

AN APPROACH TO PERFORMANCE SPECIFICATION
OF COMMUNICATION PROTOCOLS USING TIMED PETRINETES

K. GARG

Departing of Computing
Imperial College, London SW7, U.K.

ABSTRACT

There has been a lot of interest in the past decade in using Timed Petrinets to model computer systems. In this paper, we show how such Timed Petrinets can be used to great advantage in describing and algebraically specifying communication protocol performance, by making use of the time parameter of Timed Petrinets to model the delay in performing certain operations of the protocol. The algebraic notation used for the specification is borrowed from the recently reported AFFIRM language, and the protocol chosen for illustration is the ECMA transfer protocol proposed for the ISO Reference Model. However, the methodology can be used with other protocols too. We also show how Liveness properties can be specified easily using Timed Petrinets.

INTRODUCTION

Petrinets have been extensively used in the past for modelling computer systems. They have now become very popular for representing Distributed Computer Systems because of their capability of clearly describing concurrency, conflicts and synchronization of processes, and because they present a simple yet elegant formalism for their description.

Original Petrinets were used to represent only the logic behaviour of systems with no timing and performance considerations. Over the past decade, however, the notion of time has been included into Petrinet models, making it easier to analyse the performance characteristics of such systems. Performance influences all aspects of computer systems, from their design to management. Hence, it is important to study in detail a system's performance characteristics at every stage of design. It is especially important to be able to analyze and predict the

performance of a newly designed system since its actual performance measurement is intractable.

Performance specification of a protocol is concerned with stating and proving algebraically that the protocol works in such a manner as to accomplish its job within a time acceptable to its higher level user service. Indeed, this should be the case for all possible conditions of the message transfer medium and for all sequences of transitions needed to do the message transfer correctly. Thus the time taken by the protocol for correct message transfer should be less than or equal to that allowed for the transfer by the user service. This is important in order to meet the real time requirements of the system. It is also important that the time taken by the protocol be known, as it controls the buffer holding times and hence buffer requirements in the communication subsystem.

It has been shown recently by several researchers [5] that it is easy to formally specify and prove certain Safety (correct delivery) properties for a protocol. However, Liveness (effective work being done) and Performance (work being done fast enough) properties cannot be considered because of certain inherent limitations in the descriptive power of most State Transition Machine (STM) models of protocols.

This paper shows how, using Timed Petrinets rather than STMs as used in [5], it is possible to specify and prove both the performance and Liveness properties of protocols. In fact, the work reported here has been largely motivated and influenced by that reported in [5].

In this section we give an overview of Timed Petrinets and of the AFFIRM notation for protocol specification as described in [5]. In section II is given an overview of our method of the formal specification of the performance of a protocol using Timed Petrinet models. Section III illustrates in detail the

* on study leave from the Dept. of Electronics, University of Roorkee, India.

methodology as used for the ECMA protocol. In Section IV we discuss the extensions to and the scope and limitations of the methodology. Our conclusions are given in section V.

A. Timed Petrinets Timed Petrinets (TPNs) were first introduced by Ramchandani [1] by associating firing times to the transitions of ordinary Petrinets, in order to study their steady state behaviour. Since then many researchers have reported work on TPNs [2],[3],[4]. In some cases, fixed times have been used, leading to deterministic models [4]. In others, random firing times have been associated with transitions, resulting in stochastic TPN models [2]. Sifakis [3] has associated fixed delay times with the 'places' of Petrinets and shown that, given a set of relations describing the behaviour of such a TPN, its maximum computation rate can be found. We have chosen Sifakis's TPNs to model our protocols.

Timed Petrinets theory

Sifakis has defined and represented Timed petrinets as follows -

Definition A Timed Petrinet (TPN) is a 6 - tuple $N = (P, T, \alpha, \beta, \tau, v)$ where

P : is a set of places, $P \neq \emptyset$

T : is a set of transitions, $T \neq \emptyset$, $P \cap T = \emptyset$

$\alpha: P \times T \rightarrow \mathbb{N}$ forward incidence function

$\beta: T \times P \rightarrow \mathbb{N}$ backward incidence function

($\mathbb{N}$ represents the set of natural integers)

$\tau = (\tau_1, \tau_2, \dots, \tau_l, \dots)$

an increasing sequence of real numbers called time base

a mapping $v: P \times T \rightarrow \mathbb{R}$ such that

$$\forall (p, \tau_i) \in P \times T: v(p, \tau_i) = \tau_j \Rightarrow \tau_j \geq \tau_i$$

Representation

To a TPN one can associate a digraph the nodes of which are the places and the transitions, represented respectively by circles and dashes. There is a directed edge from the place p_s to the transition t_j iff,

$$\alpha(p_s, t_j) = n_{s,j} \neq 0$$

There is also a directed edge from transition t_r to place p_w iff,

$$\beta(p_w, t_r) = n_{w,r} \neq 0$$

A marking M of a PN, shown by dots in the place p , is a mapping of P into $\mathbb{N}$,

$$M: P \rightarrow \mathbb{N}$$

Rules

a) A marker in a TPN may be in one of

the two following states : **available** or **unavailable**. Initially, each place p contains $M_0(p)$ available markers.

b) A transition t is enabled iff every place p_s , which has a directed edge to t , contains $\alpha(p_s, t)$ available markers at least.

c) The firing of a transition t has to take place instantaneously as it is enabled. It consists in removing $\alpha(p_s, t)$ available markers from each of the places p_s and in placing $\beta(p_w, t)$ markers in each place p_w which has a directed edge from t .

d) A marker remains unavailable in a place p_s during the time interval z_i after which it becomes available. Thus each marker is delayed by a time z_i in the place p_i where

$$\forall (p_i, \tau) \in P \times T: v(p_i, \tau) - \tau = z_i = \text{constant}$$

For a TPN with n places, a vector $Q^t(\tau)$ is defined (Q^t denotes the transpose of Q), where

$$Q^t(\tau) = [q_1(\tau), q_2(\tau), \dots, q_n(\tau)]$$

and is called the charge variable at the instant τ . Each entry $q_i(\tau)$ represents the marking at p_i at the instant τ .

For an interval $\Delta\tau = \tau - \tau_0$, the mean variation of the charge of the net is given by the vector $I(\tau) = R(\tau)/\Delta\tau$ and is called the current vector. where $R \in \mathbb{R}^n$

Each entry i_k of the vector I represents the mean frequency of firing of transition t_k during $\Delta\tau$.

Sifakis has shown that in order for the total charge in the net to remain bounded, constant currents are to be assigned to the transitions. This results in **periodic** firings in the net.

The maximum computation rate corresponding to this periodic firing can be calculated by solving a set of n linear equations given as

$$CI = 0, \quad I > 0 \quad \dots (1)$$

$$\{J^t Q(\tau_0) \geq J^t Z C^+ I\}^k_{s=1} \quad \dots (2)$$

where C = incidence matrix of the TPN

$$C = [c_{ij}] \quad \text{with} \quad c_{ij} = \begin{cases} \beta(p_i, t_j) & \text{if } \neq 0 \\ -\alpha(p_i, t_j) & \text{if } \alpha(p_i, t_j) \neq 0 \\ 0, & \text{if not} \end{cases}$$

$$\text{and } C^+ = [c_{ij}^+] \quad \text{with} \quad c_{ij}^+ =$$

$$= \begin{cases} \beta(p_i, t_j) & \text{if } \neq 0 \\ 0, & \text{if not} \end{cases}$$

$\{J^t\}$ is a generator of $\mathcal{X}^t$ where $\mathcal{X}$ is the set of non-negative solutions of $CX = 0$.

$Q(c_0)$ is the initial marking vector and Z is the (square) delay matrix

$$\begin{bmatrix} z_1 & 0 & 0 & 0 & \dots \\ 0 & z_2 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \ddots \\ 0 & 0 & 0 & 0 & \dots z_n \end{bmatrix}$$

The net is said to function at a natural rate if the inequality (2) is changed to equality.

The above linear equations can be found very easily if the TPN is decomposed into subnets. In this respect, Sifakis has shown that searching for nonnegative solutions of $CI = 0$ (resp. $J^t C = 0$) amounts to decomposing the TPN into consistent (invariant) components, which correspond to State Graphs (Marked Graphs).

B. The AFFIRM system [5].

AFFIRM is a general purpose automated system for the algebraic specification and verification of user defined abstract data types, developed at ISI, California. It uses the State Transition Machines for modelling the behaviour of systems, rendering these in the AFFIRM language, expressing its properties algebraically as theorems and proving them using an interactive theorem prover. Work has been focussed mainly on proving the Safety properties for protocols rather than on Liveness or Performance. Details of these can be found in the reference cited above.

We have found that the algebraic notation of AFFIRM can be used, with suitable extensions to the data types defined, to formally state and prove certain performance properties of protocols. This is done by giving a complete specification of the protocol using TPNs instead of STMs. All of the AFFIRM notation for STMs is still valid for petrinets as they are a general case of STMs. In fact, a petrinet is an STM if the number of input and output places of a transition is always equal to one. It is well known that petrinets are a better tool for modelling protocols as compared to STMs.

Next we give very briefly, an outline of the AFFIRM's algebraic specification and theorem proving methodology before attempting to use it for our protocol. In the AFFIRM notation, an abstract data

type is specified by

i. defining its three functions, namely, constructors, selectors and extenders[6]

ii. the extenders and selectors are defined by equational axioms, relating how each function behaves when applied to each of the constructors.

iii. These axioms, which look like equations, are treated as left to right rewriting rules and are used to state and prove certain invariant properties of the data type being specified.

iv. The proof is by Structural Induction, i.e., based on the constructors of the data type.

II. AN OVERVIEW OF THE METHOD FOR PERFORMANCE SPECIFICATION

Our method of specifying communication protocol performance in terms of the properties of its abstract data type representation, can be summarised as follows.

1) Draw a detailed TPN diagram of the functioning of the protocol. Use 'places' to represent its states, buffers, medium and other data structures and 'transitions' to represent events or operations.

2) Reduce the TPN as much as possible by disregarding, for example, places and transitions not considered with performance issues [15].

3) Analyze the TPN model of the protocol and obtain the various possible distinct transition sequences of operation of the protocol. This is done by solving the equation $CI = 0$ for the net, where C is its incidence matrix. Find also all the independent solutions of the equation $J^t C = 0$.

4) Draw the relative subnets of the TPN, one for each I vector and J vector.

5) Write a formal specification of the protocol TPN as an abstract data type in a suitable specification language. Include a variable (Timein) to specify the time delay associated to each place. Also include a variable (selector of the protocol type) to specify the time elapsed (Timeelapsed) after each operation (transition) of the protocol. Write the axioms for this selector showing when and how it is modified (if at all) by every transition.

6) For each sequence of transitions I , write down the times taken to complete

the sequence by inspection from each J vector subnet. This can be done by simply adding the time delays z_i of the places p_i of the subnet for J, which are involved in completing this sequence. The time taken for each sequence I is the largest of the times given by the J subnets. If there is only one possible J vector, then the time taken for each I sequence is equal to the time found from that J subnet.

7) Rewrite the above formally as theorems in the specification language for the selector variable Timeelapsed using the values of the variables 'Timein' for the places. One theorem can be written for each I vector.

III. PERFORMANCE SPECIFICATION OF ECMA PROTOCOL

To illustrate that the methodology described in the last section can be used with most existing (and standard) communication protocols, we have chosen the European Computer Manufacturer's Association (ECMA) [7] transport protocol. This is an end-to-end protocol standing between the session and network layers of the ISO model. We shall concentrate only on the connec-

tion and disconnection phases for classes 1 and 2 of this protocol since this will suffice for our purpose of illustration.

Connection and Disconnection TPN model

Two transport entities communicate by establishing a transport connection first. This connection is established only between initially unconnected entities. There is one initiating and one accepting entity. The TPN model of the behaviour of the entities in the connection and disconnection phase is shown in Fig.1 [8]. In the initial state (p_1), the initiating entity sends (t_1') a connect request (p_1'). The accepting entity (p_6) accepts the request (t_1) by sending the connect confirm (p_2') and the connection is established (t_1''). Both entities now pass to the data transfer phase (p_2 and p_7). Once in this phase, each entity may end a transport connection by sending (t_2 and t_6) a disconnect request (p_4 and p_9) and pass to the waiting disconnect confirm state (p_3 and p_8). On receipt of such a request, the remote entity returns to the initial state either by sending (t_3 and t_6) a disconnect confirm (p_5 and p_{10}) if it was in the data transfer state, or not (t_5 and t_9) if a discon-

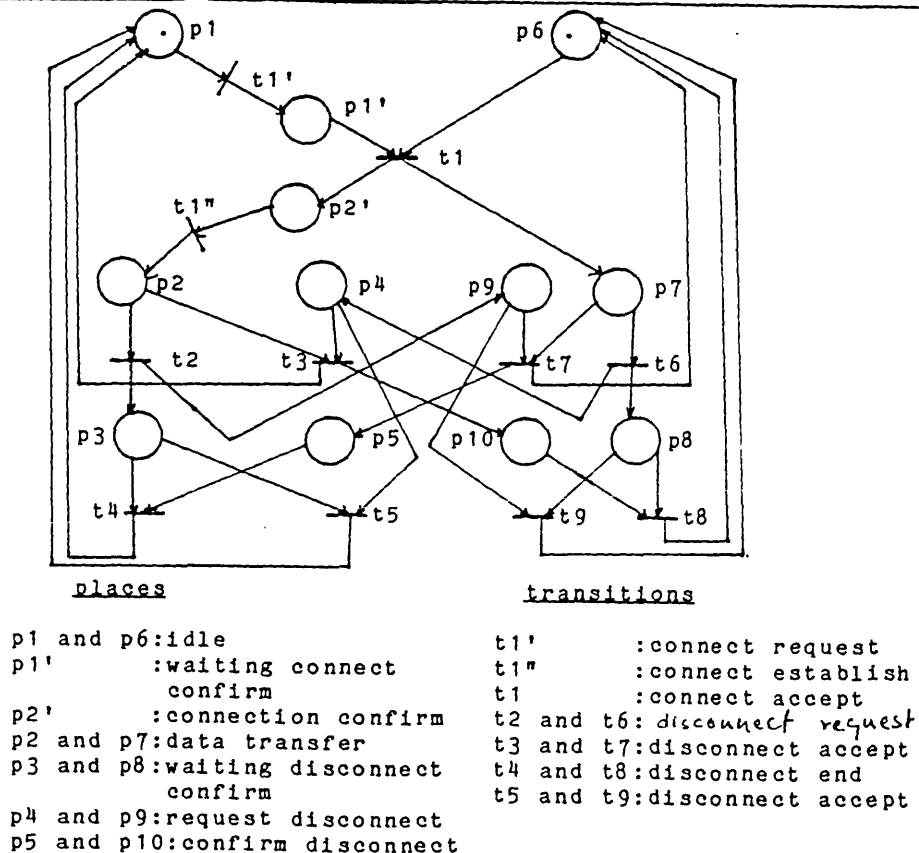


Fig. 1. Timed Petri net for ECMA connection-disconnection phase

next collision has occurred and it was in the waiting for disconnect confirm state. A time delay z_1 is assumed associated with each place p_1 (except p_3 and p_8). Because of the serial nature of the transitions t_1, t_1' and t_1 and the places p_1' & p_1 and p_2' & p_2 we can simplify the figure to that of Fig. 2 and assume that the time delays associated with p_1' and p_2' have been included in z_1 and z_2 respectively. This in fact means that the connection phase is now represented only by the transition t_1 . Fig. 2 will henceforth be used to model our protocol.

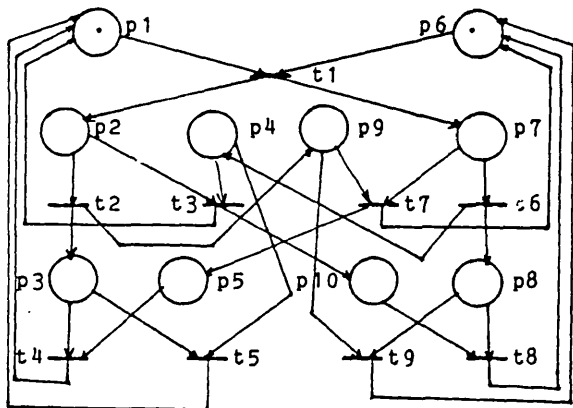


Fig. 2. The reduced Timed Petrinet model

Analysis of the TPN model

We have, for Fig. 2,

$$Z = \text{diag} [z_1 \ z_2 \ \dots \ z_{10}]$$

$$Q^t(\tau_0) = [1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0]$$

$$C = \begin{bmatrix} -1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & -1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

$CI = 0$ has three solutions given by

$$\begin{aligned} I_1 &= [1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0] \\ I_2 &= [1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0] \\ I_3 &= [1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1] \end{aligned}$$

$J^t C = 0$ has four solutions given by

$$\begin{aligned} J_1^t &= [1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\ J_2^t &= [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0 \ 0] \\ J_3^t &= [1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 0] \\ J_4^t &= [0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1 \ 1] \end{aligned}$$

The various subnets representing these

vectors are given in Figs. 3a)-c) and 4a)-d) respectively.

