AN OBJECT-ORIENTED MODEL FOR RAPID PROTOTYPING OF DATA PATH/CONTROL SYSTEMS — A CASE STUDY

Ricardo J. Machado, João M. Fernandes, Alberto J. Proença

Dept. of Informatics, School of Engineering, University of Minho
Braga, Portugal

Abstract: The aim of this article is to present how to use shobi-PN, an Object-Oriented Petri Net model to specify data path/control systems. In a case study, a transputer link adaptor is specified for rapid-prototyping by applying the shobi-PN model and the generation of VHDL code for simulation is presented. The case study demonstrates the applicability and the capability of a development process, supported by the use of object-oriented principles to model both the control unit and the data path of complex systems, with a developed CAD tool.

Keywords: Object modelling techniques, Petri-nets, Computer control system design, Rapid programming.

1. INTRODUCTION

The design of complex digital systems is conceptually divided in two parts: the control unit and the data path. The behaviour of the control unit is usually described with an FSM, whilst the data path corresponds to the interconnection of several logic blocks, directly controlled by the actions from the controller. To specify digital systems, both the control unit and the data path should be addressed by the specification model and the CAD environment. The complexity of the design task grows when the controller behaviour presents concurrency, in which case several FSMs must work simultaneously with common signals to synchronize them.

Petri Nets (PNs) provide a graphical language to specify the behaviour of systems and are well suited to parallel digital controllers (Silva and Valette, 1990; Ferrarini and Maffezzoni, 1991). A set of formal methods can be used to verify the model's properties. Synchronous and Interpreted PNs (SIPNs) are a powerful alternative to the textual HDLs for specifying control units.

The shobi-PN model was developed as an extension to SIPNs, supporting hierarchy on the PN models and the use of objects to model the data path. A full digital system can be specified and tested with a structured and incremental approach. SOFHIA, a CAD environment that covers the main development phases, was developed to directly support the shobi-PN model. A VHDL file is generated for the modelled system, which allows the system to be simulated and synthesized.

2. THE SHOBI-PN MODEL

PNs are well suited for modelling and formal analysis of complex discrete systems (David and Alla, 1992), while VHDL is a standard hardware description language which exploits concurrency in the specification of digital systems, allowing their simulation and synthesis (Baker, 1993). PNs and VHDL may complement each other and they may also provide a proof system that accepts the same user interface descriptions for all design tasks (Schoen, 1992; Mermet, 1992).

Modelling PNs in VHDL was already presented in 1990 (Agarwal, 1990). A VHDL textual PN description of parallel controllers was later presented (Pardey and Bolton, 1991), which describes a VHDL template with `ASSERT` statements to enable the syntatic and semantic correctness of the model to be tested. Experimental results, developed at Inmos in a practical design, achieved a 50% area reduction and a 40% speed improvement over the best FSM synthesis. Another straightforward VHDL model of PNs, adequate for automatic generation by a translation tool, was later presented (Bergé *et al.*, 1992). The specification of PNs in terms of LSMs (Linked State Machines) and VHDL can be found, for example, in (Baker, 1993).

Another technique to convert a textual PN design representation into a VHDL description was introduced by Peng (Peng, 1992) within the CAMAD system. The input specification is written in ADDL (Algorithmic Design Description Language), a PASCAL subset, and supports both the control unit and the data path. An intermediate representation in ETPN (Extended Timed Petri Nets) model is generated, which includes: (1) the control unit modelled by Timed PNs (similar to SIPNs) with special firing rules, and (2) the data path modelled by dataflow diagrams. The synthesis of the digital system is completed in two steps: the data path is transformed into a netlist, while a microprogram is generated for the control unit. However, the PN is transformed into the classical FSM model: the advantages of concurrency are lost and there is no direct correspondence between the original PN and its implementation.

The PN modelling techniques for simulation were also introduced (Swaminathan *et al.*, 1994; Benders and Stevens, 1992), but related VLSI synthesis results were not reported. Other methods were also proposed for a reverse transformation: from specifications defined in VHDL to PNs for performance and reliability analysis. Some developments related to modelling and analysis were reported, for example, in (Eles *et al.*, 1994; Olcoz and Colom, 1995).

