simulation relation P *Imp* Q means that the process P may be more deterministic than the process Q, and that all executable event sequences of P must be also executable in Q. Formally, the simulation relation P *Imp* Q is defined as follows. Here, let Act denote the set of finite observable events. And let $B \overset{a}{\Rightarrow} B'$ denote $B\ (\overset{i}{\rightarrow})^* \overset{a}{\rightarrow} (\overset{i}{\rightarrow})^*\ B'$.

[Definition 3.1]

Let *Imp* denote a relation between two processes and suppose that P *Imp* Q holds. If the following condition (1) holds, then the relation P *Imp* Q is called the simulation relation.

(1) For any observable event a∈ Act, if $P \overset{a}{\Rightarrow} P'$ holds, then $\exists Q'[Q \overset{a}{\Rightarrow} Q'$ and P' *Imp* Q']. []

Next, we give a transformation Proj(P) from LOTOS (LOTOS/HD) expressions into Basic LOTOS expressions.

[Definition 3.2]

For a given LOTOS (LOTOS/HD) expression P, let P_G denote the LOTOS (LOTOS/HD) expression obtained by replacing all guarded expressions ([C]-> α) and ([not(C)]-> α) in P by the expressions (i;α). By ignoring all input/output variables and selection predicates in P_G, a Basic LOTOS expression is obtained. Here, we denote such a Basic LOTOS expression as Proj(P). []

For example, for the process P in Example 2.1, Proj(P) denotes :

Proj(P) := (a ; a ; exit ||| b ; b ; exit) >> (i ; c ; stop [] i ; d ; Proj(P))

Here, we define the correctness of the implementation using the simulation relation as follows.

[Definition 3.3]

Let *S* and *C* denote two LOTOS (LOTOS/HD) expressions. If the simulation relation Proj(*C*) *Imp* Proj(*S*) holds, then we say that *C* is a correct implementation of *S*. []

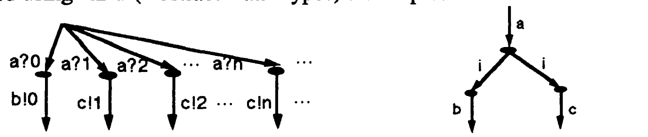
For example, for the two expressions P and P' in Examples 2.1 and 2.2, the simulation relation Proj(P') *Imp* Proj(P) holds. Therefore, P' is a correct implementation of P.

Here, we will explain why we use Proj(P) for defining the correctness of the implementation. Let's consider the following LOTOS expression R.

R := a?x:int ; ([x=0] -> b!x ; stop [] [not(x=0)] -> c!x ; stop)

For this LOTOS expression R, Fig. 2(a) represents the general LTS based on [ISO 89]. In this case, at the time when the input event a?x is executed, we have decided which event b!x or c!x is executed. This is a natural notion for general purposes. However, in hardware circuits, it is natural that such a selection should be carried out internally as an unobservable event after the input event a?x has been executed. The LTS for Proj(R) in Fig. 2(b) represents such a situation appropriately. Then, we use Proj(R) for defining the correctness of the implementation.

Note that the simulation relation *Imp* only discusses the ordering of the events and it does not consider the correctness of the correspondence of the I/O parameters. Such a correspondence should be discussed using ADT (Abstract Data Types) techniques.



(a) General LTS (b) LTS for Proj(R)

Fig. 2 Two Types of LTS's

4. Transformation into Sequential Circuits

In this section, we explain how to transform a given LOTOS/HD expression *S* into a synchronous sequential circuit *C* such that *C* is a correct implementation of *S*. For simplicity of discussion, in Sections 4.1 and 4.2, we assume that (1) we do not treat interruptions, and (2) there is only one process in a given LOTOS/HD expression. The expression P'(n:int) in Example 2.2 satisfies those conditions. We use this expression P'(n:int) as an example for explanation. A method for the cases without those assumptions is described in the end of Section 4.3.