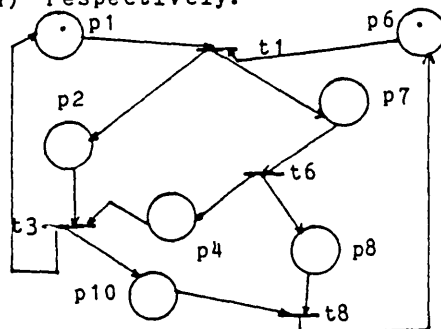


Figure 3a. Subnet for I1

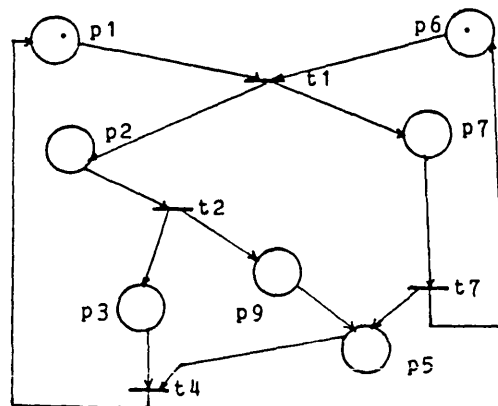


Figure 3b. Subnet for I2

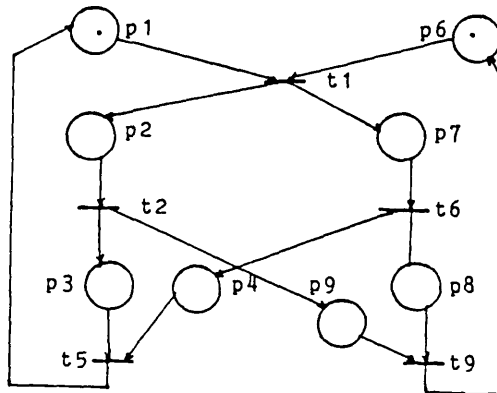


Figure 3c. Subnet for I3

For each I vector, the maximum rate of computation can be written from the inequalities

$$J_i Q \geq J_i Z C I \quad | \quad i=1, 2, 3, 4$$

Thus we have, for the sequence of transitions given by I_1 ,

$$i1_{\max} = \min. [1/(z_1+z_2) \quad 1/(z_6+z_7) \quad 1/(z_1+z_4+z_7) \quad 1/(z_2+z_6+z_{10})]$$

For the sequence given by I_2 we have

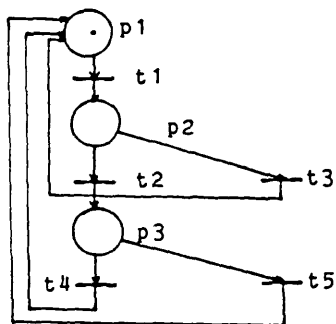


Figure 4a. Subnet for J1

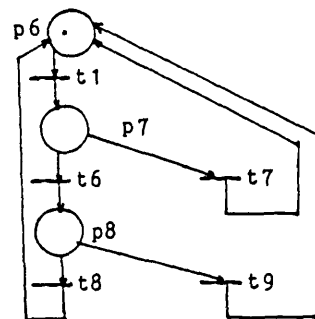


Figure 4b. Subnet for J2

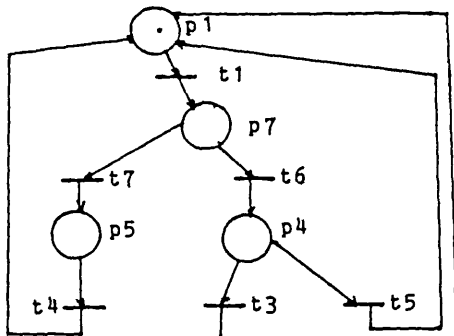


Figure 4c. Subnet for J3

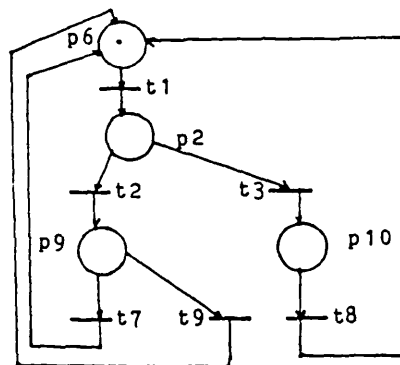


Figure 4d. Subnet for J4

$$i1max. = \min. [1/(z1+z4+z7) \quad 1/(z2+z6+z9) \quad 1/(z1+z2) \quad 1/(z6+z7)]$$

and for the sequence given by I3

$$i1max. = \min. [1/(z1+z5+z7) \quad 1/(z2+z6+z9) \quad 1/(z1+z2) \quad 1/(z6+z7)]$$

Conversely, we can state that the minimum times taken for one cycle of each sequence to be completed are

$$Tmin. = \max. [(z1+z2) \quad (z6+z7) \quad (z1+z4+z7) \quad (z2+z6+z10)] \text{ for I1} \quad \dots(1)$$

$$Tmin. = \max. [(z1+z2) \quad (z6+z7) \quad (z1+z4+z7) \quad (z2+z6+z9)] \text{ for I2} \quad \dots(2)$$

$$Tmin. = \max. [(z1+z2) \quad (z6+z7) \quad (z1+z5+z7) \quad (z2+z6+z9)] \text{ for I3} \quad \dots(3)$$

Alternatively, we can get these values from inspection of the figs.4a-d for each of the sequences I. We interpret the figures as follows-

For the sequence I1 (Fig.3a), which corresponds to the firing of the transitions (t1;t6;t3;t8), the time taken is the same as that taken for the
sequence (t1;t3) - fig.4a
sequence (t1;t6;t8) - fig.4b
sequence (t1;t6;t3) - fig.4c and
sequence (t1;t3;t8) - fig.4d

These times can be obtained from each figure by simply adding up the time

delays of the places involved in firing the constituent transitions of the sequence I1. These are

$$(z1+z2), (z6+z7), (z1+z4+z7), (z2+z6+z7)$$

The largest of the above gives (1). But we know that the transition t3 in sequence I1 fires only if

$$z2 \geq (z4 + z7)$$

Thus (1) reduces to

$$Tmin. = \max. [(z1+z2), (z2+z6+z10)] \dots(4)$$

Similar arguments lead to the following equations for I2 and I3

$$Tmin. = \max. [(z6+z7), (z1+z5+z7)] \dots(5)$$

$$Tmin. = \max. [(z1+z4+z7), (z2+z6+z9)] \dots(6)$$

Formal specification of the ECMA protocol

In order to formally specify the performance properties of the ECMA protocol, we must first write down its complete formal specification as a data type with axioms describing the relations between selectors and constructors. However, here we need to identify only the axioms concerned with 'time' and these are given below. For convenience we use the labels pi and ti for the places and transitions instead

of their actual names.

Each of the axioms for 'Timeelapsed' states that the time that has elapsed when a transition is fired can be stated in terms of the time elapsed when some previous transition fired. Its value then depends on the intervening 'Timein' variable. The previous transition(s) are identified by the resulting Marking (state) from that transition(s).

To formally specify the state of the protocol after each transition, we have used its Marking vector. This is specified as being of the type 'Register' such that operations like set, clear, testbit etc., can be performed on it [10]. The length of the

Register is given by the number of places in the protocol Petrinet, each of which can be considered as a bit of the Register. This bit is either ON (a token is present in the place) or OFF (absence of token in the place). The 'Reading' of the Register at any instant gives the Marking (state). Thus Reading(1,6) denotes the state of the protocol when there are tokens present in places p1 and p6 only. Marking is considered as a selector (variable) for the protocol type and is modified by its each operation (Transition).

The axioms for the selector variable 'Timeelapsed' for the ECMA protocol can be written as in the Figure 5.

```

Timeelapsed(t1(s))=if Marking(s)=Reading(1,10,8)
    then Timeelapsed(s)+GreaterOf( Timein(p1) or ((Timein(p6)
        +Timein(p10)) )
    else if Marking(s)=Reading(1,6)
        then Timeelapsed(s)+GreaterOf( Timein(p6)or(Timein(p1)
            -Timein(p10)) )
        else if Marking(s)=Reading(3,5,6)
            then Timeelapsed(s)+GreaterOf( Timein(p6) or (Timein
                (p1)+Timein(p5)) )
            else Timeelapsed(s),

Timeelapsed(t2(s))=if Marking(s)=Reading(2,7)
    then Timeelapsed(s)+Timein(p2)
    else if Marking(s)=Reading(2,8,4)
        then Timeelapsed(s)+Timein(p2)-Timein(p7)
        else Timeelapsed(s),

Timeelapsed(t3(s))=if Marking(s)=Reading(4,8,2)
    then Timeelapsed(s)+Timein(p2)-Timein(p7)
    else if Marking(s)=Reading(2,7)
        then Timeelapsed(s)+Timein(p2)
        else Timeelapsed(s),
{by definition, the transition t3 can fire only if  $z_2 \geq z_4 + z_7$  }

Timeelapsed(t4(s))=if Marking(s)=Reading(3,9,7)
    then Timeelapsed(s)+Timein(p5)+Timein(p7)-Timein(p2)
    else if Marking(s)=Reading(5,6,3)
        then Timeelapsed(s)+Timein(p5)
        else Timeelapsed(s),

Timeelapsed(t5(s))=if Marking(s)=Reading(3,9,7) or Reading(3,4,8,9)
    then Timeelapsed(s)+Timein(p4)+Timein(p7)-Timein(p2)
    else if Marking(s)=Reading(2,4,8) or Reading(3,4,9,8)
        then Timeelapsed(s)+Timein(p4)
        else if Marking(s)=Reading(3,4,6)
            then Timeelapsed(s)+((Timein(p4)+Timein(p7))
                -(Timein(p2)+Timein(p9)))
            else Timeelapsed(s),
{the transition t5 can only fire if  $(z_4 + z_7) > z_2$  }

Timeelapsed(t6(s))=if Marking(s)=Reading(2,7)
    then Timeelapsed(s)+Timein(p7)
    else if Marking(s)=Reading(3,7,9)
        then Timeelapsed(s)+Timein(p7)-Timein(p2)
        else Timeelapsed(s),

```

Figure 5. ECMA Specification

```

Timeelapsed(t7(s))=if Marking(s)=Reading(2,7)
    then Timeelapsed(s)+Timein(p7)
    else if Marking(s)=Reading(3,9,7)
        then Timeelapsed(s)+Timein(p7)-Timein(p2)
        else Timeelapsed(s),
{transition t7 can fire only if  $z7 \geq z2+z9$ , by definition}

Timeelapsed(t8(s))=if Marking(s)=Reading(4,8,2)
    then Timeelapsed(s)+Timein(p2)+Timein(p10)-Timein(p7)
    else if Marking(s)=Reading(1,8,10)
        then Timeelapsed(s)+Timein(p10)
        else Timeelapsed(s),

Timeelapsed(t9(s))=if Marking(s)=Reading(3,9,8,4) or Reading(2,4,8)
    then Timeelapsed(s)+Timein(p2)+Timein(p9)-Timein(p7)
    else if Marking(s)=Reading(3,4,9,8) or Reading(3,7,9)
        then Timeelapsed(s)+Timein(p9)
        else if Marking(s)=Reading(1,8,9)
            then Timeelapsed(s)+(Timein(p2)+Timein(p9))
            -(Timein(p4)+Timein(p7))
            else Timeelapsed(s);
{transition t9 can fire only if  $z2+z9 > z7$ }

```

Figure 5. (contd.) ECMA specification

theoremperformance1

for all s, $\text{Timeelapsed}(t1(t8(t3(t6(t1(s))))) = \text{Timeelapsed}(t1(s)) + \text{GreaterOf}((\text{Timein}(p1) + \text{Timein}(p2)) \text{ or } (\text{Timein}(p2) + (\text{Timein}(p6) + \text{Timein}(p10)))$)

theoremperformance2

for all s, $\text{Timeelapsed}(t1(t4(t7(t2(t1(s))))) = \text{Timeelapsed}(t1(s)) + \text{GreaterOf}((\text{Timein}(p6) + \text{Timein}(p7)) \text{ or } (\text{Timein}(p1) + (\text{Timein}(p5) + \text{Timein}(p7)))$)

theoremPerformance3

for all s, $\text{Timeelapsed}(t1(t5//t9(t2//t6(t1(s)))) = \text{Timeelapsed}(t1(s) + \text{GreaterOf}((\text{Timein}(p1)+\text{Timein}(p4)+\text{Timein}(p7)) \text{ or } (\text{Timein}(p2)+\text{Timein}(p6)+\text{Timein}(p9)))$)

Figure 6. ECMA Performance theorems

Statement of the ECMA Performance

The performance relations of the ECMA connection-disconnection phase can now be stated formally as the theorems given in Figure 6.

The theorem performance1 states formally the time taken for the sequence I1 to be completed. It says that the time taken in the Connection-Disconnection phase if the accepting entity requests for disconnection is equal to the greater of the times $(z1+z2)$ and $(z2+z6+z10)$.

The theorem performance2 states formally the time taken for the sequence I2 to be completed. It says that the time taken in the Connection-Disconnection phase if the initiating entity requests for disconnection is equal to the greater of the times $(z6+z7)$ and $(z1+z5+z7)$.

The theorem performance3 states formally the time taken for the sequence I3 to be completed. It says that the time taken in the Connection-Disconnection phase if either of the two entities requests for disconnection is equal to the greater of the times $(z1+z4+z7)$ and $(z2+z6+z9)$.

All the above theorems can be proved by Structural Induction [5].

IV. GENERAL DISCUSSION

In this section we discuss the methodology in terms of extensions possible in it and its scope and limitations.

The methodology developed as a result of our attempt to do a formal specification of the Liveness and Performance property

of the INPAR protocol designed at Imperial College for a multiprocessor type station (node) for Distributed Computer Control Systems [9]. This is an Immediate Negative, Positive Acknowledgement Retransmissions protocol providing a Reliable Broadcast Datagram service on a single bus (Message Bus) based Multicomputer. It discards duplicate or corrupt messages and informs any failure in sending a message to the user. The protocol uses immediate NAKs/Ack as responses from receiver to sender to ensure correctness of transfer. There is no explicit connection-disconnection phase in the protocol.

We were able to specify completely the working of the protocol using AFFIRM and its TPN model. It was verified that the protocol indeed implements its required service. We also specified and proved its safety, Liveness and Performance properties [15].

Liveness Properties (something eventually happens)

Initially, we developed a State Transition Machine (STM) model for our protocol [10] and used it for formal specification and validation. But it was found that while 'safety' properties for the protocol were easy to specify and prove, 'Liveness' and 'Performance' properties could not be stated and proved. This was due to the inherent limitations of using STMs.

By later opting for a TPN based specification, we were able to overcome these limitations. It was found that because of the definition of the firing of transitions in a TPN, each transition has to fire as soon as all its enabling (input) places have available tokens. This models the **eventuality** of the transitions. In other words, there are no probabilities associated with the firing of transitions and it can be said that there is always both a necessary and sufficient condition for a transition to fire.

Hence it was possible to specify liveness properties of the protocol using the Marking vector to represent the states of the protocol, as discussed in Section III. We were able to state and prove that for every possible sequence of operation of the protocol, starting from the initial state, it was eventually possible to reach the same state again.

For the ECMA protocol phase considered in this paper, we can similarly specify its Liveness by stating the theorem

for ECMA progress as

theorem ECMA progress

for all s_1, s_2 ($s_2 = t_8(t_3(t_6(t_1(s_1))))$) and
 Marking(s_1) = Reading(1,6)
 imp Marking(s_2) = Reading(1,6));

This theorem is stated for the sequence of transitions given by I1. We can similarly state the liveness property for the sequences I2 and I3.

Temporal logic has hitherto been considered the most appropriate for specifying progress of a protocol [12]. Our approach using Timed Petrinets can be considered to compare favourably with it.