The only efficient way for PN modelling with VHDL, proven in industry, was introduced in (Pardey and Bolton, 1991) and it was followed in this work. The approach followed here for synchronous parallel controllers design suggests a particular way to implement a high-level PN specification of discrete systems in VHDL for later simulation and synthesis. It generates VHDL code and it supports hierarchical specifications.

A PN-based model, shobi-PN (Synchronous, Hierarchical, Object-Oriented and Interpreted Petri Net) (Machado, 1996), was developed to support the use of hierarchy and to include the control unit and the data path in the specification of digital systems. It was shown that the nucleous of the shobi-PN model can also be used as a kernel for distributed control systems (Pina *et al.*, 1997).

The shobi-PN model presents the same characteristics as the SIPN model (Fernandes *et al.*, 1995), in what concerns synchronism and interpretation, and adds new functionalities by supporting object-oriented modelling approaches and hierarchical mechanisms. As a consequence, this new model directly supports hierarchical structures in both the control unit and the data path. This model embodies concepts present in Synchronous PNs, Hierarchical PNs, Coloured PNs, and Object-Oriented PNs (Fehling, 1993; Lakos, 1995; Jensen, 1994).

The main characteristics of the shobi-PN model are the following: (1) The tokens represent objects that model data path resources. The instance variables represent the information of the data path and the methods are the interface between the control unit and the data path. The tokens may be considered as coloured, if SIPN tokens are viewed as uncoloured. (2) A node (a transition or a place) invokes the tokens' methods, when the tokens arrive at that node. Nonetheless, the only methods invoked are those that have a direct relation with the hardware control signals, since there are additional methods available at the objects' interface that are not used by the PN. These methods are invoked by the simulation software to visualize the contents of any structure of the data path in any state of the PN. (3) Each arc has one or more colours which associate the arc to the type of objects that are allowed to pass through it. This means that, for each data path resource, there is a well-defined path on the PN. This requirement simplifies the PN and limits the capacity of some places, since it is not needed that objects, that are not invoked, unnecessarily traverse the PN. (4) To implement the hierarchy on the model, macronodes (representing sub-PNs) and macrotokens (representing a hierarchy of tokens) may be used.

To understand some of the activities followed in the case study development in Section 4, the following concepts used for shobi-PNs must be introduced: (1) *control net*: set of contiguous nodes and arcs of the shobi-PN that structurally corresponds to an SIPN without reinitializations; (2) *control track*: path defined by a token in the control net; (3) *control nodes*: nodes (places or transitions) of the control net; (4) *control arcs*: arcs of the control net; (5) *closing track*: path defined by a token outside the control net; (6) *closing nodes*: nodes of a closing track; (7) *closing arcs*: arcs of a closing track; (8) *closing cycle*: path defined by the movement of a token in the shobi-PN. It is composed of a control track and also, if applicable, by a closing track. It can be identified by the tracking of the colour associated with all the arcs of the cycle; (9) *associated net*: SIPN structurally equivalent to the control net after the introduction of the reinitializations for the uncoloured tokens.

3. THE SOFHIA ENVIRONMENT

The SOFHIA CAD environment (Machado *et al.*, 1997*a*) is appropriate for specifying digital controller systems with the shobi-PN model (Fig. 1). At the moment, it feeds any ECAD package that accepts VHDL as input. The hierarchical shobi-PN specification is directly and efficiently mapped into boolean equations (Fernandes *et al.*, 1997). This approach simplifies the VHDL code debbuging, since there is a direct correspondence between the original shobi-PN and the code produced.

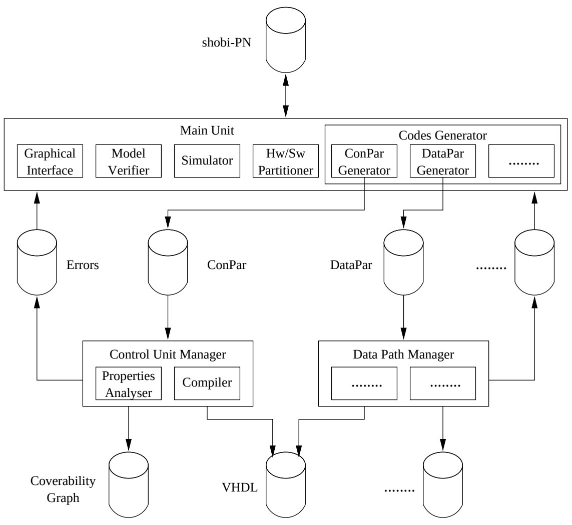


Fig. 1. The SOFHIA CAD environment.

