VLSI Implementation of Controllers for Communication Protocols from their Petri Net Models

Asjad M. T. Khan Sadiq M. Sait Gerhard F. Beckhoff

King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia

Abstract

Petri nets are popular in the communication protocol community for modelling and analysis purposes. This paper gives a procedure for extraction and implementation of controllers of protocols from their Petri net model. It adds another dimension to the use of Petri nets and is suited for Network, Transport and Data link protocols.

1 Introduction

Advances in fiber optics have resulted in increased communication speeds. The links can now operate at speeds of several Gigabits per second. The trend towards ISDN and Multimedia systems require high speed traffic handling. All this severely limits the communication processing by software on general purpose computers [1]. With these things in mind handling of communication processing by a dedicated hardware is desirable. This can be done by relatively inexpensive (as compared to the processing power and time of the host computer) VLSI circuit. A *Protocol Processor* is typically a coprocessor VLSI chip or chip set that relieves the host computer from communication processing functions. A *Protocol Controller* is a VLSI circuit that implements the functions of one or more protocol layers.

Petri nets are suitable for fields which have distributed or parallel components. Communication protocols are an instance of this class. They are one of the two most successful areas of application of Petri nets, the other being performance evaluation [2]. The Australian and the French telecommunication industries are using this model on a large scale although the United States uses a language based specification. The protocol specification and verification community were one of the first ones to experiment with Petri net modelling. The protocol in question is modelled by a Petri net and then this model is analyzed for desirable properties. If some error is found in the behavior of the protocol then the model is changed and this cycle is continued till the protocol exhibits desired behavior. Once the protocol is found to be correct it is implemented in software.

Having seen that Petri nets have been used to model protocols and the fact that hardware implementations of protocols exist, it is natural to ask *whether the protocols can be directly implemented from their Petri net model and whether this process could be automated.* This work proposes to extend this modelling –analysis cycle to include implementation in hardware so that the entire design cycle, from specification to realization can be done using Petri net model. This has the obvious advantage of eliminating translation from one specification to another and the possible errors associated with it. Further a hardware realization has the advantage of higher speed over software. Work on translation of Petri net model to high level programming language exists [3].

The terms and notations used for Petri nets are taken from the survey-cum-tutorial by Murata [2].

2 Previous Work

Protocol controllers are a relatively new area. A survey on the existing and proposed implementations appeared in the literature [1]. Several implementations of the Data Link layer protocols exist, [4] implements the complete layer. In [5] implementation of network layer is found. Attempts have also been made to implement Transport layer protocols too [6]. However, most of the work is restricted to Data Link, Network and Transport layer protocols.

Petri nets provide a formal method for modelling asynchronous systems. Initial approaches to Hardware synthesis from Petri nets models were directed towards asynchronous implementation and one of the areas of application was design of fast control circuits for computers. Patil and Furtek proposed modules for direct implementation of certain classes of Petri nets [7,8] However, no efficient and systematic method has been proposed [9]. A recent work postulates that Petri nets may not be a good method for synthesis of asynchronous circuits because states (markings) of a Petri nets do not have a direct relationship with those of the circuit (normally defined as vector of signals) [10] A new technique called state transition graph has been proposed for this purpose in [9].

Synchronous implementations have also been attempted. Kwan et al. [11] and Auguin et.al [12] proposed methods for decomposition and implementation of a class of Petri nets by PLAs. However they did not

0-8186-2610-0/92 $3.00 © 1992 IEEE

consider distributed systems and the algorithm for decomposition was restricted. In this paper we consider synchronous implementation.

3 Modelling of Communication Protocols

3.1 Communication Protocols

The OSI reference model for computer networks has a layered architecture. Each layer provides a certain service to the layer above it and shields it from the actual implementation details of the service. A layer provides service to the one above it using the services of its lower layer. The implementation of a layer on a machine is called an *entity*. The entities comprising the corresponding layers on different machines are called *Peer Processes*. The rules and conventions used in conversation between peer processes is termed as the *Protocol* for that layer. A protocol is based on message exchange, this includes synchronization messages between entities that wish to communicate before they can handle data and the message transfer that occurs during data exchange.