Timeouts

The inclusion of the selector variable representing time elapsed in the protocol (Timeelapsed) after each transition, enables us to specify any protocol in which the sender times out if a response (Ack) is not received from the receiver within a certain time interval. The actual Time Out event can be specified as an operation (transition) of the protocol TPN. Its enabling condition can be taken as the appropriate value of the variable Timeelapsed.

Control vs data

At present we have not explicitly considered separating control messages from data messages in specifying the underlying medium, or identifying the actual data message since this is not possible to do with ordinary TPNs as hitherto defined. This may be necessary in order to specify more precisely the type of response (NAK/Ack) received, or the order of their transmission. Predicate/Transition nets are considered very useful for such representation [11], where arcs to transitions give the variables (predicates) defining the pattern of tokens used by the transitions. Such Predicate/Transition (Pr/tr) nets can be expanded to obtain ordinary Petrinets [8].

By incorporating time delays in the places of the Pr/tr nets, resulting in Timed Pr/tr Nets, it may be possible to analyse them for performance characteristics and thus specify the performance properties of protocols modelled by them.

Explosion of states

It is quite possible that in the TPN model of the protocol, there is an

explosion of states for complex protocols. It then becomes difficult and sometimes impossible to specify each state separately. This difficulty was experienced for the INPAR protocol specification.

However, we were able to overcome this very common limitation of Petrinets, by considering some but not all the places of the net for defining its Marking vector (state). Places which did not contribute at all to the 'conditions' for firing of transitions but were present only to model certain data structures of the protocol, were not considered in defining the state of the protocol. If necessary such a place was specified explicitly as an enabling variable.

This approach helped to reduce considerably the total number of distinct states in the protocol and hence simplify its formal specification. We feel that this is possible to do in most cases.

Deterministic Model

The TPN used to model the ECMA protocol can be seen to be a deterministic one because of its definition for transitions. It may be argued that such deterministic models do not effectively represent the non-deterministic nature of a communication protocol as represented by the probabilistic nature of the underlying medium. For example, in the connection-disconnection phase of ECMA discussed above, it is possible that the data transfer state may be cut short by the arrival of a remote disconnection request. The model cannot represent this as the data transfer times are constant (z_2 and z_7), i.e., it is assumed that any remote request for disconnection can only arrive after the data transfer state is over. This assumption may not necessarily be true for the protocol. Thus our TPN model may be more deterministic than the actual protocol it models.

However, since the non-deterministic nature of the medium is manifested by the need for retransmissions, we are able to cover this fully by explicitly considering all possible transition sequences in the model.

V. CONCLUSION

The paper has shown how, using the effective representative power of Timed Petrinets, it is possible to specify algebraically and prove certain performance properties of a communication

protocol and hence increase our confidence in it. Timed Petrinets have been shown to be ideal tools to model, represent and specify protocol characteristics relating to liveness and performance.

We have specified performance in terms of the time taken to complete certain operations of the protocol. There may be other ways in which performance properties could be specified. As far as we are aware, this is the first reported work on formal specification of protocol performance properties, though there has been a general awareness for some time now that it is not enough to specify only Safety and Liveness for protocols [13].

Liveness properties were hitherto considered to be difficult or impossible to specify using finite state machine models, but we have shown that use of TPNs also overcomes this limitation.

We have used these TPNs to analyse the performance of our Message Bus based multiprocessor system in general, not just its protocol [14]. TPNs can be used to effectively represent and model most subsystems of multiprocessor structures, i.e., Bus Contention system and Message Buffering systems. These can be used to draw a composite TPN model for a multicomputer and then to specify and prove its performance in terms of its Speed Up and Efficiency, as shown in [15], where techniques for doing this have been presented. We have also shown how certain design choices can be validated using these techniques.

A lot of work remains to be done to consolidate the work reported here and to overcome the limitations discussed and we hope that this paper will provide a sound basis for such further work.

Acknowledgements

The author would like to thank her supervisor Dr. M. Sloman for advice and encouragement and the U.K. Commonwealth Scholarships Commission for the Scholarship which made this work possible.

References

1. C Ramchandani, 'Analysis of asynchronous concurrent systems by Timed Petrinets', Ph.D thesis, MIT, Sep.'73.
2. MK Molloy, 'On the integration of delay and throughput measures in distributed processing models', Ph.D thesis, UCLA, 1981.

3. J. Sifakis, 'Use of Timed Petrinets for performance evaluation', in 3rd Intl. Symposium on Measuring, modelling and evaluating computer systems, ed. Beilner & Gelenbe, Godesberg, 1977, NH-Pub., pp 75-95.
4. W.M. Zuberek, 'Timed petrinets and preliminary performance evaluation', Proc. 7th Annual Symp. Comput. Arch., 1980.
5. CA Sunshine et al, 'Specification and verification of communication protocols in AFFIRM using State Transition Models', IEEE Trans. SE-8, no.5, Sep.'82, pp 460-489.
6. J.V. Guttag, 'The specification and application to programming of abstract data types', Ph.D thesis, Dept. Comp. Sc., Univ. Toronto, Oct. 75.
7. ECMA, 3rd draft of Transport Protocol, ECMA/TC24/80/16 Feb. 1980.
8. G. Berthelot & R. Terrat, 'Petrinet theory for correctness of protocols', IEEE-T-COM-30, Dec.82, no.12, pp 2497-2505.
9. K. Garg & M Sloman, 'The Message Bus, an interconnection mechanism for a multicomputer', Research Report DOC 83/8, Imperial College, London, U.K., March 1983.
10. K. Garg, 'Formal specification and verification of the Message Bus protocol', Internal Report, Imperial College, London.
11. H.J Genrich & K Lautenbach, 'The analysis of distributed systems by means of Predicate/Transition Nets' in Semantics of Concurrent Computation (Lecture Notes in Comp. Sc., vol. 80) G Hahn, Ed., NY Springer Verlag 1978.
12. R Schwartz & PM Melliar-smith, 'From State Machines to Temporal logic: Specification methods for protocol standards,' pp3-20, in Proc. 2nd Intl. Workshop on Protocol Specification, Testing and Verification', Ed., C. Sunshine, 1982.
13. Proceedings of 2nd International Workshop on 'Protocol Specification, Testing and Verification' Ed. C. Sunshine, North Holland Pubs., 1982.
14. K. Garg, 'An approach to multiprocessor Performance analysis using TPNs', accepted for presentation at the International Conf. on Modelling Techniques and Tools for Performance Evaluation, Paris, May 16-18, 1984.
15. K Garg, 'Design and Performance Validation techniques for Distributed Systems using Timed Petrinets', Ph.D. thesis, University of London, in preparation.

(in Proc. International Conference on Modelling Techniques
and Tools for Performance Analysis, May, 1984)

**AN APPROACH TO MULTIPROCESSOR PERFORMANCE
ANALYSIS USING TIMED PETRINET MODELS**

K. GARG

**Department of Computing, Imperial College,
London SW7, U.K.
(on leave from University of Roorkee, India)**

We consider the performance analysis of a multicomputer system by developing and studying the Timed Petrinet models of two of its subsystems. One is the Communication Protocol used on the single, shared Bus which connects the various processors of the multicomputer. We give a technique to find the time taken to send a message correctly across the bus. The other is the Bus Contention subsystem. We discuss the implications of the analysis on the priorities to be allocated to each processor for Bus Arbitration, in order to maximize throughput on the Bus. The aim is to show that a certain class of Timed Petrinets provide the right tools for the performance analysis of typical multiprocessors

I. INTRODUCTION

Much interest has been shown in the last decade over the performance evaluation techniques for computer systems in general. Lately this interest has focussed on multiprocessors [1]. This is due to the rapid growth in the number of computer applications that can profit from the use of these structures.

Performance evaluation is concerned with studying, analyzing and optimizing the flow of data and control information in computer systems. Techniques for this include actual measurements, analytic methods, simulation, approximate solution algorithms, etc. Measurement can, of course, be done only on existing systems. For newly designed systems existing on paper only, measurement is intractable. So one can either do a suitable simulation, or alternatively, develop and analyze mathematical models to predict their performance.

In mathematical modelling techniques, the model devised captures the main factors which determine system performance. Performance measures or indices which have been previously defined and determined for the model are used as estimates of the performance of the system modelled [3].

A microprocessor based multiprocessor structure [2] has recently been proposed by us as a station (node) for Distributed Computer Control systems (DCCS). The main considerations for such a proposal have been to capitalize on such multiprocessor potential as modularity, extensibility and reliability. This is also in keeping with the rule given in [4] that a system should be so divided into functional parts that there exists maximum independence with well defined but simple interfaces. Another important criteria was to reflect as well as complement the modular structure of the CONIC Operating System being developed concurrently at Imperial College for such a DCCS [5].

Petrinets have been extensively used in the past for modelling computer systems. They have now become very popular for representing DCCS because of their capability of clearly describing concurrency, conflicts and synchronization of processes and because they present a simple yet elegant formalism for their description. Original petrinets were used to represent only the logical behaviour of systems, with no timing and hence no performance consideration. Over the past decade, however, the notion of time has been included into Petrinet models, making it possible to analyze the performance characteristics of such systems. Timed Petrinets (TPNs) were introduced by Ramchandani [6] by associating firing times to the transitions of ordinary Petrinets, in order to study their steady state behaviour. Since then many researchers have reported work on TPNs [7,8,9] giving rise to many different interpretations and extensions.

This paper discusses performance analysis techniques for two subsystems of the proposed Message Bus based multiprocessor. These are the communication protocol existing on the Message Bus and its Bus Contention system. TPN models of both are constructed and analyzed using the TPN theory as given by Sifakis [7]. In Sifakis's TPNs, fixed delay times are associated with the 'places' of the Petrinet.

In this Section we introduce the Message Bus multiprocessor architecture considered for performance evaluation and also the underlying theory of the TPNs used to model the behaviour of its subsystems. Section II is concerned with the performance analysis of the communication protocol used on the Bus and illustrates a technique for predicting the time taken to perform certain operations of the protocol. In Section III we discuss the TPN model for the Bus contention subsystem and show how it results in the calculation of the Message throughput and processor delays in Bus Acquisition. The implications of these results on the Bus Arbitration policy are then considered. Various possible extensions to the model are discussed. Conclusions are given in Section IV.

Architecture of the Message Bus Multiprocessor[18]

The architectural organization of the Message Bus based multiprocessor and its other salient characteristics are as follows, in so far as they are relevant to the present paper.

The various (upto 32) microcomputers are connected by a single, shared serial Bus called the Message Bus. Each microcomputer has a local (private) memory for its own use only. There is no other (shared) memory on the Bus. The architecture is shown in Figure 1, where it is clear that interprocessor communication is solely by message passing on the Bus.

The Bus arbitration policy used in the multiprocessor is a dynamic, distributed one, based on processor priority. Every potential sender on the Bus requests it by putting its Priority Byte on the Bus first and the highest priority processor gets the bus, if others are also requesting it.

Each processor is connected to the Bus via its Bus Interface (BI) unit which is responsible for requesting and acquiring the bus. The messages are stored in buffers for onward transmission on the Bus.

The complete error control protocol on the Bus is also implemented in hardware by the BI logic. It is an Immediate Negative, Positive, Acknowledgement and Retransmission (INPAR) protocol which provides a Reliable Datagram Broadcast service on the Bus. Its salient features will be discussed in Section II while drawing its TPN model.

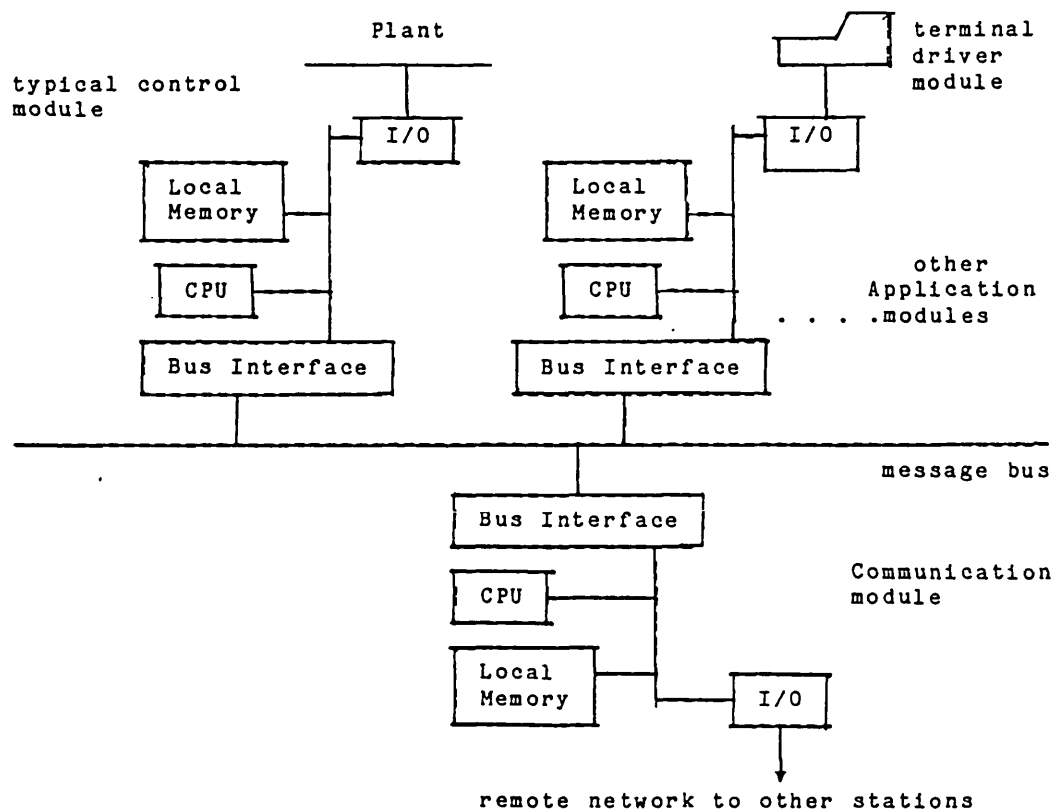


Fig. 1 The Message Bus multiComputer station

Various design choices were made for the Message Bus based multicomputer. These were mainly concerned with its structure, its Bus arbitration policy and mechanism, the type of error control desired, the facility for reliable broadcasting, etc. One of the main purposes of such performance evaluation was to prove conclusively and quantitatively that these choices were correctly made, as shown in [18].

Timed Petrinets Theory [7]

Sifakis has defined and represented Timed Petrinets as follows -

A Timed Petrinet (TPN) is a 6-tuple $N = (P, T, \alpha, \beta, \tau, v)$ where

P : is a set of places, $P \neq \emptyset$

T : is a set of transitions, $T \neq \emptyset$, $P \cap T = \emptyset$

α : $P \times T \rightarrow \mathbb{N}$ forward incidence function

$\beta : P \times T \rightarrow \text{IN}$ backward incidence function
 (IN represents the set of natural integers)
 $T : (\tau_1, \tau_2, \dots, \tau_i, \dots)$ an increasing sequence of real numbers
 called time base
 $v : P \times T \rightarrow T$ such that $\forall (p, \tau_i) \in P \times T : v(p, \tau_i) = \tau_j \Rightarrow \tau_j \geq \tau_i$

To a PN one can associate a digraph, the nodes of which are the places and the transitions, represented respectively by circles and dashes. There is a directed edge from the place ps to the transition t_j iff, $\alpha(ps, t_j) = n(s, j) \neq 0$

There is also a directed edge from transition tr to place pw iff,

$$\beta(pw, tr) = n(w, r) \neq 0$$

A marking M of a PN, shown by dots in the place p , is a mapping of P into IN,

$$P \xrightarrow{M} \text{IN}$$

Rules a) A marker in a TPN may be in one of the two following states : **available** or **unavailable**. Initially, each place p contains $M_0(p)$ available markers.
 b) A transition t is enabled iff every place ps , which has a directed edge to t , contains $\alpha(ps, t)$ available markers at least.
 c) The firing of a transition t has to take place instantaneously as it is enabled. It consists in removing $\alpha(ps, t)$ available markers from each of the places ps and in placing $\beta(pw, t)$ markers in each place pw which has a directed edge from t .
 d) A marker remains unavailable in a place ps during the time interval between the instant τ_i of its arrival and the instant $v(ps, \tau_i)$ when it becomes available. Thus each marker is delayed by a time z_i in the place p where

$$\forall (p_i, \tau) \in P \times T : v(p_i, \tau) - \tau = z_i = \text{constant}$$

For a TPN with n places, a vector $Q^t(\tau)$ is defined where

$$Q^t(\tau) = [q_1(\tau), q_2(\tau), \dots, q_n(\tau)]$$

and is called the charge variable at the instant τ . Each entry $q_i(\tau)$ represents the marking at p at the instant τ . For an interval $\Delta\tau = \tau - \tau_0$, the mean variation of the charge of the net is given by the vector

$$I(\tau) = \frac{Q(\tau) - Q(\tau_0)}{\Delta\tau} \quad \text{and is called the current vector.}$$

Each entry i_k of the vector I represents the mean frequency of firing of transition t_k during $\Delta\tau$.