When a system is built largely out of reusable components, designer's efforts can be focused on understanding the parts of the system that are new. The systems can be generated more quickly and easily. This leads to a new discipline known as "Rapid Prototyping", which is useful in cases where the aims and the requirements of the system are only vaguely understood at the beginnig of the project. PNs are very useful for rapid prototyping of digital systems due to their versatility in supporting both top-down and bottom-up approaches to modelling.

All the tasks needed for digital controllers rapid prototyping using shobi-PN-based specifications are supported by the SOFHIA environment. Among those tasks are: (1) automatic verification of the properties of the model; (2) simulation; and (3) code generation to implement the system. The verification of the properties of a hierarchical shobi-PN can be executed by using a method based on the principles presented by Lee and Favrel (Lee and Favrel, 1985; Lee and Favrel, 1987), as long as the following conditions are satisfied: (1) properties preservation whenever using the reduction rules to Synchronous PNs, and (2) building of a permanently-verified PN in relation to the properties to be tested except in the highest level, which is analysed by the Control Unit Manager's Properties Analyser.

4. CASE STUDY

The concepts introduced in the previous sections will be explored through a case study: the transputer link adaptor (Taylor, 1986), which interfaces a transputer network with its host bus system. A transputer network uses serial links to perform nearest-neighbour communication. To enable full duplex data flow, two wires are used for each link, and data transmitted along the outgoing wire is synchronized by acknowledgments from the receiving transputer on the incoming wire. Thus data, and acknowledgments for received data, are interleaved on each wire.

A data packet is transmitted as two start bits, followed by the data byte and the stop bit. The transmitter then waits for an acknowledgment packet, consisting of a start bit followed by a stop bit. To enable continuous data transfer, the receiver can transmit an acknowledgment packet as soon as an incoming data header is detected (provided it is not also transmitting a data packet at that time).

The communication between the transputer network and an external bus-based system requires serial-to-parallel and parallel-to-serial conversions to be performed at the network. The link adaptor shown in Fig. 2 provides a full duplex interface between a transputer link and two undirectional 8-bit buses. Data on the input bus (`I0-I7`) is multiplexed onto `LinkOut` whilst data on `LinkIn` is latched, via an 8-bit register, onto the output bus (`Q0-Q7`). A complete handshake protocol (`IValid,IAck,QValid,QAck`) controls data transfer to and from the buses.

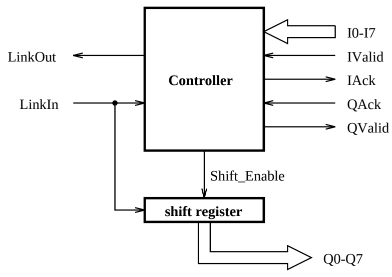


Fig. 2. The link adaptor.

In parallel-to-serial conversion, the bus driver deposits a data byte on signals `I0-I7` and raises `IValid`. The link adaptor packages the data, multiplexes it onto `LinkOut`, and looks for an acknowledgment packet on `LinkIn`. When this acknowledgment is received, the link adaptor relays it to the bus by raising `IAck`. The bus driver replies by lowering `IValid` and waits for the link adaptor to lower `IAck`.

In serial-to-parallel conversion, the transputer sends a data packet along `LinkIn`. On detecting the incoming data header, the link adaptor discards the packaging, latches the data byte on `Q0-Q7`, and asserts `QValid`. When `QAck` is raised in reply, the link adaptor relays it to the transputer as an acknowledgment packet on `LinkOut`. Then, the link adaptor lowers `QValid` and waits for the receiver to lower `QAck`.

Before starting to build the shobi-PN, the link adaptor's data path resources need to be modelled, by identifying the objects and defining its variables and methods.

After a careful analysis of the system statement, three different objects are identified to model the data path: the serial links, the data buses, and the shift register. According to this selection, the corresponding classes of the identified objects are declared and coded (in Fig. 3 only class *register* is presented as an example). Use of OMT's object model notation could also be considered for this activity.