3.2 Protocol Specification

The specification for a layer involves *service specification*, *interface specification* and the *protocol specification* [13]. From the point of view of layers above it, the input-output behavior of the layer constitutes a service specification. It is an abstract black box description. It is based on a set of abstract service primitives which describe the input/output functions e. g. for data link services basic primitives are send, receive etc. These service primitives are executed in a certain order which is given in the specification. The order may depend on constraints due to operations by the same user (local) or other user (global). The execution of service primitives involve exchange of parameter values between the adjacent (vertically) entities. The exact format of how the services are provided is defined by the Interface specification which could be different for different users.

Although interaction occurs between adjacent layers (vertically) the basic concept behind the layered architecture and the design of layers is that each entity assumes that it is dealing with it's peer. In the network with physically separated users the protocol layer is distributed and entities local to each user communicate with one another via services of lower layers. The rules for interaction between entities in providing the layer's service constitutes the layer's actual protocol. The term communication protocol, as commonly used, refers to this protocol. Figure 1 shows a symbolic representation of a protocol. The protocol specification is basically a refinement or distributed implementation of the service specification.

Three techniques have been used to model protocols, finite state models, language based approaches and a hybrid of the two. The finite state models include Finite State Machines (FSM), directed graphs, Petri nets etc. and the programming language approach includes special languages designed for parallel/distributed systems including *Estelle* , *Lotos* etc. The transitional (Finite state) models are based on the fact that protocols consist of simple processing in response to occurrence of *events* such as commands from the user, message arrival from lower layer) and internal events (such as timeouts). The general approach in modelling protocols is to consider the protocol as a distributed system (which it is) [14]. Each of the site can be represented by a process and the overall model can be represented by individual processes with inputs and outputs which communicate or interact with each other. The model then reduces to one of communicating processes. The entire network can be studied with this modular multilevel approach [14].

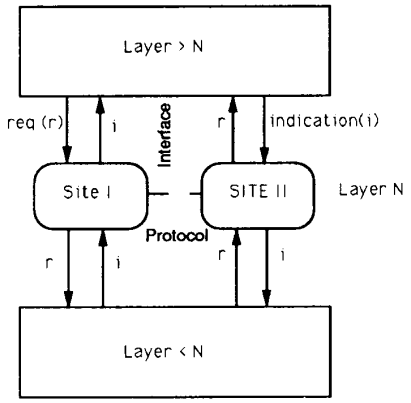

Figure 1: The Protocol Specification

Using this approach each site processor can be modelled as a FSM. The inputs of site a, FSM are the outputs of the channel FSM and its outputs are inputs to the channel FSM. The same holds true between FSM of site b, and the channel. However some of the events occurring in a process might be internal to it. These events occur outside the specification of the protocol but are related to the correct behavior. They in a way imply that not only the protocol but some detail of the process implementing it is also modelled. These internal events occur without any external input and may or may not produce an output. In the FSM model they can be represented by the always true input ϵ.

3.2.1 Petri Net Model

Petri nets evolved as mechanisms for interprocess communication. They can model interaction between different components. It is this which makes Petri nets useful in protocol modelling. The Petri net model of protocols is derived in two steps [14], at first each entity and the channel is modelled as a state machine or a net and then the components are connected together by explicit interconnection mechanism corresponding to the actual implementation to give the global system model. In the approach given in [15],

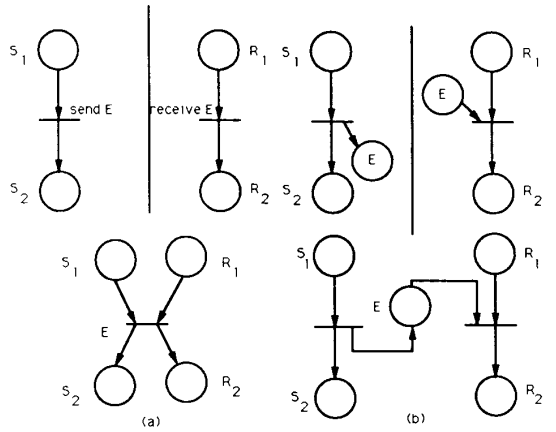

Figure 2: The global model obtained using (a) Shared place and (b) merged transition mechanism

each entity and the medium is modelled as a FSM and then a Petri net model is built by connecting the machines together. For simple models of communication medium a number of interconnection mechanisms have been proposed [14]. The basic ones are given below: **Shared Place**: A place shared between the sender and the receiver processes represents the medium. It represents the fact that the message has been sent by the sender and not yet reached by the receiver. This is really the actual behavior denoting the fact that the message is in transit in the medium. It has been the most used interconnection mechanism [14]. A shared place representation implies potentially unbounded buffering, the sender is released after sending the message.