Sifakis has shown that in order for the total charge in the net to remain bounded, constant currents are to be assigned to the transitions. This results in **periodic** firings in the net. The maximum computation rate corresponding to this periodic firing can be calculated by solving a set of n linear equations given as

$$C I = 0, \quad I > 0 \quad \dots (1)$$

$$\{ J_s Q(\tau_0) \geq J_s \sum_{k=1}^n C_{s,k} I_k \} \quad \dots (2)$$

where $s=1$

C = incidence matrix of the TPN

$$C = [c_{ij}]_{n \times m} \quad \text{with } c_{ij} = \begin{cases} \beta(p_i, t_j) & \text{if } \neq 0 \\ -\alpha(p_i, t_j) & \text{if } \alpha(p_i, t_j) \neq 0 \\ 0, & \text{if not} \end{cases}$$

$\{J_s\}^t$ is a generator of $\mathcal{L}$ where $\mathcal{L}$ is the set of non-negative solutions of $CX = 0$.
 $Q(t_0)$ is the initial marking vector
 and Z is the (square) delay matrix given by $\text{Diag}[z_1 \ z_2 \ \dots \ z_n]$

The net is said to function at a natural rate if the inequality (2) is changed to equality. The above linear equations can be found very easily if the TPN is decomposed into subnets. In this respect, Sifakis has shown that searching for nonnegative solutions of $CI = 0$ (resp. $J^t C = 0$) amounts to decomposing the TPN into consistent (invariant) components, which correspond to State Graphs (Marked Graphs).

II. PERFORMANCE ANALYSIS OF THE INPAR PROTOCOL

Performance analysis of a protocol is concerned with finding out if the protocol works in such a manner as to accomplish its job within a time acceptable to its higher level user service. Indeed, this should be the case for all possible conditions of the message transfer medium and for all sequences of transitions needed to do the message transfer correctly. Thus the time taken by the protocol for correct message transfer should be less than or equal to that allowed for the transfer by the user service. This is important in order to meet the real-time requirements of the system. It is also important that the time taken by the protocol be known, as it controls the buffer holding times and hence the buffer requirements in the Bus Interface [18].

In Distributed computer controlled real time systems, a certain 'information rate' is to be maintained in order for the system to be safely controlled [20]. This information rate depends not only on the actual data transfer rate on the medium, but also on the time taken by the protocol overheads, i.e., conversion of messages to packets, retransmissions and timeouts.

1. INPAR description

The INPAR protocol provides a 'Reliable Datagram' type of service on the Message Bus, i.e., it guarantees delivery of the correct message, or the user is notified of failure and duplicates are discarded. The ACKs (Acknowledgements) and NAKs (Negative Acknowledgements) are sent immediately after the message, hence the name of the protocol. It can be described as follows -

- * A packet-id (SPI) and a CRC-word is attached to each message sent
- * Since the medium is unreliable, the message may be lost or corrupted and not reach the receiver. Moreover, if there are no available buffers at the receiver, a correct message may still not be received at the receiver
- If the message is lost, no ACK is received at the sender, but this also amounts to receiving a NAK (called NAK3) from the receiver
- If the message is received but is found corrupted (by a CRC check) the receiver sends a NAK (NAK1)
- If no buffers are available with the receiver, it sends a NAK (NAK2)

- If all is well and a correct message is delivered to the receiver, an ACK is sent to the sender
- If an ACK is received by the sender, it releases the Bus, indicating that the transaction is over
- If a NAK1 or NAK2 or NAK3 is received by the sender, it retransmits upto 'n'times. If all retries fail, the sender 'gives up' and releases the bus, indicating end of a (unsuccessful) transaction

* The packet-id SPI is used to check if the message received is a duplicate. This is as done as follows -

- The receiver maintains a packet-id of its own (RPI) whose value is set to zero on every Bus Release signal. On reception of a correct message, the receiver checks the received SPI with its RPI
- If the value of RPI is zero, the message is considered new, is ACKed and delivered to the user. The value of RPI is then changed to that of SPI
- If RPI is not zero, but equal to SPI, the message is a duplicate. It is ACKed but not delivered to the user
- If RPI is neither equal to SPI nor zero, the message is considered new (the Bus Release signal is assumed lost in this case), the message is ACKed and delivered and RPI made equal to SPI.

2. The TPN model for the INPAR protocol

Figure 2 shows the complete TPN for the INPAR protocol. The places User, SenderToReceiver and ReceiverToSender have time delays Z1, Z2 and Z3 associated with them, representing the fact that a time delay is encountered by the protocol when it is in either of these places. These represent the time taken by the User to prepare a packet, to send the packet on the bus and to receive the NAKs/Ack on the bus, respectively. All other places are assumed to be instantaneous, ie., the markers in them become available as soon as they arrive at these places.

Some places are not labelled at all. These are dummy places needed only because each transition should be followed and preceded by places. The N arcs to the place 'Retry' from the transition ProtocolSend shows clearly the fact that the Retransmit transition will be fired only upto N times for every ProtocolSend transition; hence that only N retries are allowed in the protocol.

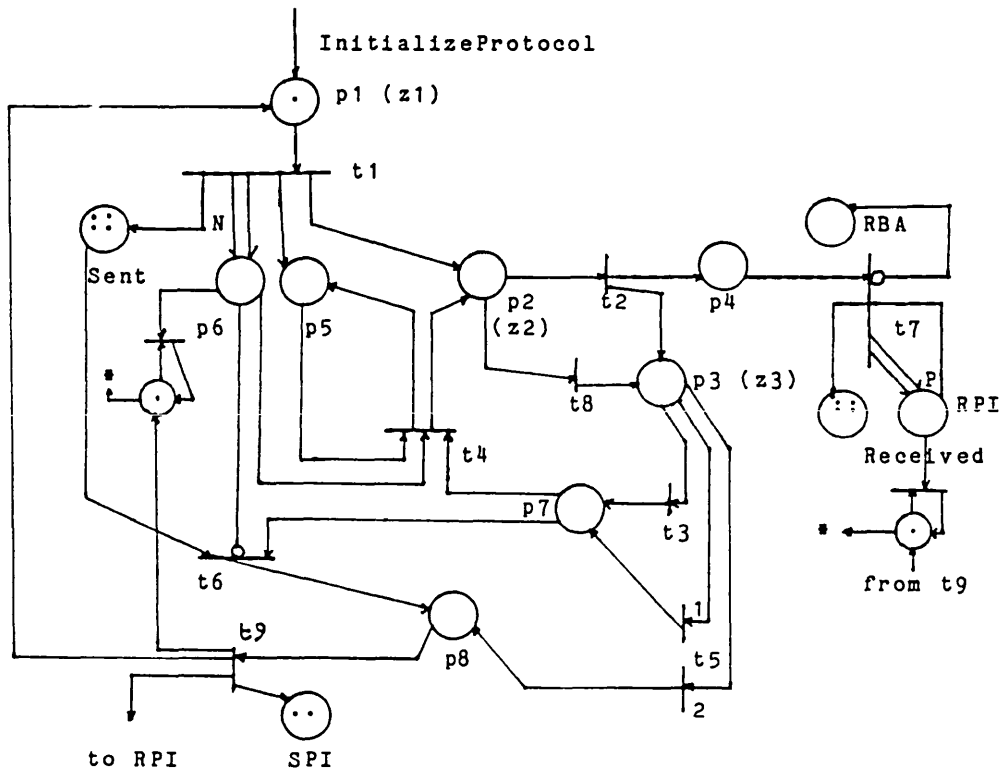
The tokens shown with some places describe their initial conditions (values) in each cycle of the protocol operation. (The P in the RPI place represents P markers.)

The inhibitor arc from the place Retry to the transition GiveUp shows that the GiveUp transition will fire only when there are no tokens left in Retry. This models the fact that if after N retries, the message is still not delivered correctly, the transaction is given up. Similarly, the inhibitor arc from RPI to Deliver models the fact that the transition Deliver fires only when RPI has zero tokens (a value of zero). Inhibitor arcs in Petrinets are used for 'Zero testing' [12]. They enhance the modelling power of the Petrinet. They shall not, however, be used for the performance analysis of the protocol.

The unlabelled place and transition on the left of Retry represents

the fact that the ReleaseBus activates this transition repeatedly to result in the removal of all remaining tokens from place Retry. This is also the case for the place RPI as shown. These have again been drawn only to model the conditions under which the protocol operates in order to transfer messages correctly.

Finally, the TPN as shown in figure 2 depicts clearly one complete cycle of the protocol from the transition ProtocolSend to ReleaseBus. We will see that such a cyclic representation makes it easy to specify the performance of the protocol.



p1 : User	t1 : ProtocolSend	t7: Deliver
p2 : SenderToReceiver	t2 : ReceiveRecognize	t8: Losepacket
p3 : ReceiverToSender	t3 : LoseByte	t9: ReleaseBus
p4 : ReceiverBuffer	t4 : Retransmit	
p5 : Pending	t5 : ReceiveByte	* to t1
p6 : Retry	t6 : GiveUp	

Figure 2. Timed Petrinet machine for INPAR protocol

3. Analysis of the INPAR protocol TPN model

In order to do a performance analysis of the INPAR protocol from its Timed Petrinet model, we shall redraw the net of figure 2 using only those places and transitions which affect its performance characteristics. The resultant net is shown in figure 3 where the

places and transitions have been labelled in a different way. The places Sent, Received, RPI, SPI, ReceiverBuffer and Pending and the transition Deliver are not drawn in Figure 3 as they do not affect the firing of the transitions and are in fact needed only to show that the protocol functions 'correctly'. Also in this figure we have coalsced transitions t2 and t8 and t3 and t5(1) into t2 and t3 respectively, since each pair has the same effect.

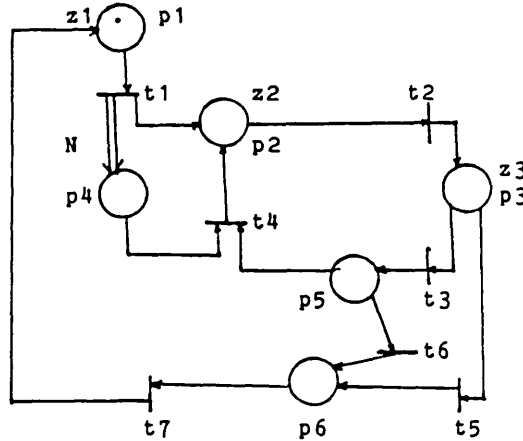


Figure 3. INPAR protocol Timed Petrinet for performance

For the TPN of Fig. 3 we have,

$$C = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & -1 & 0 & 0 \\ N & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 \end{bmatrix} \quad \begin{aligned} Q^t(\tau_0) &= [1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] \\ Z &= \text{diag} [z_1 \ z_2 \ z_3 \ 0 \ 0 \ 0] \end{aligned}$$

The equation $CI = 0$ has three possible solutions given by the vectors

$$I_1 = [1 \ N+1 \ N \ N \ 1 \ 0 \ 1], \quad I_2 = [1 \ N+1 \ N+1 \ N \ 0 \ 1 \ 1], \quad I_3 = [1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1]$$

The equation $J^t C = 0$ has only one solution, $J^t = [1 \ 1 \ 1 \ 0 \ 1 \ 1]$

$$\begin{aligned} \text{Thus } JQ &\geq JZC I_1 \Rightarrow 1 \geq [z_1 + (N+1)(z_2+z_3)].11 && \dots(1) \\ JQ &\geq JZC I_2 \Rightarrow 1 \geq [z_1 + (N+1)(z_2+z_3)].11 && \dots(2) \text{ and} \\ JQ &\geq JZC I_3 \Rightarrow 1 \geq [z_1+z_2+z_3].11 && \dots(3) \end{aligned}$$

It is to be noted that the vectors I_1 , I_2 and I_3 represent the three possible sequences of operations of the protocol and correspond to the operations of transferring a message with N retransmissions [I_1], without retransmissions [I_3] and no possible transfer (transaction given up) [I_2]. The subnets of the protocol TPN corresponding to these vectors are given in Figures 4 a-c respectively. Similarly, the subnet corresponding to the J vector is given in Figure 5.

The inequalities (1) - (3) give the maximum computation rates for the transition $t1$ ie., ProtocolSend in the three cases. Since functioning at natural rate is possible, we can rewrite these as

$$i1 = 1 / [z1 + (N+1)(z2+z3)] \text{ for } I1 \text{ and } I2 \text{ and}$$

$$i1 = 1 / [z1+z2+z3] \text{ for } I3$$

Alternatively, it can be said that the time elapsed between two ProtocolSend operations or the time taken to complete one cycle of the protocol in each case, whether a message is transferred or not is given by

$$T1 = z1 + (N+1)(z2+z3) \text{ for } N \text{ retransmissions and}$$

$$T2 = z1+z2+z3 \text{ for no retransmissions.}$$

The above equations can also be written down by inspection from Fig. 5 by simply adding the time delays of the places involved in each of the sequences I of figure 4.

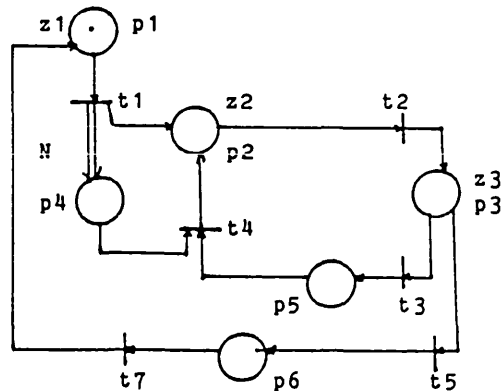


Figure 4a. INPAR protocol Subnet for sequence I1

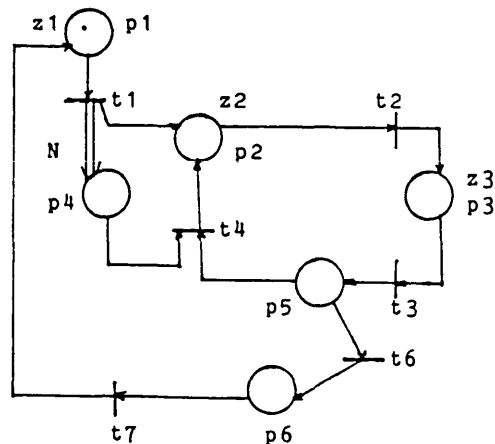


Figure 4b. INPAR protocol Subnet for sequence I2

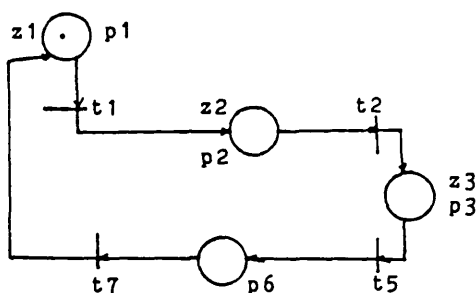


Figure 4c. INPAR protocol Subnet for sequence I3

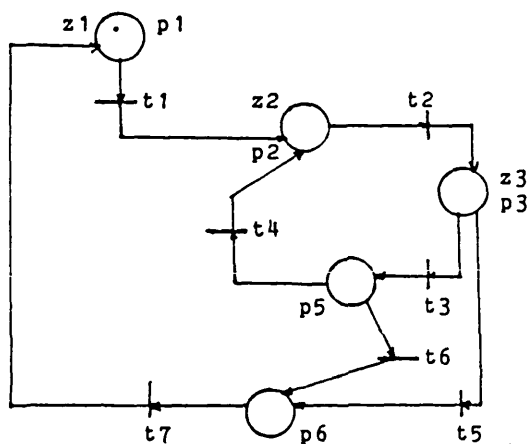


Figure 5. INPAR protocol Subnet for J

4. Extensions

(a) The foregoing performance analysis by means of a TPN model of a communication protocol, in terms of the time taken to complete certain operations of the protocol, can be directly applied to do a formal specification for it and hence to state and prove its performance properties [13,15]. Most researchers have up to now concentrated only on stating and proving **Safety** (correct delivery) properties for protocols. Of late though, there has been an awareness that **Performance** (fast delivery) properties are also important to specify [14]. But this has not been possible to do since the models used for representing the protocols had no time parameters.