4.1 Construction of Synchronous Sequential Machines

4.1.1 Data Dependency between Input/Output Variables

The temporal ordering of the events in a given LOTOS/HD specification must also hold in a constructed sequential machine. Here, the constraints concerning with the temporal ordering are given as integer linear inequalities. First, for each event, we will add the identifier representing its execution step in the constructed sequential machine. Each variable in an input event can be used as the identifier representing its execution step. Here, we assume that P_x denotes an identifier representing the step number in which an input event g?x is executed. For each output event, we will give an identifier so that we can distinguish each other. For example, in Example 2.2, there are two output events c!0 and d!z5. Here, we use, for example, P_c1 and P_d1 as the identifiers for c!0 and d!z5, respectively.

The constraints are given as follows. First, we give the constraints for observable events. If an observable event, say a?x, must execute before another observable event, say b?y, then we give the following constraint.

(O-1) P_x < P_y

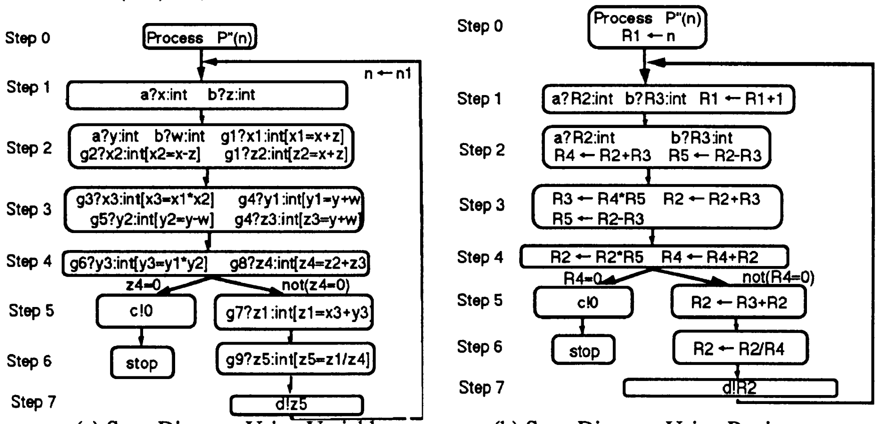
Next, for two unobservable events and for the pair of an observable event and an unobservable event, the constraints are specified from their data dependency relations. If an expression "g1?x[..] ; .. ; g2?y[..] ; g3?z[z=f(x,y)]; .." is given, and if the events g1, g2 and g3 are unobservable events, then we give the following constraints. These constraints mean that in order to calculate the value of the variable z, the values of the variables x and y must be executed before g3?z[z=f(x,y)] is executed.

(U-1) P_x < P_z (U-2) P_y < P_z

Note that P_x < P_y may not hold if there are no data dependency relations between the variables x and y.

Finally, we give the constraints between the synchronous parallel events. If an expression "... g1?x[..] ; ... |[G]| ... g1?w[..] ; ..." is given and the gate g1 belongs to G, then we give the following constraint. This constraint means that the two events g1?x and g1?w must be executed at the same step.

(S-1) P_x = P_w



(a) State Diagram Using Variables (b) State Diagram Using Registers
Fig. 3 State Diagram of Derived Synchronous Sequential Circuit

Let Max({x1,...,xn}) denote the maximum value in the set of variables x1,..,xn. Let *Cons*(P) denote the set of all identifiers used in a process P. If we can minimize the value of Max(*Cons*(P)), then the number of steps of the constructed sequential machine is also minimized. That is, if the minimum value of Max(*Cons*(P)) is k, then the LOTOS/HD expression P can be implemented by a sequential machine with k steps. Since all the constraints among *Cons*(P) are described as integer linear inequalities, the minimum solution k can be obtained by using a procedure for solving integer linear programming problems where Max(*Cons*(P)) is treated as the objective function.

In the hardware research areas, the decision of the concrete values for the variables in the set *Cons*(P) is called a scheduling [McPa 88]. The scheduling is a technique for deciding when we should execute each event. There are many possibilities. In the ASAP (As Soon As Possible) method, all executable events must be executed as soon as possible. That is, possible minimum values are assigned for the variables in the set *Cons*(P). In this paper, we use the ASAP method.