Merged Transition: In this technique the sender and the receiver processes share a transition to exchange a message. This implies no buffering, direct transfer and a synchronized send/receive. The shared place and merged transition mechanisms are illustrated in Figure 2.

The shared place is the basic interconnection mechanism for processes belonging to distinct entities. More advanced models of medium have been considered. These include a FIFO model. These have been modelled using Predicate-Transition nets and Predicate-Action nets. However, in the opinion of Diaz [14], as a first step, protocols should be validated with the shared place model for every exchanged message as the corresponding graph model imposes very few constraints on the medium. This approach has been widely used [16]. As a next step, specific medium can be accounted for.

It will be seen in the next section (on implementation) how this information about modelling helps in implementation. If the abstract model of interconnection is not known then it has to be derived from the design.

4 VLSI Synthesis of Protocol Controller

The implementation is in the form of a PLA which is suitable for VLSI. This method can be automated to realize a Petri net based DA system. The implementation method can be logically divided into two phases – *Frontend*, it takes the Petri net model of the protocol and produces a FSM for each of the controller entities and the *Backend* which generates a PLA implementation of the FSM.

4.1 Frontend

This phase consists of two basic steps, decomposition of the protocol model into two entities representing the N^{th} layer and for each entity, obtaining the FSM for it's implementation.

4.1.1 Decomposition of Protocol

Both the shared place and the merged transition mechanisms can be shown to be equivalent in terms of liveness and boundedness properties using the reduction rules by Kwang et al. [17]. Although both representations are equivalent, in a sense, using both simultaneously is not recommended as it is difficult to handle them semantically and theoretically [18]. This is understandable from the point of view of the interconnection service they provide. In terms of protocols, it has been suggested that shared place be used for modelling the protocol of a layer (the interaction of distributed entity) and that merged transition be used to model interface protocols (or, equivalently, interfacing of protocol levels) [14].

The implementation of the protocol from the Petri net model can be integrated into the design-analysis cycle. The Petri net model is built using the structured approach of modelling each entity by an FSM and the channel is represented by shared places or merged transitions. The overall system can then be analyzed and when the design is found to be correct the task of implementation can be initiated. With this scenario, it is realistic to assume that the composite protocol model is known along with the information about the models of the entities i. e. the model with entities demarcated is known. This corresponds to the existing approach in literature where the overall model is given demarcated [19,20]. The task of automated implementation then consists of detection of channel and building a FSM for each component. With the knowledge of entities, the task of channel detection reduces to finding the common places or transitions.

Procedure 1 (Break_Entities)

Input The Petri net model of the protocol with each entity demarcated in the protocol

Output Petri net model of each isolated entity

1. **if** shared place mechanism is used **then** Search for places which are common to both entities

2. **for** each entity, **for** each shared place, **for** each arc connected to shared places

3. **if** the arc is connected from the transition to the shared place, x (output arc) **then** assign an output, x to the transition, (x) and cut the arc connecting the transition to shared place

4. **if** the arc is connected from the shared place to the transition, x (input arc) **then** assign an input condition, x to the transition,$< x >$ and cut the arc connecting the shared place to the transition

5. **if** merged transition is used **then** Search for transitions that are common to both entities

6. **for** each merged transition, **for** for each entity, duplicate the transition; **for** each arc connected to merged transition, **if** the arc connected to the merged transition belongs to the other entity, **then**

7. **if** it is an input arc and it comes from place v **then** assign an input condition, $< v >$ to the transition and cut the arc

8. **if** is an output arc coming from place w **then**

9. assign an output, (w) to the transition and cut the arc

By the application of this procedure, the Petri net is separated into components representing each entity. The Petri net obtained is annotated with input conditions and outputs associated with the transitions. This is not different from the Petri nets defined earlier. They just represent a modelling convenience. The input condition on the transition is equivalent to a source place connected to the transition and models the interface to the external world.

4.1.2 FSM Extraction

The process of obtaining the FSM from the annotated Petri net has two steps: first, refinement of the entity and next generating a FSM for the underlying unannotated Petri net.