It can be seen that with the use of Timed Petrinet models, performance characteristics are easily modelled and hence performance properties can be formally specified and proved. Indeed, we have been able to do such a specification for the INPAR protocol and state and prove two performance properties, one for each of the sequences I1 and I2 discussed above [18]. This is a natural extension of the above method of analysis.

(b) Again, the **Liveness** (eventuality of transfer) properties of a protocol can be easily modelled and represented using Timed Petrinets. This is because of the definition of firing of transitions in these TPNs. A transition has to fire as soon as all its enabling (input) places have available tokens, implying that there are no probabilities associated with its firing. There is, in effect, both a necessary and sufficient condition for a transition to fire.

We believe that such TPNs are thus ideal tools for representing eventuality of the protocol operations and should be used to specify its liveness. In this respect, TPNs compare favourably with Temporal Logic [16] which has hitherto been considered the only effective means of describing eventuality constraints for protocols. We have also specified and proved Liveness for the INPAR protocol [18].

(c) Multiple solutions of $J \leq 0$ We have used our method of performance analysis for analyzing other well known protocols and protocol standards. For example, we have used it to predict the times taken to complete the connection-disconnection phase of the Class I type ECMA protocol standard [19], assuming the time taken to send and receive data and control frames is known. This has been done for all possible conditions of operations of the connection initiator and acceptor.

One important difference has been found between the models of the INPAR and the ECMA protocols. For the INPAR model, there is only one solution of the equation $J \leq 0$. One reason for this is that every transition there has only one enabling input place with a delay. In general there may be many solutions of this equation as found in the ECMA model. In order to find the time taken to complete each operation as given by the sequences I of the model, we have to take into account all J subnets for each I. Each subnet J gives a different value for the total time taken. The actual time taken for each I sequence to be completed is then the 'largest' of the times taken for the individual subnets J. This, in fact, is the interpretation of the inequality (2) of Section I.

III. DEVELOPMENT AND ANALYSIS OF THE BUS CONTENTION MODEL

1. TPN model for Bus Contention

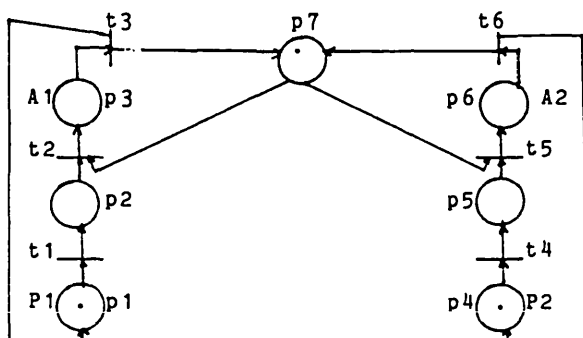
As mentioned earlier, Sifakis's TPNs can be used to great advantage in modelling for performance evaluation, the Bus contention subsystem of the Message Bus multiprocessor. For simplicity, we shall discuss a two-processor system first. Later, it will be shown how this can be easily extended to an n-processor system.

Figure 6 shows the TPN model of the Bus contention between two processors in a multiprocessor station to be used for DCCS. The places p1 and p4 correspond to the state of the processors when they are active, ie., **processing** some information. We assume that the time taken by each processor in this activity is P1 and P2 respectively. The values P1 and P2 depend on the computation time of the 'tasks' being performed.

The places p3 and p6 correspond to the states of the processors when they are **accessing** the Bus, ie., sending information.

Obviously, only one processor at a time can so access the Bus. We assume that the accessing time of the processors are A1 and A2 resp. The values of A1 and A2 depend on the message being sent on the Bus and also the overheads incurred due to the error-control protocol operating on the Bus. This time may change from access to access depending on the conditions in the system; but here we assume for simplicity that A1 and A2 are the 'average' access times. In [18] we show how the changing values of A1 and A2 affect our evaluation of the system.

The place p7 models the availability or nonavailability of the Bus at any instant. This is represented by the presence or absence of the marker in p7. The places p2 and p5 are 'dummy' places added in the TPN to complete the net between transitions t1 to t2 and t4 to t5 respectively.



p1, p4: Processing	t1, t4: message ready (Bus request)
p3, p6: Accessing	t2, t5: Bus acquired
p7 : Bus available	t3, t6: Access over
p2, p5: waiting	

Figure 6. Bus contention TPN model for 2 processors

The transitions t1 and t4 model the event Bus Request by the processors P1 and P2. These fire as soon as the processor has completed its processing and wants to send a message on the Bus next. The transitions t2 and t5 model the event Bus Acquired by the processors. The conflict between these transitions is clearly seen - only one will fire (if the bus is available) disabling the other. The transitions t3 and t6 correspond to Bus Release by P1 and P2. When any of these fire, the corresponding processor goes back to processing and the Bus becomes available.

Each processor may have to wait for a certain time between the Bus Request and the Bus Acquired events, if the Bus is not available due to its use by the other. It will be shown that this waiting time is constant for the processors when the TPN is in the steady state and can be kept to the minimum possible, for such DCCS as ours.

2. Calculation of Maximum computation rate

For the TPN of Figure 6, we have $Z = \text{diag} [z1 \ 0 \ z3 \ z4 \ 0 \ z6 \ 0]$, where

$z1 = P1$, $z3 = A1$, $z4 = P2$ and $z6 = A2$.

$$C = \begin{bmatrix} -1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & -1 & 1 & 0 & -1 & 1 \end{bmatrix} \quad Q^t(\tau_0) = [1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1]$$

Solving $CI = 0$, we find $I = [11 \ 11 \ 11 \ 12 \ 12 \ 12]$
and $J1 = [1110000]$ $J2 = [0001110]$ $J3 = [0010011]$

The solution of the equations (1) and (2) give the maximum values of $i1$ and $i2$ as satisfying the conditions

$$\begin{aligned} 1 &\geq i1(P1 + A1) && \text{..... (3)} \\ 1 &\geq i2(P2 + A2) && \text{..... (4)} \\ 1 &\geq A1.i1 + A2.i2 && \text{..... (5)} \end{aligned}$$

3. Interpretation of the maximum rate

The above calculation shows that each processor periodically processes and accesses the Bus and thus transfers a message at the maximum possible rate of $i1$ and $i2$ respectively. Thus the minimum time period for one such cycle is given by $Tp1 = 1/i1$ and $Tp2 = 1/i2$, where

$$Tp1 = P1 + A1 \quad \text{and} \quad Tp2 = P2 + A2.$$

The above 'minimum' signifies the fact that the actual periodic rate is larger than $Tp1$ and $Tp2$ and takes into account the 'waiting' times for each processor between requesting the bus and acquiring it in each cycle. Let us denote by T the time in which each processor can actually access the Bus once and let this be the common bus acquisition cycle time for the processors. Then, from the inequalities (3)-(5), we have

$$\begin{aligned} i1 &\leq 1 / (P1 + A1) &\Rightarrow i1 &= 1 / (P1 + A1 + d1) \\ i2 &\leq 1 / (P2 + A2) &\Rightarrow i2 &= 1 / (P2 + A2 + d2) \end{aligned}$$

such that $T = P1 + A1 + d1 = P2 + A2 + d2 \Rightarrow i1 = i2$

Then (5) $\Rightarrow 1 \geq A1 / (P1 + A1 + d1) + A2 / (P2 + A2 + d2)$

where $d1$ and $d2$ model the delay in getting the Bus in each cycle. Two messages in all are transferred on the bus in this cycle time T . It may also happen that the values of $P1$, $A1$, $P2$ and $A2$ are such that there can be found a slightly larger value of T in which one processor accesses the Bus only once, but the other (with a smaller value of $P+A$) can access it m times in T , $m > 1$. This gives rise to $m+1$ messages being transferred in the time T , and $(1+m) > 2$, with a slight increase in T .

Let us define the **Message Throughput Rate** Mt on the Bus as the total number of messages sent on the Bus in a cycle of time T . Then we can find the value of Mt in terms of the P and A times of each processor, using the relations given by the TPN. The delays $d1$ and $d2$ associated with each processor when it waits for the Bus to be acquired can also be found using the above equations.

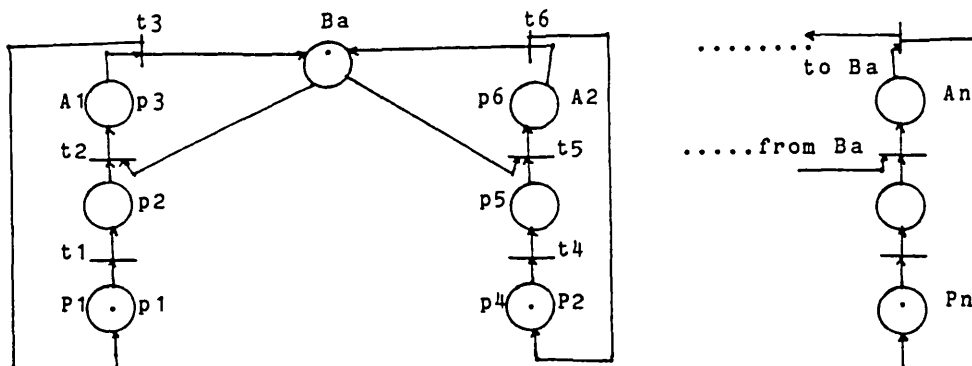


Figure 7. Bus contention model for n processors

The TPN model for the case of more than two processors is an easy extension of Figure 6 and is shown in Figure 7. It can be easily verified that it yields the following relationships between the processing and accessing times of the $n+1$ processors:

$i_0 \leq 1/(P_0+A_0)$, $i_1 \leq 1/(P_1+A_1)$, $i_2 \leq 1/(P_2+A_2)$, ...
and $i_n \leq 1/(P_n+A_n)$

$$A_0.i_0 + A_1.i_1 + A_2.i_2 + \dots + A_n.i_n \leq 1 \quad \dots\dots\dots(6)$$

where $i_0, i_1, i_2, \dots, i_n$ are the maximum possible rates for $P_0, P_1, P_2, \dots, P_n$ respectively.

Let T be the period of one complete cycle of bus acquisition and let it be defined in the general case as the time when each processor accesses the bus at least once. Let us also assume that the values of the P and A times are such that the processor with the largest value of $(P+A)$, designated P_0 , accesses once and the others access it $m_1, m_2, \dots, m_n$ times resp., i.e., $i_0 = i_1/m_1 = i_2/m_2 = \dots = i_n/m_n$.

Now $T = T_0 = P_0 + A_0 + d_0$ and let

$$T_1 = (P_1+A_1+d_{11}) + (P_1+A_1+d_{12}) + \dots + (P_1+A_1+d_{1m_1}) = m_1(P_1+A_1) + d_{10} \quad \dots (7a)$$

$$= (P_2+A_2+d_{21}) + (P_2+A_2+d_{22}) + \dots + (P_2+A_2+d_{2m_2}) = m_2(P_2+A_2) + d_{20} \quad \dots (7b)$$

etc.

where all $d_s \geq 0$ and $m_1, m_2, \dots, m_n \geq 1$

Hence (6) $\Rightarrow$ $A_0.i_0 + A_1.i_0 + A_2.m_2.i_0 + \dots + A_n.m_n.i_0 \leq 1$

$$\Rightarrow A_0 + A_1.m_1 + A_2.m_2 + \dots + A_n.m_n \leq (P_0+A_0+d_0) \quad \dots\dots(8)$$

The conditions (inequalities) (7) and (8) give the relations between the processor times P and A and the cycle time T and the m . Hence the Message Throughput Rate M_t is given by

$$M_t = (1 + m_1 + \dots + m_n)/T = (1 + \sum m_i) / T$$

The value of M_t as given by its definition can be maximized when T is minimum and m_2 and m_3 are maximum, provided the conditions (7)-(9) are satisfied. This can be shown [18] to be equivalent to the Knapsack problem in Dynamic Programming and we have solved it using a simple computer program for the case of 16 processors on the Bus. The graphical solution is given in Figure 8 which shows Bus access for each processor and their delays, given their P and A

times. The Message Throughput Rate is equal to $28/52$, i.e., 28 messages can be transmitted in 52 'time durations'. (Note that this is only a hypothetical value arrived at by considering arbitrary values for P and A and is no reflection on the performance of our multiprocessor).

4. Implications on choice of Bus Arbitration policy

The above results indicate a very important implication regarding the choice of the Bus Arbitration policy to be used in a shared Bus Multiprocessor. It shows that in order to maintain the highest possible message throughput rate on the Bus, it is necessary to maintain the cyclic acquisition shown by the graphical solution. In other words, each processor should be able to access the bus in the correct order in each cycle. The Bus Arbitration policy chosen for the multiprocessor should ensure that this cycle is followed even in the face of any contention that is likely to arise.

Usually for multiprocessors, the arbitration policy used is priority based. The priorities are allocated to the processors on some adhoc basis, usually message priority. The processor with the highest priority gets the Bus if there is contention, i.e., if more than one processor is requesting the Bus at that instant. No consideration is given to the processing and accessing times of the processors.

We believe that the priorities should be so allocated as to automatically result in the desired cyclic Bus acquisition. The cyclic order can be found from a graphical representation of the delays calculated for each processor and priorities then allocated to each. For example, in the case of Figure 8 the priorities p_1 to be allocated to the processors are

$p_5 > p_2 > p_1 > p_4 > p_0 > p_{10} > p_3 > p_8 > p_{11} > p_6 > p_9 > p_{15} > p_7 > p_{13} > p_{14} > p_{12}$

5. Extensions to the Model

The Bus contention model shown in Figure 6 was chosen because it best represents the characteristics of the Message Bus multiprocessor. The versatility of such a TPN model is easily demonstrated by showing that a change in the assumptions can be immediately represented with a slight modification of the Net, resulting in a different system being modelled. We will discuss two such extensions in the following paragraphs.

a) **Non-zero Bus Acquisition time** The place p_7 in Figure 6 was not allocated time delay, i.e., $z_7 = 0$ there. This models the fact that there is no delay in the next processor acquiring the Bus after the previous one has released it. This is truly representative of the situation on the Message Bus.

However, in some multiprocessors, eg., those that use a Central Arbiter for allocating the Bus amongst the different processors, a certain non-zero time is taken up for arbitrating the Bus. This is also characteristic of Local Area Networks like Ethernet [10], where the CSMA/CD mechanism takes a lot of time to acquire the network for the processor, due to collisions in a high load environment.

The above Bus characteristic is easily incorporated in the Model by letting $z7 > 0$. This changes the inequality (5) of the existing TPN model to

$$1 \geq \sum_{p=1}^n (Ap + z7).ip.mp \quad \text{for } n \text{ processors on the Bus.}$$

The net effect is to decrease the maximum allowable values of the computation rate for each processor. The time of one common cycle T increases by an amount equal to $n.z7$. Thus delays for each processor increase, bringing the message throughput rate down by a factor of $T/(T+n.z7)$. It is to be noted in this case that the Bus is idle for a time $z7$ after each release.

b) **Multiple Buses** The TPN model of Figure 6 allows a single marker in the place $p7$, representing the fact that there is only a single Bus available to the processors. Many multiprocessor systems have more than one Bus between the processors to increase its Reliability as in the case of the SIFT multiprocessor [11]. This fact can be easily represented in our TPN model by allowing the number of markers in place $p7$ to be greater than one. This number b will normally be less than the total number of processors n in the multiprocessor.

The inequality (5) is now modified to

$$b \geq \sum_{p=1}^n Ap.ip.mp$$

This means that the computation rates ip now approach more closely the maximum allowable value of $1/(Pp+Ap)$ for each processor p . The corresponding delays for each processor are considerably reduced. The message throughput rates have to be redefined in order to accomodate all the available buses in the system.

IV. CONCLUSION

Timed Petrinet models of two subsystems of a typical multiprocessor have been developed and analyzed. Timed petrinets, with fixed delays associated to places, are ideal tools for representing computer systems which are to be used in Distributed Computer Control systems. These can be classed as deterministic systems because of their behaviour, as opposed to the probabilistic or nondeterministic character of general purpose, user-oriented computer systems.

Performance characteristics of a communication protocol have been modelled and analyzed by applying Timed Petrinet theory. We have specified performance in terms of the time taken to complete certain operations of the protocol. There may be other ways in which performance could be predicted. As far as we are aware, ours is the first reported work on performance analysis of protocols using TPN models.

It has been shown that the eventuality of firing of transitions of a TPN results in the model being used also to formally specify and prove liveness as well performance properties of protocols.