The control unit is then specified with a shobi-PN, using instances of the previously defined classes. Each instance is invoked by methods existing in its interface during its travelling along the PN.

In this example, two instances of the *link* class (`Lin` and `Lout`), two instances of the *bus* class (`Bin` and `Bout`), and one instance of the *register* class (`Reg`) represent the serial links, the data buses and the shift register, respectivelly.

```
CLASSE: register
VAR. INST.: BOOL: bit[8], shift
METDS. INST.:
bool RD_BIT (BOOL lvl, INT idx)  void WR_BIT (BOOL lvl, INT idx)
{ if (lvl == HIGH)               { if (lvl == HIGH)
  then return (bit[idx])           then bit[idx] = HIGH
  else return (NOT bit[idx])       else bit[idx] = LOW
}                                }
bool RD_SHIFT (BOOL lvl)         void WR_SHIFT (BOOL lvl, val)
{ if (lvl == HIGH)               { if (lvl == LOW)
  then return (shift)              then shift = LOW
  else return (NOT shift)          else shift = HIGH
}                                        for (i=7;i=1;i--)
                                           bit[i] = bit[i-1]
                                         bit[0] = val
                                 }
```

Fig. 3. The class register.

The system controller of the link adaptor can be modelled by the shobi-PN in Fig. 4. The design, a revision of an earlier prototype (Pardey *et al.*, 1994), is a concurrent Mealy machine comprising 30 places and 35 transitions. Two macroplaces were introduced to specify the serial-to-parallel and parallel-to-serial conversions.

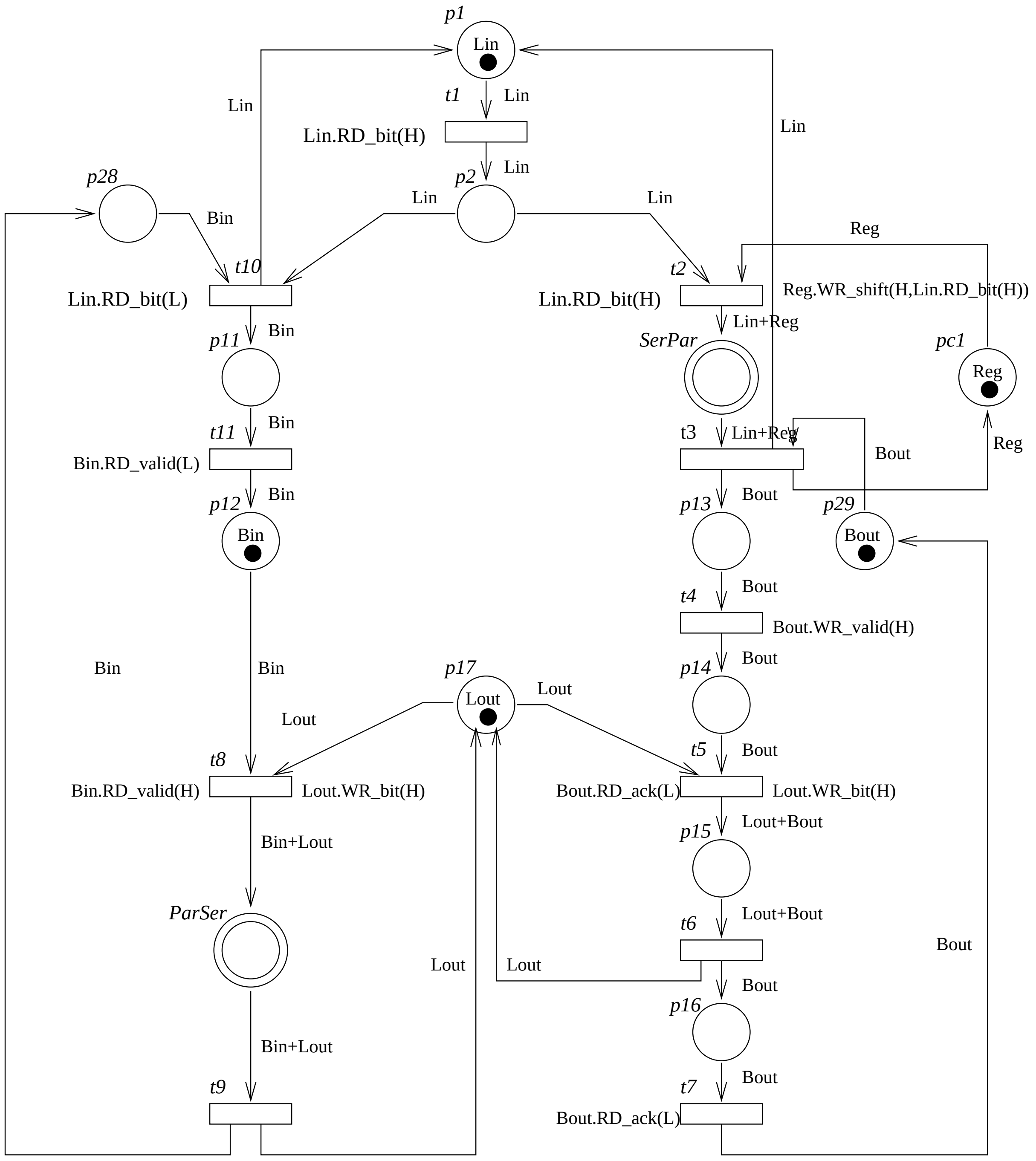
The PN can be roughly divided into four partially overlapped parts, being three of them complete closing cycles: (1) the path comprising (`p1,t1,p2,t2,SerPar,t3`) performs serial-to-parallel conversion and `LinkIn` acknowledge packet detection; (2) the closing cycle of `Bout` (`p29,t3,p13,t4,p14,t5,p15,t6,p16,t7`) generates the `QValid` output status signal, and waits for acknowledge signal on the input `QAck`, to complete the serial-to-parallel conversion; (3) the closing cycle of `Bin` (`p12,t8,ParSer,t9,p28,t10,p11,t11`) performs parallel-to-serial conversion and `LinkOut` acknowledge packet sending, with signals `IAck` and `IValid`; (4) the closing cycle of `Lout` (`p17,t8, ParSer,t9,t5,p15,t6`) manages the access to the `LinkOut` output serial port.