Refinement: As discussed in the section on modelling, each modelled entity contains some local actions which occur within the entity itself. These do not involve any communication with the peer but are related to the correct operation of the protocol (e. g. timer in a data link protocol). Moreover, there is one more type of signals coming to the controller, from the upper layer. As the interface and service specifications are usually treated separately from the protocol specification this can be handled by a separate Interface controller. One way of handling refinement for local actions is to associate labels to transitions signifying local actions. In the protocol analysis stage these labels are ignored and no interpretation is associated, however, in the implementation stage these are interpreted as input conditions and outputs. Alternatively, this can be done at this stage (after entities have been separated). In the Petri net model of protocol PAR presented Figure 3 the place transition label 'timout' models a local action.

FSM from annotated Petri net: The reachability graph of the underlying unannotated Petri net is isomorphic to an FSA which accepts the language of firing sequences if the graph is bounded [21]. The conditions and the output are used then to replace the transitions labelling the arcs of the transition. This then defines a FSM. Two theorems are presented to support this assertion.

Theorem 1 For every isolated entity a FSM can be obtained.

Outline of Proof: The proof is based on two previous results and the property of protocols [22]. It can be assumed that the protocol to be implemented has been verified to be correct. The protocol design rules require that the protocol is live and bounded [23].

Using the result of [24] on partitioning of Petri nets, it can be shown the entities will also be live and bounded. Having shown that the subnets representing entities are live and bounded, a result from Valk [21] can be used to show that for every regular net, bounded being a proper subclass of regular nets, the reachability graph defines a finite automaton.

Theorem 2 (VALK) *There is an effective procedure that associates to every regular Petri net N, a finite automaton accepting F(N), defined as,*

$$F(N) = \{w \in T^{\star} \mid M_0(w >\}$$

Using this result from Valk [21] and the fact that each entity subnet is bounded a FSA, can be obtained. In the obtained FSA the arcs are labelled by transitions. These are then replaced by input conditions and outputs associated with the transitions. The procedure of obtaining FSM from the annotated Petri net is summarized below:

Procedure 1 1. **begin**

2. **for** each entity

3. find the reachability graph of underlying unannotated Petri net,RG=(V,E), where, Nodes of the graph, V=R(M_0) and the set of labelled, directed arcs, E has an arc from V' to V'' labelled by t if for the corresponding markings, M' and M'', $M'[t > M''$

4. **do** for the reachability graph

5. **for** every arc

6. substitute the transition labelling the transition, t by the input conditions, $x, y, \ldots$ and outputs, $a, b, \ldots$ associated with the transition as, $< x, y, \ldots > /(a, b, \ldots)$, where, all the conditions $x, y, \ldots$ should be satisfied for the transition to occur and the change produces outputs $a, b, \ldots$. This is finite and isomorphous to a FSM.

4.1.3 Implementation Aspects

The basic tool for implementation of frontend was P-NUT (Petri Net UTilities) developed at the University of California at Irvine.

4.2 Backend

The backend is basically a silicon compiler for FSM. It takes a description of the state machine to be implemented and gives the layout of the PLA implementing it. It is based on a subset of tools made at the University of California, Berkeley, called **1986 VLSI Tool Kit** [25] and **VLSI Design Tools** [26]. The entire process is automated. The first of the tools **meg** (Mealy Equation Generator) takes a high level input of the FSM and outputs the equations of the output and next_state variables in terms of present_state and input variables. This is then fed to **eqntott** which outputs the truth table for implementation by a two level circuit(PLA personality). This format is compatible with the **espresso** input which is used to minimize the function. The output can then be fed to a PLA generator, **mpla** which outputs the layout in the desired technology and style. What remains after that is to place the PLA in a pad frame, connect the input of FSM and the next state outputs, route the inputs and outputs to I/O ports and the chip would be ready to be sent for fabrication. These tasks can also be done using available tools. Finally, the FSM can be extracted from the PLA and simulated to verify correctness of operation. The FSM is extracted using circuit extractor, **mextra** and simulated using **rnl**. The simulation output is viewed using another tool, **simscope**.

Example 1 Certain aspects of the synthesis procedure are explained with the help of an illustrative example. The Petri net model of a of a simplex alternating bit protocol, Positive Acknowledgment Retransmission, PAR is given in Figure 3. The places and the transitions have been named according to the state/action to facilitate understanding. There are two processes, a sender and a receiver. The communication between these is modelled by the channel shown. The sender numbers the packets it sends by a 0 or a 1. In the initial state, it sends packet_0 and waits for the acknowledgment (abbreviated as ack) to arrive from the receiver. If everything goes right, an ack will arrive. It would then transmit packet_1 and wait for the ack. However two things can go wrong.