For the set *Cons*(P') of the process P' in Example 2.2, the minimum value of Max(*Cons*(P')) is 7. Therefore, the LOTOS/HD expression P'(n:int) can be implemented by a sequential machine with 7 steps. The synchronous sequential machine in Fig. 3(a) satisfies the constraints for *Cons*(P'), and it corresponds to one of the most efficient implementations.

4.1.2 LOTOS/HD Representation of Sequential Machines

Here, we define the semantics of the derived sequential machines in LOTOS/HD. We assume that the LOTOS/HD expression P"(n:int) in the following Example 4.1 corresponds to the sequential machine in Fig. 3(a) where the events executed in each step of the sequential machine are treated as the parallel events.

[Example 4.1]

```
P"(n:int) := hide g1,...,g9 in
      (a?x:int ; exit(x,..) ||| b?z:int ; exit(..,z,..) )
   >> accept x:int, y:int in
      (a?y:int ; exit(y,..) ||| b?w:int ; exit(..,w,..) ||| g2?x2[x2=x-z] ; exit(..,x2,..) |||
            ( g1?x1:int[x1=x+z] ; exit(..,x1,..) |[g1]| g1?z2:int[z2=x+z] ; exit(..,z2)) )
   >>  .........
   >> accept y3:int, z4:int in
      ( ([z4=0] -> ( c!0 ; exit ) >> ( stop ) )
      []
         ([not(z4=0)] -> ( g7?z1:int[z1=x3+y3] ; exit(z1) )
      >> accept z1:int in ( g9?z5:int[z5=z1/z4] ; exit(z5) )
      >> accept z5:int in ( d!z5 ; exit ) >> P"(n+1) ) )                    []
```

For the LOTOS/HD expressions P'(n:int) in Example 2.2 and P"(n:int) in Example 4.1, the simulation relation Proj(P"(n:int)) *Imp* Proj(P'(n:int)) holds. Therefore, P"(n:int) is a correct implementation of P'(n:int). The LOTOS/HD expression P"(n:int) is more deterministic than the LOTOS/HD expression P'(n:int) because the input event a?x:int must be executed before the input event b?w:int is executed in P"(n:int) although b?w:int may be executed before the input event a?x:int is executed in P'(n:int).

4.1.3 Conditions for Deriving Correct Sequential Machines

The sequential machines obtained by the method described in Section 4.1.1 may not be meaningful as the circuits.

Suppose that two input/output events at the same gate may be allocated to the same step. If the LOTOS/HD expression "(a?x:int ; ... ||| a?y:int ; ...)" is given, then the events a?x:int and a?y:int may be allocated to the same step. However, it is natural that, in the derived sequential machine, only one input is permitted for each gate at a step. Therefore, if such two input/output events are allocated to the same step, the designers must allocate them to different two steps.

If "(g?x:int[x=f1(..)] ; ... |[g]| ..g?y:int[y=f2(..)] ; ...)" is given and g?x and g?y are executed simultaneously, and if the functions f1(..) and f2(..) are not the same, then the deadlock may occur. For such a case, the designers must check whether there exist other possibilities of rendezvous by themselves.

If the state diagrams of the derived circuits are meaningful, then the derived circuits are correct. Since some heuristics may be used for constructing sequential machines, the designers must check whether Proj(*C*) *Imp* Proj(*S*) holds for the pair of a given LOTOS/HD expression *S* and the LOTOS/HD expression *C* corresponding to the obtained sequential machine.