The only part that is not a complete closing cycle corresponds to a portion of the the closing cycle of `Lin`, since the other portion of the cycle (`p1,t1,p2,t10`) allows the switching for the parallel-to-serial conversion. The `Reg` object is the only one that possesses a closing cycle (`p1,t1,p2,t10`) with a control track and a closing track, since for the other objects of the system the respective closing cycles are only composed of control tracks.

The shobi-PN has no declarations of reinitializing nodes, so the control net is reinitialized by the initial marking of the control places (`p1,p12,p17,p29`) Additionally, to obtain the structure of the associated net (Fig. 5), it is necessary to remove the only existing closing place (`pc1`) and the arcs that connect it to the control transitions `t2` and `t3`. This place is only used for the data path simulation and it does not contain any information related to the control unit. For the interpretation of the associated net, the colours references must be deleted and the methods invokations must be transformed into hardware signals.

A data flow VHDL file is produced by the compiler module (Fig. 6). A compilation option was selected to direct the compiler to create a `BLOCK` statement, although a `PROCESS` statement could also be generated.

The `ASSERT` statements, automatically generated by the compiler tool, help the user in the system simulation. Those statements detect transitions in conflict and deadlock situations. If transitions `t3` and `t10` were simultaneously enabled, the place `p1` would be not safe. If both transitions were fired, output place `p1` would be marked twice, violating the desired PN safeness. Similar situations would occur for transitions `t6`, `t9` and transitions `t5`, `t8` with respect to place `p17`.


p1
Lin
t1
Lin
Lin
Lin
Lin.RD_bit(H)
p2
Lin
p28
Lin
Lin
Bin
Reg
t10
t2
Reg.WR_shift(H,Lin.RD_bit(H))
Lin.RD_bit(L)
Lin.RD_bit(H)
Lin+Reg
Bin
p11
SerPar
pc1
Reg
t11
Bin
t3
Lin+Reg
Bout
Reg
Bin.RD_valid(L)
Bin
p12
Bin
p13
Bout
p29
Bout
t4
Bout
Bout.WR_valid(H)
Bout
p14
Bin
Bin
p17
Lout
Lout
Lout
t8
t5
Bout
Bin.RD_valid(H)
Lout.WR_bit(H)
Bout.RD_ack(L)
Lout.WR_bit(H)
Bin+Lout
p15
Lout+Bout
ParSer
t6
Lout+Bout
Bout
Lout
Lout
Bout
p16
Bin+Lout
t9
t7
Bout
Bout.RD_ack(L)