1. The message may get lost. This simply means that an error occurred in some part of the packet received by the receiver, and so no ack was sent. After a fixed amount of time (determined by various factors including the distance between sender and receiver) the sender times out and retransmits packet_0.

2. The msg was received but the acknowledgment sent by the receiver got lost. The sender does not know what really happened and retransmits packet_0. The receiver however is waiting for packet_1. It rejects this packet and sends an ack. When this acknowledgment is received by the sender it transmits packet_1 and a sequence similar to the above is repeated.

Figures 4, 5, 6, 7 show the major steps in realization and verification of the protocol.

5 Summary and Conclusions

The paper presents preliminary results. A method for synthesizing communication protocols was presented. It was assumed that the composite Petri net model of the protocol is available. This is a realistic assumption if this process is integrated in the design cycle and it agrees with the current approach in modelling protocols. The problem of decomposition of Petri nets into interacting components is not trivial. In general, it is not possible to decompose a Petri net into cooperating state machines. If it is assumed that only the composite model is known then it is difficult to decompose the net [22]. However, if certain restrictions are made on the model then the task is easier. The option of using a subclass is investigated in another work [27].

Acknowledgments

Thanks are due to Dr. T. Morgan for providing P-NUT, the University of Washington for the VLSI Design Tools and Dr. M. S. T. Benten for his commitment to excellent computing facilities. The first author also wishes to thank Dr. S. Ghanta for introducing him to Petri nets and King Fahd University of Petroleum and Minerals, Dhahran for support.

References

[1] A. S. Krishnakumar and K. Sabnani, "VLSI implementations of communication protocols – a survey," *IEEE Journal on Selected Areas in Communication*, vol. 7, pp. 1082 –1090, Sep. 1989.

[2] T. Murata, "Petri nets: properties, analysis and application," *Proceedings of the IEEE*, vol. 77, pp. 541–580, Apr. 1989.

[3] G. C. Illing, "Automatic Petri net based protocol implementation," in *IRRECON International*, (Melbourne, Australia), pp. 356–361, Sep. 1989.

[4] I. Ericsson, "Protocol controller chip manages X.25 interface," *Computer Design*, vol. 24, pp. 78 – 81, September 1st 1985.

[5] M. Aoki, T. Uchiyama, S. Tonami, A. Hayakawa, and H. Ichikawa, "Protocol processing for high speed packet switching systems," in *Proceedings of International Zürich Seminar*, (Zürich, Switzerland), pp. 125 – 130, March 1986.

[6] G. Chesson, "The protocol engine project," *Unix Rev.*, pp. 70 –77, Sep. 1987.

[7] S. S. Patil, "The description and realization of digital systems," in *COMPCON 72:Sixth Annual IEEE Computer Society International Conference Digest of Papers*, (New York), pp. 223–226,

IEEE press, Sep. 1972. Also, Computation Structures Group Memo 71, Project MAC, MIT, Cambridge, Massachusetts, October 1972, 4 pages.

[8] F. Furtek, *Modular Implementation of Petri Nets.* Master's thesis, Department of Electrical Engineering, MIT, Cambridge, Massachusetts, Sep. 1971.

[9] T. Chu, "On the models for designing VLSI asynchronous digital systems," *Integration, The VLSI Journal*, no. 4, pp. 99–113, 1986.

[10] C. E. Molnar, "Synthesis of delay insensitive modules," in *Proceedings of the 1985 Chapel Hill Conference on VLSI*, (Rockville, MD), Computer Science Press, 1985.

[11] C. L. Kwan, P. L. Beux, and C. Michel, "The design of structured digital systems controlled by PLA," in *Microcomputer Architecture, Euromicro 1977*, (J. D. Nicoud, J.Wilmink, and R. Zaks, eds.), North Holland Publishing Company, 1977.

[12] M. Auguin, F. Boeri, and C. Andre, "Systematic method of realization of interpreted Petri nets," *Digital Processes*, vol. 6, no. 1, pp. 55–68, 1980.

[13] G. V. Bochmann and C. A. Sunshine, "Formal methods in communication protocol design," *IEEE Transactions on Communications*, vol. COM-28, pp. 624–631, Apr. 1980.

[14] M. Diaz, "Modelling and analysis of communication and coperation protocols using Petri net based models," *Computer Networks*, no. 6, pp. 419–441, 1982.