4.2 Data Path Allocation

Here, we must allocate some registers for keeping the values of variables in a given LOTOS/HD expression. For simplicity of the circuit, the number of the registers should be minimized. Suppose that the value of a variable x is defined at the step i and that it is referred at the step j and it is not referred after the step j. Then we say that the life time of the variable x is [i,j-1]. If x is emitted as an output at the step j, then we say that the life time is [i,j]. If two variables x and y are used in rendezvous events, then we introduce the life time for the pair of the variables x and y. For example, in Fig. 3(a), the life time of the variables x1&z2 is [2,3] because the life times of x and y are [2,2] and [2,3], respectively. Fig. 4 represents the life times for all variables used in the sequential machine of Fig. 3(a). In Fig. 4, the life times of the variables x2 and y2 are not overlapped. The values of such variables may be kept in the same register. Fig. 5 represents a register allocation. In Fig. 5, only five registers are used for keeping the values of the variables in Fig. 4. It is known that such a register allocation is obtained by the technique called a clique-partitioning technique [GaJo79]. In Fig. 5, we use a special register R1 for keeping the process parameter. It is used when the process P" is invoked recursively.

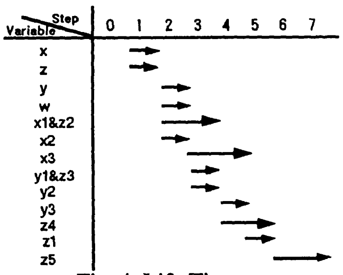


Fig. 4 Life Times

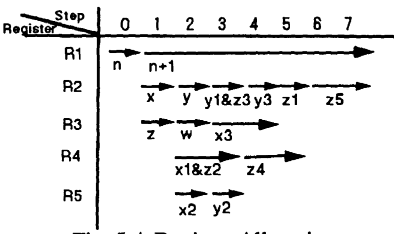


Fig. 5 A Register Allocation

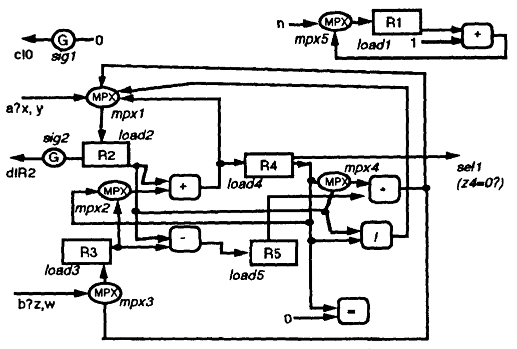


Fig. 6 Derived Circuit

4.3 Construction of Circuits

Here, we explain how to construct the circuit from a given data path allocation described above. We can construct a new state diagram using registers by replacing the variables by the allocated registers. In the new state diagram, each unobservable event, say g?x[x=f(y,z)], is also replaced by the substitution statement "Rx←f(Ry,Rz)" where Rx denotes the register keeping the value of "x". Fig. 3(b) is the replaced state diagram based on the register allocation in Fig. 5. For example, for the events a?y and g2?x2[x2=x+y] in Step 2 in Fig. 3(a), the corresponding actions are a?R2 and R5←R2-R3, respectively.

Next, we construct a concrete circuit from the state diagram using the allocated registers. If there is a substitution statement Ri←f(R1,...,Rn), then the outputs of the registers R1,...,Rn must be connected to the inputs of the combinational logic circuit calculating the value of the function f(...), respectively. If some of R1,...,Rn are constants, then the corresponding constants are given to the inputs. The output of the combinational logic circuit must be connected to the input of the register Ri. If the outputs from two or more combinational logic circuits must be connected to the input of a register Ri, then we use a multiplexer for selecting one of them. If the outputs from two or more registers must be connected to an input of a combinational logic circuit, then we also use a multiplexer for selecting one of them. We also use a gate for controlling whether an output should be emitted to each output gate.

Fig. 6 represents the derived circuit based on this method. In Fig. 6, the symbols MPX and G represent a multiplexer and a gate, respectively. Each multiplexer MPX selects one of the inputs depending on the value of its control signal *mpxi*. Each gate G emits its input as the output only if its gate signal *sigi* is 1. Each register Ri loads the input only if its load signal *loadi* is 1. For selecting one of the conditional branches of the sequential machine, one selection signal *sel1* is used. This selection signal emits 1 if and only if the value of the variable z4 is zero.