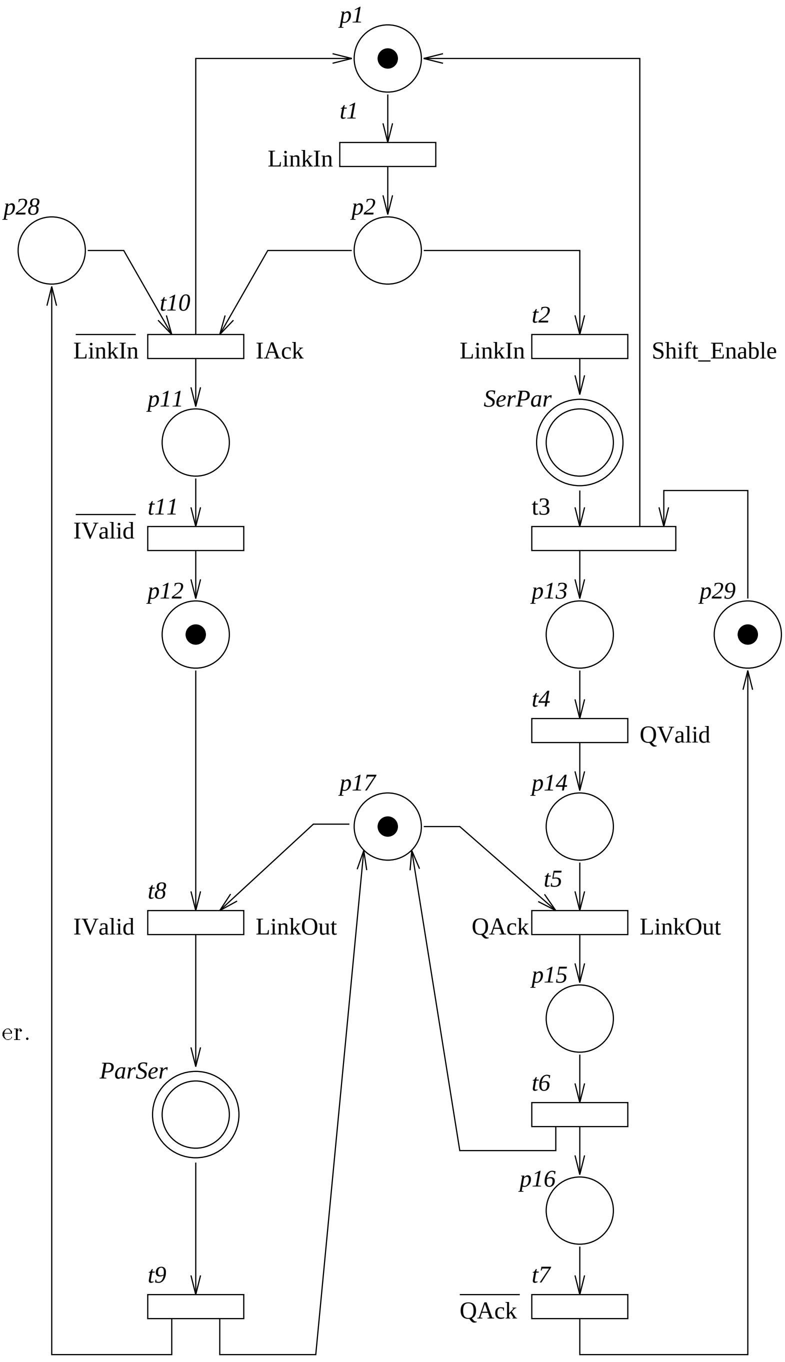


Fig. 4. shobi-PN for the link adaptor controller.

Fig. 5. SIPN for the link adaptor controller.