[15] G. V. Bochmann, "Finite state description of communication protocols," in *Conference on Computer Network Protocols*, (Liège), 1978. Also, in Computer Networks-2, 1978, pp.361–72.

[16] A. A. S. Danthine, "Protocol representation with finite state models," *IEEE Transactions on Communication*, vol. COM-28, pp. 632 –642, Apr. 1980.

[17] K. Lee and J. Favrel, "Hierarchical reduction method for analysis and decomposition of Petri nets," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-15, March/April 1985.

[18] G. Berthelot, *Transformations and Decompositions of Nets.* Vol. Petri nets: Central Models and their Properties of *Lecture Notes in Computer Science, Vol. 254*, New York: Springer Verlag, 1987.

[19] S. M. Shat, P. S. Kajkaet, and A. S. Chauhan, "Formal modelling and automated analysis of lapd protocol," *Computer Networks and ISDN Systems*, pp. 293–314, May 1990.

[20] E. T. Morgan and R. R. Razouk, "Computer aided analysis of concurrent systems," in *Protocol Specification, Testing and Verification V.* (M. Diaz, ed.), pp. 49 –58, Elsevier Science Publishers B. V., North Holland, 1986.

[21] R. Valk, "Petri nets and regular languages," *Journal of Computer and System Sciences*, vol. 23, pp. 299 –325, 1981. in Mathematical Foundations of Computer Science 10^{th} Symposium , pages 140-155.

[22] A. M. T. Khan, *VLSI Implementation of a Class of Digital Systems from their Petri Net Models.* Master's thesis, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, 1992. Expected January 1992.

[23] D. P. Sidhu, "Protocol design rules," in *Protocol Specification, Testing and Verification I*, (C. Sunshine, ed.), pp. 283–300, Elsevier Science Publishers B. V., North Holland, 1982.

[24] H. M. A. Fahmy, "Analysis of Petri nets by partitioning:splitting transitions," *Theoritical Computer Science*, pp. 321–330, Jan. 1990.

[25] W. S. Scott, R. N. Mayo, G. Hamachi, and J. K. Ousterhout(eds.), "1986 VLSI tools:still more work by the original artists," Report UCB/CSD 86/272, University of California at Berkeley, Berkeley, Dec. 1985.

[26] W. S. Scott, R. N. Mayo, G. Hamachi, and J. K. Ousterhout(eds.), *VLSI Design Tools Reference Manual, Release 3.1.* Seattle, Washington, Feb. 1987.

[27] A. M. T. Khan, S. M. Sait, and G. F. Beckhoff, "High level synthesis of controllers for communication protocols from their Petri net models," in *1992 International Symposium on Circuits and Systems*, 1992. Accepted for presentation.