We have given a novel way of applying Sifakis's TPN theory to result in a technique to calculate the least possible cyclic time

6483

period of Bus acquisition in which each processor accesses the Bus at least once. This results in the maximum achievable message throughput rates on the Bus for given values of processor processing and accessing times.

One of the main results obtained from the foregoing study is that the Bus Arbitration policy chosen for such a multiprocessor should be compatible with cyclic bus acquisition. As such it has been shown that the priority based Bus Arbitration mechanism of the Message Bus multiprocessor was the correct choice, since it allows such a cyclic bus access.

In contrast to the generally prevailing concept, it was found that TPNs do not necessarily lead to an explosion in the number of states, which complicates the mathematical analysis of large system models. Our three processor Bus contention model has only 20 possible distinct states. Even these play no part in the theoretical analysis of the model. Hence larger and more complex subsystems of multiprocessor structures can also be evaluated for performance using the above techniques.

We have used the techniques proposed in this paper to analyze the performance of the ECMA protocol as given in [19]. There we also show that the throughput maximization problem is equivalent to the well-known Knapsack problem of Dynamic Programming and have solved it for the case of 16 processors on a bus, using a simple computer program. Further we have analyzed the Message Buffering subsystem of our multiprocessor to find the number of buffers required. Then the three subsystems have been combined into a composite model to find the performance figures for the multiprocessor in terms of its Speed Up and Efficiency. Timed Petrinets allow such composite analysis to be done easily and elegantly.

Acknowledgements

The author is grateful to her supervisor Dr. M.Sloman for advice and encouragement. The work reported here was financed by a grant from the U.K. Commonwealth Scholarships Commission.

REFERENCES

1. Special issue on multiprocessor performance evaluation, IEEE-T-C-32, no. 1, Jan. 1983.
2. Garg, K. and Sloman, MS, 'The Message Bus, an interconnection mechanism for a multicomputer,' Research Report, Imperial College, DOC 83/8, March 1983, London.
3. Sauer, CH and Mani Chandy, K., 'COMPUTER SYSTEMS PERFORMANCE MODELLING,' Prentice Hall, 1981.
4. Spang, HA, 'Distributed Computer Systems for control,' 1st IFAC-IFIP Symposium on Software for Computer Control, SOCOCO 1976, pp 47-65.
5. Kramer, J et al, 'CONIC: an integrated approach to DCCS', IEE proc., Pt. E, vol. 130, no. 1, Jan 1983, pp1-10.
6. Ramchandani, C., 'Analysis of asynchronous concurrent systems

- by Timed PetriNets', Ph.D. thesis, MIT, Sept. 1973.
7. Sifakis, J., 'Petrinets for performance evaluation', Intl. Symp. on Measuring, Modelling and Evaluating Computer Systems, 1977, pp 75-93.
 8. Zuberek, W.M., 'Timed Petrinets and preliminary performance evaluation', in Proc. 7th Annual Symp. on Computer Arch., 1980, pp 88-96.
 9. Molloy, M.K., 'On the integration of delay and throughput measures in Distributed Processing models', Ph.D. thesis, UCLA, 1981.
 10. Metcalfe, RM & Boggs, DR., 'ETHERNET : distributed packet switching for local computer networks', CACM, July 1976, pp 395-404.
 11. Wensely et al, 'SIFT: Design and analysis of a Fault-tolerant computer for aircraft control', Proc.IEEE, vol. 66, no 10, Oct. 1978, pp 1240-1268.
 12. Peterson, JL., 'PETRINET THEORY AND THE MODELLING OF SYSTEMS', 1981, Prentice-Hall Pubs.
 13. Sunshine, CA et al, 'Specification and verification of communication protocols in AFFIRM using State Transition Models', IEEE Trans. SE-8, Sep.'82, pp 460-489.
 14. Proceedings of 2nd International Workshop on Protocol Specification, Testing and Verification', ed. C Sunshine, North Holland Pubs., 1982.
 15. Garg, K., 'Formal specification and verification of the Message Bus protocol', Internal Report, Imperial College, London, March 1983.
 16. Schwartz, R. & Melliar-Smith, PM., 'From state Machines to Temporal logic: Specification methods for protocol standards', pp3-20, Proc. 2nd Intl. Wksp. on Protocol specification, testing and verification, ed. C. Sunshine, NH-Pubs. 1982.
 17. ECMA, 3rd Draft of Transport protocol, ECMA/TC24/80/16 Feb. 1980.
 18. Garg, K., 'Design and Performance validation techniques for Distributed Systems using Timed Petrinets', Ph.d. thesis, University of London, in preparation.
 19. Garg, K., 'An approach to performance specification of communication protocols using timed Petrinets,' accepted for presentation at the 4th International Conf. on Distributed Computing Systems, May 1984, SF-USA.
 20. Prince, S. & Sloman, M.S., 'Communication requirements for DCCS', Proc. IEE, Pt. E, vol. 128, Jan. 81, pp 1-10.

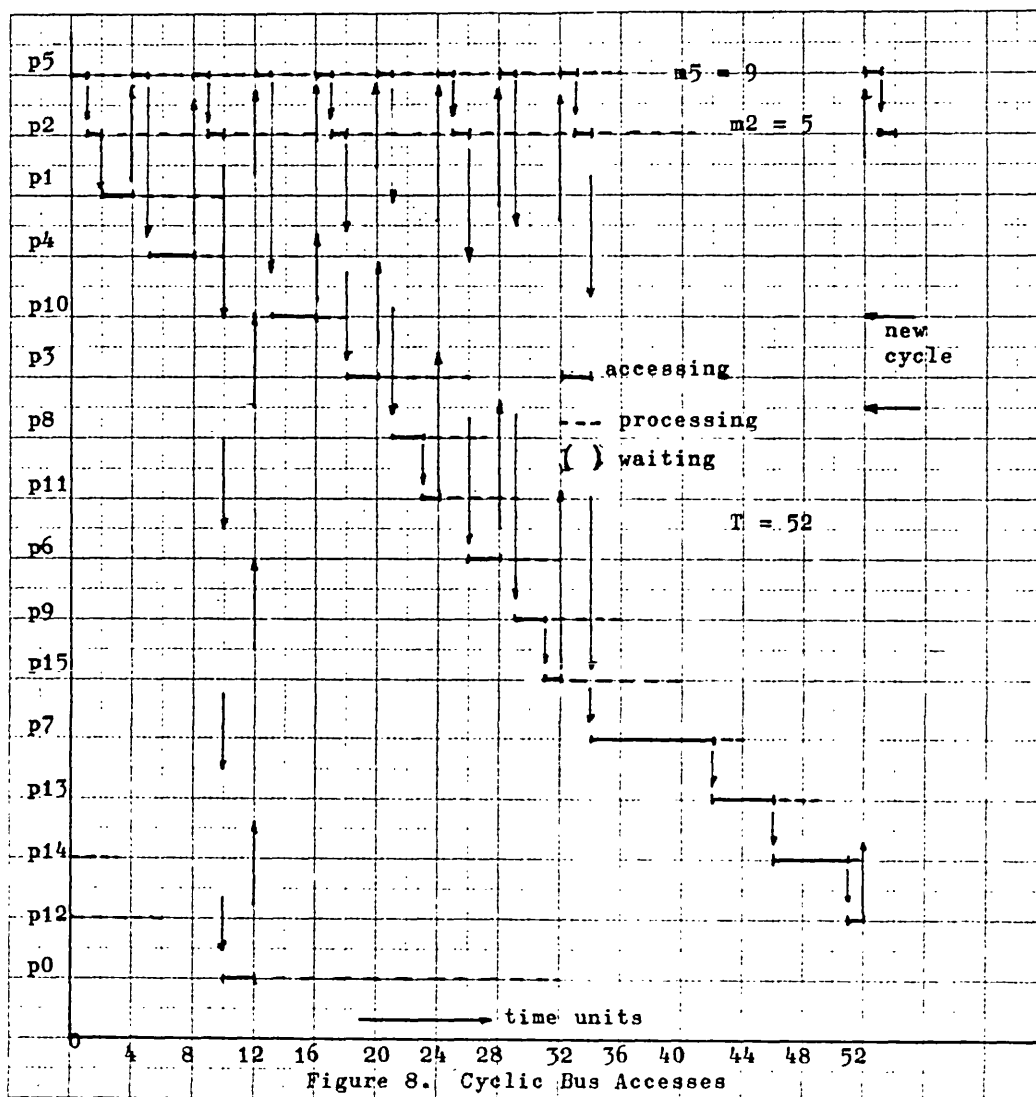


Figure 8. Cyclic Bus Accesses

**DESIGN AND PERFORMANCE VALIDATION TECHNIQUES
FOR DISTRIBUTED SYSTEMS USING TIMED PETRINETES**

by

KUMKUM GARG

A thesis submitted for the degree of Doctor of Philosophy
in the Faculty of Engineering in the University of London

Department of Computing
Imperial College of Science & Technology
London SW7.

DESIGN AND PERFORMANCE VALIDATION TECHNIQUES
FOR DISTRIBUTED SYSTEMS USING TIMED PETRINET

ABSTRACT

The thesis deals with the design and net based performance evaluation of a multiple microprocessor station for distributed real time systems such as process control. A loosely coupled, shared bus multiprocessor structure is proposed for such a station, based on an in-depth examination of its functional requirements. The communication protocol on the bus is designed for hardware implementation and provides a Reliable Broadcast facility for messages.

Evaluating performance of such real time systems has proved difficult. This thesis describes analytical performance evaluation techniques for distributed systems, based on net formalisms. In this respect, we have investigated and demonstrated the suitability of a class of Timed Petrinets to model the more deterministic characteristics of multiprocessor systems used in the process control environment. We have shown how Timed Petrinets theory can be used not only to evaluate performance but at the same time to validate the design of such systems.

The approach taken is to analyze models of certain subsystems of the multiprocessor first and then to consider its composite model, developed from these. This approach also proves that Timed Petrinets are ideal tools for large system performance evaluation because of their ease of use and flexibility.

Analysis of the bus contention system results in a novel technique, whereby priorities can be allocated to each processor for bus arbitration, in such a manner as to result in maximum message throughput. Results for a 16-processor system have been computed using a simple program.

We have integrated the use of automated formal specification and verification systems like AFFIRM, with Timed Petrinet models, to prove that the designed protocol satisfies its user requirements and exhibits properties not only of logical correctness, but also of performance commensurate with real time requirements. Consequently, a technique is proposed for the performance evaluation of protocols using their Timed Petrinet models. It is illustrated using the ECMA standard protocol.

It has been shown that the usual limitation of Petrinet modelling, ie., state explosion, can be overcome in a simple but efficient manner for performance considerations; also that Eventuality of protocol operations, and hence its Liveness, can be formally specified using Timed Petrinets.

The third and final subsystem considered is the message buffering system whose analysis gives quantitative results for number of buffers required and buffer holding times.

Since it is now possible to automate Petrinet analysis, these techniques can be viewed as basic tools in the Computer Aided Design of Large Scale Systems.

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my thesis supervisor, Morris Sloman, for his very useful guidance, suggestions and helpful criticisms throughout my period of work.

I also take this opportunity to thank Jim Cunningham for motivating me in the right direction at the right time, to John Darlington and Rajit Chandra for advice and assistance whenever sought, and to Jeff Kramer, Jeff Magee and Kevin Twidle, the other members of the Group, for help given from time to time.

I am grateful to my colleagues Orlando Loques, Trevor Moreton, Sebastian Danicic and Xenophon Andriopoulous for useful discussions; to my friend Sharmani Visuwanathan for listening sympathetically to my constant woes; and to my parents back home for their encouragement at all times.

My sincere thanks go to Ellen Haigh for her efficiency in obtaining Papers and Reports, whenever needed.

I also acknowledge the support - both financial and opportunity - of the Univeristy of Roorkee and the Ministry of Education in India, the Dept. of Computing at Imperial College and the U.K. Commonwealth Scholarships Commission, during my stay in London.

Finally, the last three years have been a difficult period for my family, to say the least, and I am eternally grateful to my husband and daughter for their wonderful cooperation throughout this period. This thesis is dedicated to them.

to NARESH and MOLLY

श्रेयान् द्रव्यभयाद्य ज्ञानज्ञानयज्ञः परंतप
सर्वं कर्माश्रितं पार्थ ज्ञाने परिसमाप्यते ॥३३॥
श्रीमद्भगवद्गीता चतुर्थोऽध्याय

(Sacrifice in the form of knowledge is superior to sacrifice performed with material things. For, all actions, without exception, culminate in knowledge.)

- Srimad Bhagavad Gita, Chapter 4, verse 33

CONTENTS

Abstract	(i)
Acknowledgements	(ii)
<u>Chapter 1</u> INTRODUCTION	1
1.1 Introduction and motivation	1
1.2 Objectives of the work undertaken	5
1.3 Overview and organization of thesis	6
<u>Chapter 2</u> THE MESSAGE BUS MULTIPROCESSOR DESIGN	8
2.1 Introduction	8
2.2 The CONIC Distributed Computer Control System	9
2.3 Message Bus Philosophy	11
2.3.1 The Message Bus concept	13
2.3.2 Functional requirements	14
2.4 Primary features	16
2.5 Error protocol for Message Bus	20
2.5.1 Reliable Datagrams	20
2.5.2 Errors on the broadcast bus	21
2.6 The INPAR protocol	22
2.6.1 Reliable broadcasts	25
2.7 General Discussion of the Message Bus	27
2.7.1 The Message Bus vs. Local Area Networks	27
2.7.2 The Message Bus and Ethernet	27
2.7.3 Performance characteristics of the Message bus	28
2.8 Comparison with other Multiprocessor buses and Bus standards	30
2.9 Conclusion	34
<u>Chapter 3</u> BUS CONTENTION SUBSYSTEM ANALYSIS	38
3.1 Introduction	38
3.2 Timed petrinets	39
3.2.1 Origin of Timed Petrinets	39
3.2.2 Choice of Timed Petrinets	40
3.2.3 Timed Petrinet theory	41
3.3 Development and analysis of the bus contention model	43
3.3.1 Timed petrinet model	43
3.3.2 Calculation of maximum computation rate	45
3.3.3 Interpretation of the maximum rate	46
3.4 Technique for maximizing Message Throughput	47
3.5 Discussion	51
3.5.1 Implications on choice of bus arbitration policy	51
3.5.2 Implementation considerations	52
3.5.3 Extensions to the model	53
3.5.4 Performance prediction of the multiprocessor	55
3.5.5 Discussion of assumptions	56
3.6 Comparison with related work	58
3.6.1 Stochastic TPN models for multiprocessor bus contention	58
3.6.2 Analysis of Real Time multiprocessors systems	59
3.6.3 Time valued petrinets for control algorithm decomposition	60

3.7 Conclusion	61
 <u>Chapter 4 FORMAL SPECIFICATION AND VERIFICATION INPAR PROTOCOL</u>	
4.1 Introduction	65
4.2 The AFFIRM system	68
4.3 The Message Bus service specification	69
4.4 The INPAR protocol	75
4.4.1 Informal description	75
4.4.2 The TPN model for the INPAR protocol	76
4.4.3 TPN Markings and control state	78
4.4.4 Timeelapsed and control state	80
4.5 INOAR protocol specification	81
4.5.1 Data types used in the specification	81
4.5.2 AFFIRM representation of the protocol specification	85
4.6 The INPAR protocol verification	85
4.7 Discussion and conclusion	87
 <u>Chapter 5 PROTOCOL LIVENESS AND PERFORMANCE</u>	
	90
5.1 Introduction	90
5.2 Performance analysis of INPAR protocol	91
5.2.1 Reduction of Timed petrinet	91
5.2.2 Formal specification of protocol performance	95
5.3 Proving Liveness for INPAR protocol	97
5.4 An overview of our method for performance specification	98
5.5 Performance specification of ECMA protocol	
5.5.1 Connection and Disconnection phase and its TPN model	100
5.5.2 Analysis of the TPN model	103
5.5.3 Formal specification of the ECMA protocol	105
5.5.4 Statement of ECMA performance	110
5.6 General Discussion	
5.6.1 Liveness properties	111
5.6.2 Timeouts	112
5.6.3 Control versus data	112
5.6.4 Explosion of states	112
5.6.5 Deterministic model	113
5.6.6 Related work	113
5.7 Conclusion	115
 <u>Chapter 6 MESSAGE BUFFERING SUBSYSTEM AND COMPOSITE MULTIPROCESSOR ANALYSIS</u>	
	117
6.1 Introduction	117
6.2 TPN model for sequential buffer access	118
6.3 TPN model for simultaneous buffer access	120
6.4 Composite TPN model for a multiprocessor system	122
6.5 Performance evaluation of multiprocessor	124
6.6 Comparison with related work	127
6.6.1 TPNs with fixed transition firing times	127
6.6.2 Asynchronous concurrent real time systems	128
6.6.3 Time driven systems	128
6.6.4 Comparison with PERT/CPM scheduling procedures	129
6.6.5 Comparison with Queueing Network models	130
6.7 Conclusion	133

Chapter 7 CONCLUSION

136

- 7.1 Summary and contributions
- 7.2 Directions for further work

136
140

REFERENCES

143

- APPENDIX I Knapsack program for a 16-processor system
- IIa Formal specification of INPAR in AFFIRM
- b Correspondence between service and protocol
- c The theorems

151
152
161
163

List of symbols

(vii)

List of symbols used with Timed Petrinets

P	the set of places in a Petrinet
T	the set of transitions in a Petrinet
ϕ	the null set
$\mathbb{N}$	the set of integers
α	forward incidence function in a Petrinet
β	backward incidence function in a Petrinet
M	the marking of a Petrinet
τ	a time base for Timed Petrinets
τ_i	the time instant of arrival of token at a place p_s
$v(p_s, \tau_i)$	the time instant at which the token becomes available at p_s
z_1	the time delay associated with place p_1
$Q(\tau)$	the marking of the Timed Petrinet at time τ
C	the incidence matrix of the Petrinet
C^+	submatrix of C with positive elements only and all others zero
$R(\tau)$	the firing vector of the Timed Petrinet at time τ
$I(\tau)$	the current vector of the Timed Petrinet at time τ
Z	the delay matrix of the Timed Petrinet
J	a place invariant of a Petrinet

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION AND MOTIVATION

This thesis is concerned with two comparatively new, but at the same time distinct, areas of Computer Science, viz., Timed Petrinets theory and distributed, multiple processor systems. It deals with the application of Petrinet based techniques to the design and performance validation of distributed multiprocessor systems.