5. CONCLUSIONS

This paper shows that the shobi-PN model is an useful and efficient modelling tool to specify digital control systems. This model is the only known formalism using object-oriented PNs to specify both the parallel control unit and the data path in an integrated and modular way. The shobi-PN model presents synchronous behaviour, object-oriented modelling approaches, and hierarchical mechanisms. This model directly supports hierarchical structures in both the control unit and the data path, allowing the specification of digital parallel control systems in a modular, hierarchical and incremental way.

```
ENTITY controller IS
  PORT (reset, i0, i1, ... i7, linkin, qack, ivalid, clock : IN BIT;
        linkout, iack, qvalid, shiftenable : OUT BIT);
END controller;

ARCHITECTURE dataflow OF controller IS
-- Place Signals
  SIGNAL p1   : REG_BIT REGISTER;
  SIGNAL Np1 : BIT;
  ...
  SIGNAL serpar_p3  : REG_BIT REGISTER;
  SIGNAL Nserpar_p3 : BIT;
  ...
-- Transition Signals
  SIGNAL t1 : BIT;
  ...
  SIGNAL serpar_st7 : BIT;
  ...

BEGIN
  PART : BLOCK (clock=’1’ AND NOT clock’STABLE)
  BEGIN
    p1 <= GUARDED Np1 WHEN reset=’0’ ELSE ’1’;
    ...
    serpar_p3 <= GUARDED Nserpar_p3 WHEN reset=’0’ ELSE ’0’;
    ...
  END BLOCK;
-- Dataflow description for transitions
  t1 <= NOT p2 AND linkin AND p1;
  ...
  serpar_st7 <= serpar_p9 AND NOT serpar_p10;
  ...
-- Dataflow description for next place markings
  Np1 <= t10 OR t3 OR (p1 AND NOT t1);
  ...
  Nserpar_p3 <= t2 OR (serpar_p3 AND NOT serpar_st1);
  ...
-- Output Signals Equations
  qvalid <= t4;
  shiftenable <= t2 OR serpar_st1 OR ... OR serpar_st7;
  ...
---- ASSERT Commands
-- Transitions in conflict
  ASSERT NOT (t10 AND t3)
    REPORT "output place p1 overflows (transitions t10,t3)"
    SEVERITY ERROR;
  ...
END dataflow;
```

Fig. 6. Generated data flow VHDL code for the link adaptor controller.

The analysis of some case studies (Machado *et al.*, 1997*b*) —included in this communication is the Transputer Link Adaptor (TLA)— also shows that there is a relation between the structure of the PNs and the kind of approach followed in the system specification. In the TLA example, the obtained PN reflects a data-driven approach, because there is only a closing track. This kind of nets embodies in the control net the reinitializations of the objects with no need to partially or totally incorporate the closing tracks. The data-driven control nets can be totally closed (i.e. they do not have reinitializing nodes). On the other hand, the control-driven control nets must be opened. This analysis shows that the shobi-PN directly follows a data-driven approach in the specification of parallel digital systems, and it properly supports the specification of both the control unit and the data path.

6. REFERENCES

Agarwal, S. (1990). Thinking Petri Nets Through VHDL. In: *1990 VHDL Fall User's Group Meeting.* Oakland, USA. pp. 51–9.

Baker, L. (1993). *VHDL Programming with Advanced Topics.* John Wiley & Sons.

Benders, L.P. and M.P. Stevens (1992). Petri Net Modelling in Embedded System Design. In: *CompEuro 1992: Computer Systems and Software Engineering* (P. Dewilde and J. Vandewalle. (Ed)). pp. 612–7.

Bergé, J.-M., A. Fonkoua, S. Maginot and J. Rouillard (1992). *VHDL Designer's Reference.* Chap. 5: System Modeling, pp. 129–45. Kluwer Academic Publishers. Dordrecht.

David, R. and H. Alla (1992). *Petri Nets & Grafcet; Tools for modelling discrete event systems.* Prentice-Hall International. UK.

Eles, P., K. Kuchcinski, Z. Peng and M. Minea (1994). Synthesis of VHDL Concurrent Processes. In: *Euro-DAC'94, European Design Automation Conference with Euro-VHDL'94.* Grenoble, France. pp. 540–5.

Fehling, R. (1993). A Concept of Hierarchical Petri Nets with Building Blocks. In: *Advances in Petri Nets 1993* (G. Rozenberg. (Ed)). Vol. 674 of *Lecture Notes in Computer Science.* pp. 148–68. Springer-Verlag.