For executing this circuit, the control circuit must be also designed. That is, we must decide the values of the control signals when we execute the circuit. Using popular techniques, the control circuit and its micro program can be derived mechanically. In [Higa 94], we give the control circuit and its micro program for the above circuit.

In Sections 4.1 and 4.2, we assume that there is only one process. If two processes P and Q are used, then we construct the state diagrams for P and Q and connect them into one state diagram. From the connected state diagram, we can construct the concrete circuit. In Sections 4.1 and 4.2, we also do not treat interruptions. If an interruption (P [> Q) is given, then we construct the state diagrams for P and Q. During an interruption has not occurred, we execute the state diagram for P. If an interruption occurs, then we execute the state diagram for Q. Since the timing of the interruption is arbitrary, a control mechanism for changing from the state diagram for P to that for Q is needed. For the details, see [Higa 94].

5. Conclusion

In this paper, we have proposed a methodology for synthesizing hardware circuits from a restricted class of LOTOS expressions called LOTOS/HD. In the hardware research areas, such synthesis techniques have been studying actively. In the proposed technique, some restrictions are given. The further studies are needed for removing the restrictions. Now, we have been developing a tool for deriving the register transfer level's (RTL) circuits from given LOTOS/HD expressions based on our technique. For the transformation from the register transfer level's (RTL) descriptions into the IC level's real concrete circuits, we will use the synthesizer PARTHENON developed in NTT Corp., Japan, which is based on the hardware description language SFL [Naka 87]. One of the future works is to investigate the usefulness of our technique by using this tool.

References

[DiPa 81] S. W. Director, A. C. Parker, D. P. Siewiorek and D.E. Thomas : "A Design Methodology and Computer Aids for Digital Systems", IEEE Trans. on Circuits & Systems, 28, 7, pp.634-645, 1981.

[FaLo 93] M. Faci and L. Logrippo : "Specifying Hardware Systems in LOTOS", Proc. of Computer Hardware Description Languages and their Applications XI (CHDL'93), pp.305-312, North-Holland, 1993.

[GaJo79] M. R. Garey and D. S. Johnson : "Computers and Intractability", FreeMan, 1979.

[Higa 94] T. Higashino : "Synthesis of Sequential Circuits from a Restricted Class of LOTOS Expressions ", ICS Research Report, 94-ICS-5, Dept. I.C.S., Osaka Univ., 1994.

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

[ISO 89] ISO : "Information Processing System, Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour", IS 8807, 1989.

[McPa 88] M C. McFarland, A. C. Parker and R. Composano : "Tutorial on High-Level Synthesis", Proc. of 25th Design Automation Conf., pp.330-336, June 1988.

[Miln 89] R. Milner : "Communication and Concurrency", Prentice-Hall, 1989.

[Naka 87] Y. Nakamura : "An Integrated Logic Design Environment Based on Behavioral Description", IEEE Trans. on Computer-Aided Design Integrated Circuits & Systems, 6, 3, pp.322-336, 1987.

[TsSi 86] C. Tseng and D. P. Siewiorek : "Automated Synthesis of Data Paths in Digital Systems", IEEE Trans. on Computer-Aided Design Integrated Circuits & Systems, 5, 3, pp.379-395, 1986.

[Turn 93a] K. J. Turner : "An Engineering Approach to Formal Methods", Proc. 13th IFIP WG 6.1 Symp. on Protocol Specification, Testing and Verification (PSTV-XIII), North Holland, pp.357-380, 1993.

[Turn 93b] K. J. Turner and R. O. Sinnott: "DILL : Specifying Digital Logic in LOTOS", Proc. Sixth Int. Conf. on Formal Description Techniques (FORTE'93), North-Holland, 1993 (to appear).

[Veri 91] Open Verilog International : "Verilog Hardware Description Language Reference Manual", 1991.

[WaTh 89] R.A. Walker and D.E. Thomas : "Behavioral Transformation for Algorithmic Level IC Design", IEEE Trans. on Computer-Aided Design Integrated Circuits & Systems, 8, 10, 1989.