It is generally agreed that the price-performance revolution in micro-electronics, on the one hand, and the development of efficient and cost-effective interconnection structures, on the other hand, have been the driving force for distributed computer systems [Davies81].

In the Real Time control industry, distributed, multiple computer systems are being increasingly used to give greater extensibility, availability and performance. Examples of such distributed systems are airplane/missile guidance systems, laboratory automation systems, public utility systems like power stations, transport control, electricity, gas distribution etc., and process control systems. Multiple microcomputers are physically distributed around the plant or apparatus they control, forming processing sites (nodes) or **stations**. Stations communicate with each other in order to cooperate and coordinate their activity and this is done through an interconnecting medium.

The performance of the distributed system is affected as much by the processing power of the microcomputer based stations as by the efficiency of their interconnecting medium.

Traditionally, a Distributed Computer Control System (DCCS) station has been designed as a uniprocessor structure, with the single processor performing all control (application) functions as well as the communication ones. But it is becoming increasingly clear that in order to provide reliability and extensibility together with processing power, a multiprocessor type structure is required for the station. This means, in effect, that a second level of 'distribu-

tion' is introduced in our already distributed system, allowing each of its stations to consist of interconnected microprocessors.

This thesis is concerned with the design and performance validation of such a multiple microprocessor station in a distributed real time system. The motivation for this work came from the association of the author with the Distributed Systems Group at Imperial College's Department of Computing, where the CONIC system has been designed and developed for DCCS [Kramer82].

Each CONIC station is presently implemented as a single processor containing multiple software modules. The main objective for the design of a station based on multiple heterogeneous processors was to allow a CONIC software module to be implemented in microprocessor firmware, but still retain the configuration flexibility exhibited by the current system. A station could then be configured from a set of standard hardware or software CONIC modules, to suit the application requirement. This would also increase the processing power at the station. Chapter 2 of this thesis discusses the decisions made during the design of a Message bus to support communication between CONIC hardware modules.

It soon became apparent that it would be very difficult to validate the design and performance of the multiprocessor without the use of formal methods, or with the tools and techniques that are available for conventional uniprocessor systems. For example, while it is possible to consider most qualitative aspects of the design of a distributed, multiple processor system informally, certain quantitative design parameters like the number of buffers required in the Bus Interface or the number of allowed retransmissions in the protocol, cannot be ascertained accurately. Also, there was a need to verify that the bus protocol functioned correctly. Similarly, the performance of the multiprocessor could not be predicted in depth without some means of gauging the effect of its complex interactions and behaviour patterns. These need more formal tools and techniques.

The need for more sophisticated techniques for multiprocessors has been recognized by several people [Agarwa79, Wietzm82, Agarwa83]. Consequently, much interest has been focussed in recent years on the design and performance validation of multiprocessor systems. This thesis is a logical outcome of this interest.

Functional design verification for computer systems has been formalized, for example, with interactive systems like AFFIRM [Sunshi82b] which aid in formally specifying system requirements and in verifying that these are met by the design. We have used AFFIRM to verify the functional correctness of our bus protocol.

Performance validation is synonymous with performance evaluation which, according to Sauer and Chandy [Sauer81], 'is concerned with studying, analyzing and optimizing the flow of data and control information in computer systems'. The main objectives for this are

- (i) to formally describe the real behaviour of the system being studied,
- (ii) to define and determine characteristic performance measures,
- (iii) to develop tools for the design of optimal hardware or software structures.

Performance evaluation techniques include actual measurements, analytic methods, simulation, approximate solution algorithms, etc. Measurement can, of course, be done only on existing systems. For newly designed systems existing on paper only, measurement is intractable. So one can either do a suitable simulation, or alternatively, develop and analyze mathematical models to predict their performance.

While simulation is an attractive method of evaluating a system, it has traditional shortcomings, as listed by Sauer and Chandy [Sauer81]. These are concerned mainly with the expense of constructing a simulation program and with the computational cost of running it.

This is certainly not true in the case of mathematical models which are also simpler to derive and are more flexible. Hence we chose this method.

Our requirements for a suitable performance modelling tool were fourfold. These are as follows-

- (i) it should be deterministic,
- (ii) it should be able to model the inherent parallelism of distributed systems,
- (iii) it should be applicable to individual subsystems as well as a

composite system, and
(iv) it should allow validation of design choices as well.

As regards the first requirement, both deterministic and nondeterministic models are in use today. Most nondeterministic modelling is based on the theory of Markovian Stochastic processes, for eg., Markov Chain models and Queueing Network models, which have been extensively used since the past two decades to model large and complex computer systems [Sauer81, Lavenb83].

However, we concentrate here on deterministic models only as these are characteristic of real time computer system used for eg., in process control. Such distributed systems require very small delay variations in the processing of messages and in their communication between processors. We discuss this in greater detail in Chapter 3.

The other three requirements are adequately met by Timed Petrinets. Petrinets have been used in the past for modelling computer systems. They have now become very popular for representing DCCS because of their capability of clearly describing concurrency, conflicts and synchronization of processes and because they present a simple yet elegant formalism for their description. There are many other advantages of using Petrinet modelling, as given in [Ozsu82, Reisig83]. The research on which this thesis is based involves the development of analytical design and performance evaluation methodologies based on net formalisms.

Original petrinets were used to represent only the logical behaviour of systems, with no timing and hence no performance consideration. Over the past decade, however, the notion of time has been included into Petrinet models, making it possible to analyze a system's performance characteristics.

Timed Petrinets (TPNs) were introduced by Ramchandani [Ramcha73] by associating time with Petrinet transitions. Since then many researchers have reported work on TPNs [Sifaki77, Zubere80, Molloy81] giving rise to many different interpretations and extensions.

We have chosen TPNs as defined by [Sifaki77] to model all subsystems of our multiple processor station, because we feel that these TPNs, with their fixed delays associated to 'places' are most representative

of real time process control systems, as discussed in Chapter 3.

The thesis reports on novel applications of such Timed Petrinets for the design and performance validation techniques for multiprocessor systems using the example of the proposed multiprocessor station. The approach taken is to identify various subsystems of the multiprocessor, which exhibit concurrency, contention and synchronization. These are the bus contention subsystem, the communication protocol and the message buffering subsystem. TPN models of these are developed and analyzed separately at first to validate certain design parameters and also to evaluate performance.

A composite model of the complete multiprocessor is then drawn by combining the individual models, to see how each subsystem affects the other and also the overall system performance, which is thus obtained in more detail. Some other very interesting conclusions can be made from such analysis and these are discussed separately for each subsystem in the relevant chapters. For example, we have shown how to integrate the use of systems like AFFIRM with Timed Petrinet models to prove that the designed protocol satisfies its user requirements and exhibits properties not only of logical correctness, but also of performance commensurate with real time requirements.

1.2 OBJECTIVES OF THE WORK UNDERTAKEN

The objectives addressed in the thesis are -

- * to propose and design a suitable multiprocessor type architecture for a station in a Distributed Computer Control System;
- * to validate the design and performance of the proposed multiprocessor, using formal modelling methods;
- * to develop some techniques based on these for multiprocessor systems in general;
- * to demonstrate that the chosen method provides the right tools for such validation.

1.3 OVERVIEW AND ORGANIZATION OF THESIS

The thesis is organized as follows -

In Chapter 2 we present a proposal for the design of a multiprocessor type architecture for a DCCS station, based on an in-depth study of its functional requirements. The concept and philosophy behind the bus are discussed. We discuss the various design choices available for it and the decisions made therein. The communication protocol designed to ensure error-free message passing on the bus is given next. We show how it achieves reliable broadcasts on the bus in a simple but unique manner. The differences between the Message Bus and Local Area Networks and other multiprocessor bus standards are presented. Some preliminary performance characteristics for the Bus are also included in this chapter.

In chapter 3 we trace the origins and present the theory of Timed petrinets as given by Sifakis. We also discuss why we have chosen his Timed Petrinets for modelling purposes. We then discuss the bus contention subsystem. It is modelled using Timed Petrinets and its analysis leads to a procedure to calculate the maximum achievable message throughput of the system. It also provides a technique for allocating priorities to processors while ensuring this maximum. We discuss multiprocessor Speed Up and Efficiency based on the above maximization technique. Possible extensions to the technique and related work are also considered.

Chapter 4 is concerned with the formal specification and verification of the INPAR protocol designed in Chapter 2, using its Timed Petrinet model. It serves to validate the functional design of the protocol. The methodology followed is basically that of AFFIRM but uses suitable extensions for incorporating TPN models.

In Chapter 5 we present the **performance evaluation** of the INPAR protocol using Timed Petrinets. The technique is shown to apply equally well to other communication protocols such as the ECMA transport layer protocol. Liveness considerations are also presented for the protocol. We discuss our technique in terms of possible extensions and shortcomings and report on some related work in this area.

Chapter 6 deals with the performance analysis of the third and final subsystem of the multiprocessor, ie., the message buffering subsystem. This is modelled as the classical producer-consumer problem and two different cases are considered, related to the type of memory used for the buffer. Some interesting results on design and performance are obtained. This chapter also sees the development and analysis of a composite Timed Petrinet model of the complete multiprocessor and the ease with which this is achieved using TPNs is emphasised. Some realistic and detailed quantitative results are obtained for the performance of the whole system, with respect to Speed Up, Processing Power and Efficiency. We discuss some related work on multiprocessor performance and also compare TPN models to PERT-CPM charts and Queueing network models.

In chapter 7 we present a detailed summary of the work reported in the thesis while emphasising the contributions made therein. We also give some ideas and directions for further research work in related areas.

Appendix I presents a computer program for the 'Knapsack' type maximization problem considered in Chapter 3. Appendix IIa gives the complete formal description of the INPAR protocol in AFFIRM using Timed Petrinets. In Appendix IIb & c, some related theorems concerned with its verification against this specification and with its Safety property are given.

CHAPTER 2

THE MESSAGE BUS MULTIPROCESSOR DESIGN

2.1 INTRODUCTION

We have entered a new era in which Distributed Computing Systems are being driven by rapidly expanding computer software and hardware technologies. The cheap availability of powerful 8- and 16- bit microprocessors has led to their increasing use in the process control industry. They are connected to sensors and actuators placed close to the equipment being controlled - and often physically distributed around the plant, forming a network of such microprocessor-controlled sites or **stations**. Stations communicate with each other by information transfer through this network.

The design of a station in any Distributed Computer Control System (DCCS) is dependent largely on the control (application) function to be performed by it. These may range from simple monitoring and sequence control to fairly complex computations. A station in a DCCS is also required to perform communication functions in order to communicate in a reliable manner with other stations in the system.

Traditionally, a DCCS station has been implemented as a single processor station, with the processor performing all application and communication functions. But it is now becoming clear that in order to achieve desirable properties like modularity, extensibility and reliability in a single station, a multiprocessor type structure is needed.

Recognizing this need, a proposal has been made for the design of a multicomputer station, in which multiple heterogeneous microcomputers would be interconnected by a simple, shared bus called the **Message Bus** which would in turn fulfill all the requirements of interprocessor communication within the station. Rapid advances in VLSI technology have encouraged us to design a multiprocessor structure that blurs even further the already thin dividing line between computer and communication.

In this chapter we present the philosophy and concepts behind the

Message Bus design and its functional requirements as an efficient interconnection medium for a multicomputer station. We start by examining the salient features of the CONIC distributed system [Sloman82] for which the multiprocessor station was to be designed. An in depth study of the requirements leads to the primary features of the bus, including a discussion of the various design decisions. We present also the communication protocol designed for error-free message passing on the Bus and show how, by using this protocol and the Wired-AND implementation of the bus, it is possible to achieve **Reliable Broadcasts** across it. We explain how the Message Bus differs from Local Area Networks and also from other existing multiprocessor buses and Bus Standards. Some preliminary performance characteristics of the Bus are included.

This chapter brings to light the fact that while it is possible to informally consider most qualitative aspects of the design of a distributed multiple processor system, certain other (quantitative) design parameters cannot be ascertained and need the use of more formal methods.

2.2 THE CONIC DISTRIBUTED COMPUTER CONTROL SYSTEM

The CONIC project, under way at Imperial College, provides a unified and integrated approach to the **design, implementation and management** of large DCCS. It consists of: a **network architecture**, which permits the interconnection of large numbers of computer stations; a **software methodology**, which supports the design of application systems as a set of interconnected re-usable components; a **distributed operating system** and **communication system**, which provide run-time support for application software; and a **management system**, which enables a system to be tailored from a set of components and subsequently modified and extended to meet changing application needs. These entities of the CONIC system are shown in Figure 2.1.

The major objective of the CONIC software architecture was to separate the concerns of writing individual software components (programming-in-the-large) from those of constructing or configuring a system from a set of components (programming-in-the-small). Such a separation allows components to be programmed without knowledge of the configuration in which they would be used; and also allows a configuration to be modified without re-compilation of its constituent components.

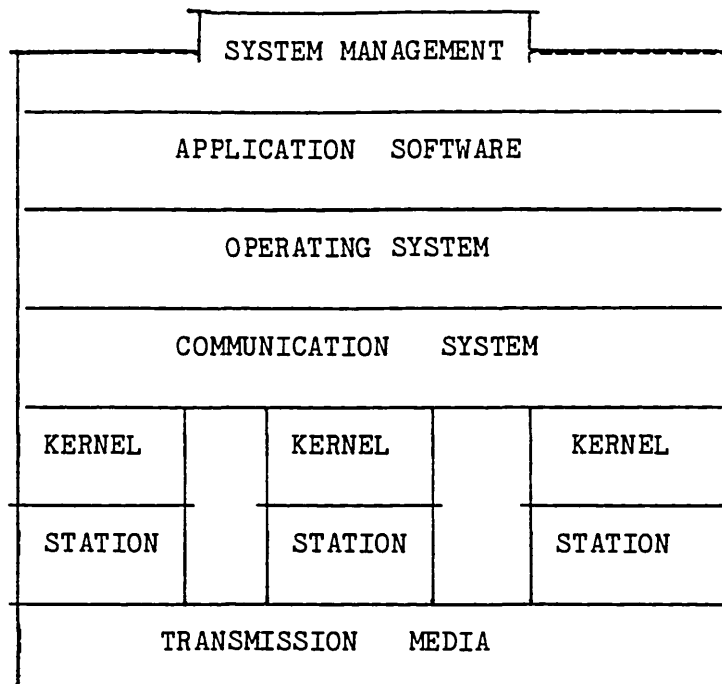


Figure 2.1 The CONIC System Architecture

Software components A software component is executed at a station and implements some subfunction of the overall control function. Components can communicate solely by exchanging messages and hence their interfaces can be defined solely in terms of the messages they transmit and receive. They must be able to deal with communication failures and failures in other components of the system. This is provided by the loose coupling of stations by serial data links in a DCCS, which considerably enhances the system hardware robustness by confining the effect of the physical fault to one station.