Fernandes, J.M., A.M. Pina and A.J. Proença (1995). Concurrent Execution of Petri Nets based on Agents. In: *1st Workshop on Object-Oriented Programming and Models of Concurrency within the 16th International Conference on Applications and Theory of Petri Nets.* Torino, Italy.

Fernandes, J.M., M. Adamski and A.J. Proença (1997). VHDL Generation from Hierarchical Petri Net Specifications of Parallel Controller. *IEE Proceedings: Computers and Digital Techniques*, **144**(2), 127–37.

Ferrarini, L. and C. Maffezzoni (1991). Designing Logic Controllers with Petri Nets. In: *5th IFAC Symp. Computer-Aided Design in Control Systems*, (H.A. Baker (Ed)). pp. 319–24.

Jensen, K. (1994). An Introduction to the Theoretical Aspects of Coloured Petri Nets. Technical report. Comp. Science Dept, Aarhus University, Denmark.

Lakos, C. (1995). The Object Orientation in Object Petri Nets. In: *1st Workshop on Object-Oriented Programming and Models of Concurrency within the 16th International Conference on Applications and Theory of Petri Nets.* Torino, Italy.

Lee, K.-H. and J. Favrel (1985). Hierarchical Reduction Method for Analysis and Decomposition of Petri Nets. *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-15**(2), 272–80.

Lee, K.-H. and J. Favrel (1987). Generalized Petri Net Reduction Method. *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-17**(2), 297–303.

Machado, R.J., J.M. Fernandes and A.J. Proença (1997*a*). SOFHIA: A CAD Environment to Design Digital Control Systems. In: *XIII IFIP Conference on Computer Hardware Description Languages and Their Applications (CHDL'97)*, (C.D. Kloos and E. Cerny (Ed)). pp. 86–8. Chapman & Hall, Toledo, Spain.

Machado, R.J., J.M. Fernandes and A.J. Proença (1997*b*). Specification of Industrial Digital Controllers with Object-Oriented Petri Nets. In: *IEEE International Symposium on Industrial Electronics (ISIE'97).* vol. 1, pp. 78–83. Guimarães, Portugal.

Machado, R.J. (1996). Hierarchy in Object-Oriented Petri Nets for the Specification of Digital Systems. MSc. thesis. Dep. Informatics, University of Minho. Braga, Portugal. (in Portuguese).

Mermet, J. (Ed) (1992). *VHDL for Simulation, Synthesis and Formal Proofs of Hardware.* Kluwer Academic Publishers. Dordrecht.

Olcoz, S. and J.M. Colom (1995). A Colored Petri Net Model of VHDL. *Formal Methods in System Design*, **7**(1/2), 101–23.

Pardey, J., A. Amroun, M. Bolton and M. Adamski (1994). Parallel Controller Synthesis for Programmable Logic Devices. *Microprocessors and Microsystems*, **18**(8), 451–8.

Pardey, J. and M. Bolton (1991). Logic Synthesis of Synchronous Parallel Controllers. *Proceedings of the IEEE International Conference on Computer Design*, pp. 454–7.

Peng, Z. (1992). Digital System Simulation with VHDL in a High-level Synthesis System. *Microprocessing and Microprogramming*, **35**, 263–70.

Pina, A.M., J.M. Fernandes and R.J. Machado (1997). Genetic regulatory mechanisms by means of Extended Interactive Petri Nets. In: *IEEE International Conference on Systems, Man, and Cybernetics (SMC'97).* Hyatt Orlando, Orlando, Florida, USA.

Schoen, J.M. (Ed) (1992). *Performance and Fault Modeling with VHDL.* Chap. 3.3.2: Petri Net Models. Prentice Hall. Englewood Cliffs, USA.

Silva, M. and R. Valette (1990). Petri Nets and Flexible Manufacturing. In: *Advances in Petri Nets 89* (G. Rozenberg (Ed)). Vol. 424 of *Lecture Notes in Computer Science.* Springer-Verlag, Berlin, Germany. pp. 376–417.

Swaminathan, G., R. Rao, J.H. Aylor and B.W. Johnson (1994). A VHDL based Environment for System Level Design and Analysis. In: *VHDL International Users Forum (Spring Conference).* pp. 110–6.

Taylor, R. (1986). Transputer Communication Link. *Microprocessors and Microsystems*, **10**(4), 211–5.