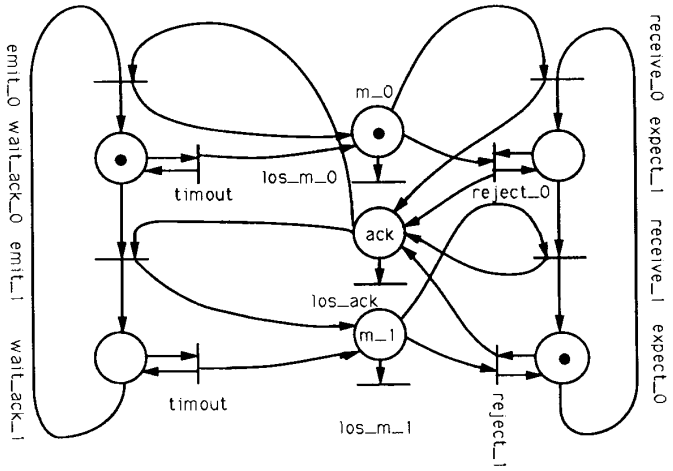

Figure 3: The Petri net model of the protocol PAR

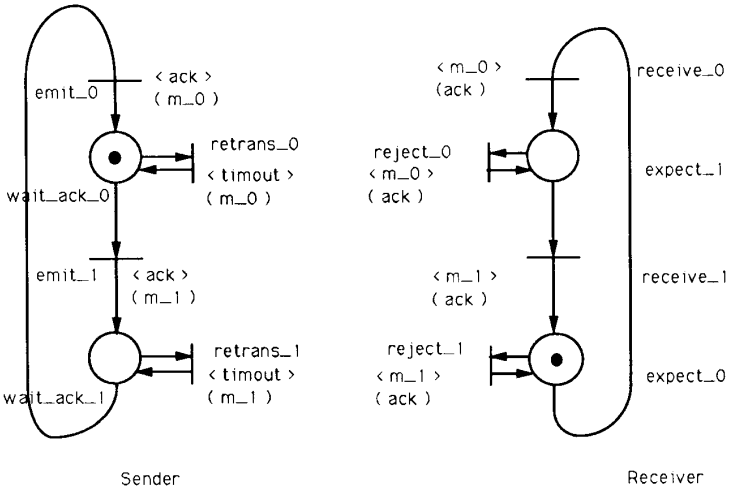

Figure 4: Application of procedure Break_Entities on the Petri net model of PAR

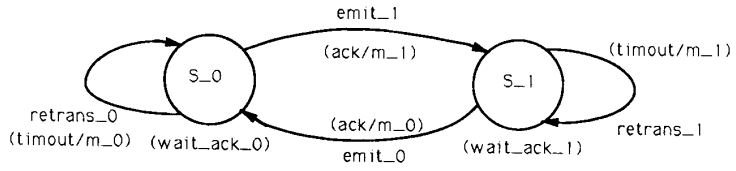

Figure 5: The FSMs obtained for the sender process of the protocol PAR

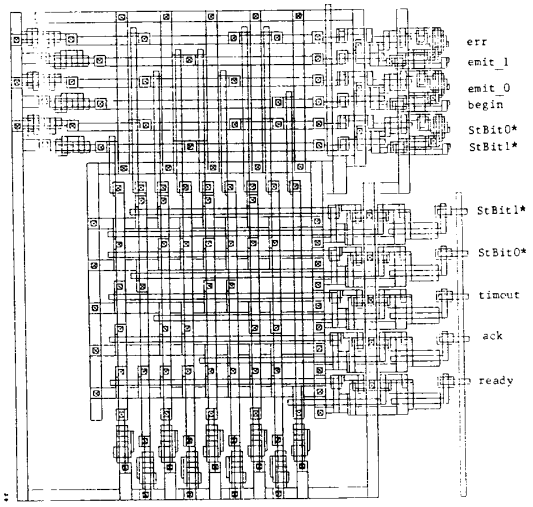

Figure 6: The nmos PLA layout of the sender process of the protocol PAR

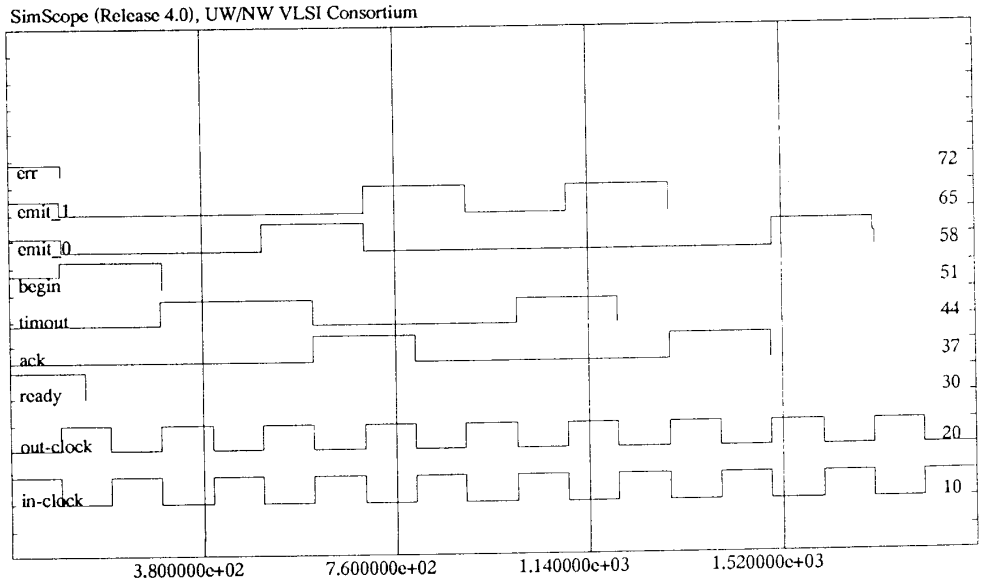

Figure 7: The timing diagram for extracted sender process of the protocol PAR