Modules In order for the software architecture to mirror the underlying structure of microprocessor stations interconnected by a communications network, CONIC has proposed the **module** as the software abstraction of a station. This provides the link between programming activities and those of system configuration. The module is the smallest software component which can be distributed or replaced. A station may have more than one module.

Modules communicate by message passing in order to cooperate and synchronise their actions. The interface which a module presents to the rest of the system is given by the messages which can be sent and received by the module. In CONIC this interface is characterised by sets of **exit** and **entry ports** respectively. An entryport may be

thought of as a 'hole' through which messages can pass into the module; an exitport as the 'hole' through which messages are sent out of the module.

This message passing is implemented in the operating system kernel which therefore allows software modules to coexist and communicate within a processor.

The association between exit and entry ports, called **linking**, provides a communication path which can be established during system installation and reestablished as necessary during system modification. Port linkage may be between one entryport and one or many exitports, or between many exitports and one entryport, in order to provide the three types of interconnection structures mentioned below.

Configuration Configuration is the construction of a control system from a set of components. It involves the assignment of these components to the computer stations on which they run and their interconnection for communication. Three interconnection patterns are used in CONIC.

- * one-to-one - a single or return path between two specific components
- * one-to-many - a broadcast path between one sending component to many receiving components
- * any-to-one - one at a time access by many components to a server component.

2.3 MESSAGE BUS PHILOSOPHY

The initial requirement that led to the Message Bus was the need to develop a hardware architecture to support the software structure of the CONIC station. It was thought that the capability of implementing each software module in a hardware module was appealing. It would then be possible to interchange hardware and software components transparently. In other words, a software module could be implemented by a separate microcomputer or hardware module. It is also possible that more than one software module reside on a single hardware module.

Depending on requirement, such a hardware (processor) module can

simply be plugged in or out of the station in order to change its processing power. For example, we could add a NSC800 to perform a particular control function or a 6805 to handle a particular I/O. This seems a very attractive and easy way of including or removing terminal handlers or virtual circuit implementing transport modules in the station.

In fact, the need for a separate Communications Processor(CP) are many and listed below [Garg82] -

- * Communications is a complex and time-critical function. Some application functions are also time-critical and the multiprocessor structure prevents interference.
- * A separate dedicated network protocol processor gives the application processor(s) or APs the necessary independence from the network characteristics.
- * It also resolves the problem of portability of the transport protocol software between APs of different makes. The CP can be considered a standard 'black box' identical for all applications.
- * A separate CP is being increasingly used in all office automation and other data communication systems.

Similarly, there are many advantages of having a multicomputer type station for the DCCS and these are discussed as follows -

- * Multicomputers are known to exhibit the desirable characteristics of functional modularity, extensibility and fault isolation.
- * The different processors in the station can now share power supplies. This is a very desirable feature in process control systems which require Intrinsically Safe (IS) hardware and the cost of a single power supply can be very high. On a LAN it is not possible for processors to share power supplies because of the distances involved. This greatly increases their number and hence their cost.
- * Separate computers grouped together within a station are still able to maintain their close interaction as desired by the control function, even though they are independent.

The decision to have a multicomputer station naturally led to the need for an efficient communication for the hardware modules and thus to the Message Bus. Various existing multiprocessor buses and a few Bus

standards like MULTIBUS [AP28 79], I2C [Goldbe82], EUROBUS [Biscoe81], IEEE Serial P896 link [Kirman81], HXDP bus [Jensen78], COBUS [Sommer76], GPIB 488 bus [Osborn79] and VME [Motoro82] were examined for their possible use in our multicomputer station.

It was found that while no single Bus was able to satisfy wholly the requirements we have laid out for our Bus, a few, like the IEEE P896, the HXDP and the I2C had some useful features that have been used to advantage in our multiprocessor. The main considerations for comparison were regarding speed, bus access mechanism, provision for connecting heterogeneous processors, no shared memory and facility for broadcast and multi-destination message passing. We will review these various buses later in this chapter.

2.3.1 The Message Bus concept

The Message Bus can be thought of as doing for the hardware modules what the operating system kernel does for CONIC software modules. Thus the message bus can be thought of as a hardware kernel which allows multiple hardware modules to

- (i) coexist in the station, and
- (ii) communicate with each other.

Figure 2.2 shows a typical Message Bus based multicomputer station. As can be seen, there is no shared (common) memory between the various computers on the Bus and communication is solely by message passing on the Bus.

The existence of message bus based multicomputer stations in a DCCS gives rise to a three-level system of communication in it. The lowest or zero-level layer is the local communication between modules resident in a single computer. The middle or level-1 layer corresponds to communication on the message bus between modules resident on the various computers in the station. The highest or level-two layer comprises remote communication i.e. between modules residing in different stations.

A major feature of the message bus design addresses the requirement of coordinating the three levels of communication in the DCCS to provide for efficient and speedy message passing between modules resident anywhere in the system.

It is to be noted that all stations in the DCCS need not be multicomputers. Some stations may be uniprocessors. The internal structure of a station is not known to other stations on the network, ie., it is not possible to know from outside, whether a particular station is a uniprocessor or multiprocessor.

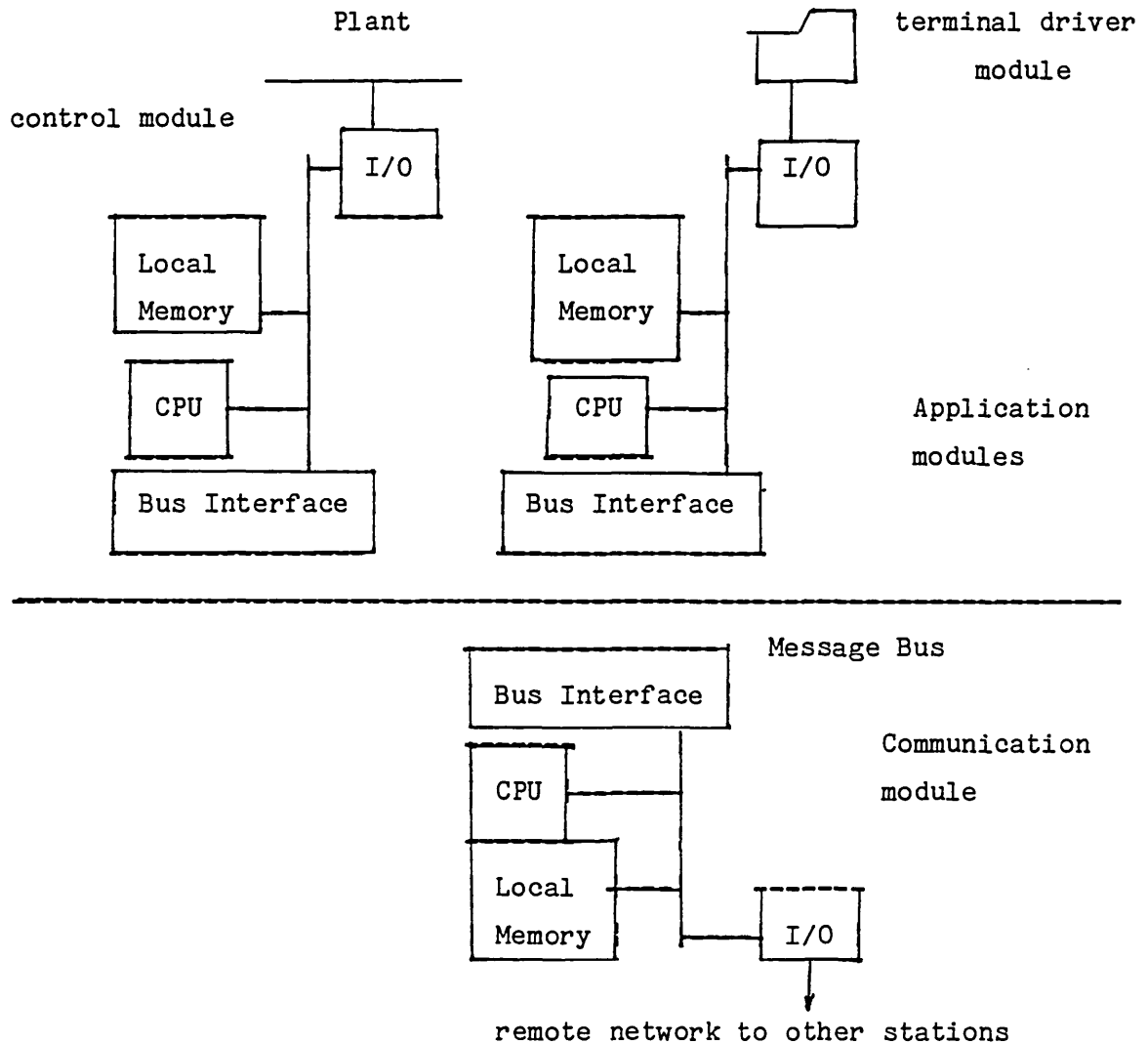


Fig. 2.2 The Message Bus multiComputer station

2.3.2 Functional requirements

The functional requirements of our message bus can be set down as follows -

(a) to allow interconnection of multiple heterogeneous microcomputers to form a multicomputer station. Depending on the requirement of any control function, we would like to connect any single board computer, eg., an NSC800 or MC6800 or Z80 etc., in the station and the

design of the Bus should support this. Implicit here is the requirement that the system software should be designed to be portable over the various processors.

(b) to support efficient and speedy transfer of application messages between modules resident on the processors in the station and also between modules remote to the station. Message Bus communication will form an essential part of the communication between remote modules and hence there should be minimum delay added due to the bus. The communication speeds should thus be in keeping with those laid down in [Prince81].

- (c) to allow the use of standard CONIC operating system features like
1. module creation, deletion and control
 2. linkages between exitports and entryports
 3. station-wide error reporting

In other words, in addition to application messages, the Message Bus would be required to transfer some operating system messages between modules, for example, for down line loading of module code, for link messages to these dynamically created modules and error messages. This effectively means that the average length of messages on the bus will be greater than the normal 8-60 bits needed for application control messages only.

(d) to support redundancy of modules and hence processors in the station, for fault-tolerance and reliability purposes.

(e) to support control of the state of a processor from any other across the bus, for maintenance and debugging purposes, by the use of immediate or direct function interprocessor messages. These will be short messages exchanged between the Bus Interfaces only and will not be sent to the host computers.

(f) to allow the sending of broadcast and multideestination messages between the various computers in the station. These may be either application (control) messages like alarms and sensor readings or system messages of the kind mentioned above. This requirement leads to the need for the design of a transfer protocol which provides a Reliable Broadcast service on the Message Bus.

2.4 Primary features

Due to the proliferation in the design and existence of multicomputer systems today, we faced many choices and hence decisions in arriving at the following features for the Message Bus, based on our requirements. In the following paragraphs we discuss some of these in detail.

(a) Serial Bus Our multicomputer station is a highly localized one, as such a parallel bus would have been quite adequate for interprocessor communication. On the other hand, the station is intended to be part of a real time process control system where size and power consumption have to be kept at a minimum; this precludes the use of numerous (parallel) lines for the bus. The Message Bus is thus a bit-serial synchronous bus with a separate line for the Clock.

(b) Broadcast (shared) medium The Message Bus is a broadcast highway and not a ring or loop. This is in keeping with our requirement of a basic broadcast facility. In addition, a bus can employ passive interfaces in which failures are less likely to halt operation of the station - this is not possible with a ring topology.

(c) Technology Real time process control environments are often subject to extensive electrical noise, hence a balanced twisted pair cable is considered suitable for the Message Bus, in order to provide immunity against noise. Data transmission rates on the twisted pair now go upto 10 Megabits/sec and these are sufficient for our purpose [Prince 81].

Fiber optic cables have many advantages over conventional coaxial or twisted pair cables. These include high data rates, low attenuation, immunity to EMI radiation and increased data security. Hitherto fiber optics did not support broadcast communication. But a recent report [CD82] says that it is now possible to use fiber optical couplers to provide this. These could, in future, be used to implement the Message Bus.

(d) Bus Access and Arbitration Since the Message Bus is a shared medium, we needed a suitable bus access mechanism for it. To achieve maximum reliability in the station in terms of fault isolation, we chose a decentralized access mechanism where each processor (rather

its Bus Interface) is itself responsible for acquiring the bus and there is no dependence on a Central Arbiter to allocate it. It is also a dynamic one ie., an on-demand rather than static one, and is not wasteful of bus band-width.

A major decision was to have a simple contention based mechanism where there is a real possibility that two or more processors would start sending simultaneously and 'collide'. It is now known that for bursty communications at low loads, contention mechanisms are very efficient [Davies 81]. A well known example of such a contention based access mechanism is the CSMA/CD method of Ethernet [Metcal76]. But we felt that such an elaborate and complex method is not in keeping with our Message Bus requirement of simplicity. Instead we chose a priority based bus access method like the one used in the IEEE P896 Serial Bus [Kirman81]. A priority based mechanism also fits in rather well with our concept of the Message Bus as a hardware kernel.

How it works Each Bus Interface BI is assigned a 'priority' in keeping with the nature of the application function executed by its processor's modules. Each potentially sending BI starts transmitting by first putting its priority byte on the Bus and comparing it with what then appears on the Bus. If it finds a difference between what it is sending and what is actually appearing on the Bus, it switches off its output and is said to have 'lost' the arbitration. If no difference is detected, it 'wins' the Bus and continues to transmit.

In the case of more than one BI sending simultaneously, the BI with the highest priority (lowest value) gains control of the Bus and the others sense a 'jamming' of their own signals and retire immediately without sending further. If two BIs have the same priority allocated to them, the arbitration is carried over to their 'Identifier' address which is the next byte to be transmitted. This is the address recognized by the BI as its own and which is hardwired on to the board. The above is achieved by operating the Message Bus as a WIRED-AND bus and implemented by open-collector drivers, so that if a logical 1 is sent at the same time as a logical 0, the state of the bus will be defined by 0. See Figure 2.3 for an explanation of this scheme.

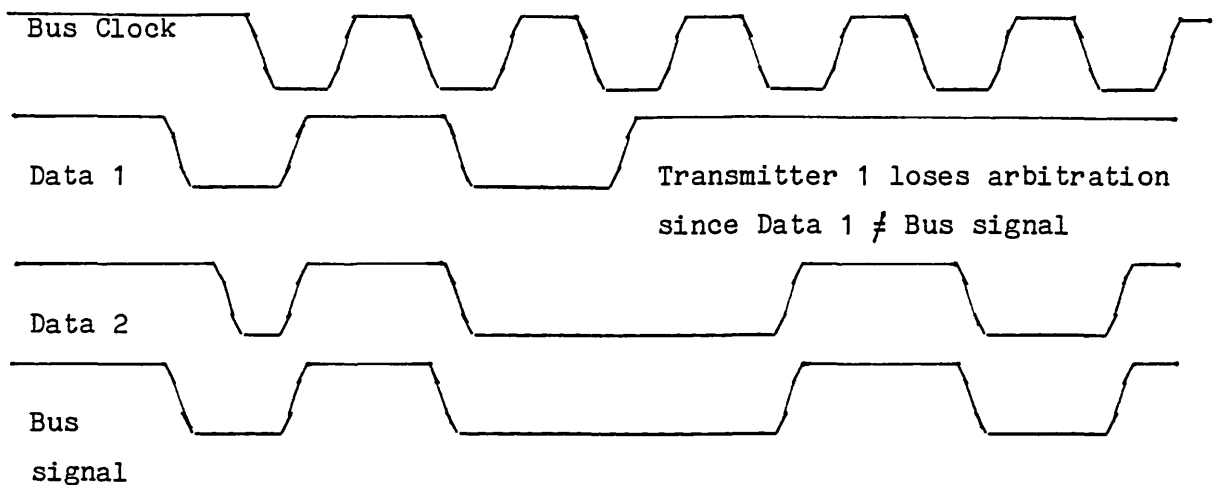


Figure 2.3 A WIRED-AND bus

(e) Direct interprocessor control As mentioned in the requirements for the Message Bus, it is intended to use it for direct control of the state of one processor from any other on the Bus, for maintenance and debugging. This will be achieved by having short system messages sent on the bus from one Bus Interface to another without host processor interference. The BI receiving such **immediate function** messages, would activate one of a set of lines in the interface itself. These would act directly on the host processor in order to halt, initialize, interrupt, connect or disconnect it from the bus. There would be a special 'bit' in the Message Bus format to indicate the presence of such direct control messages. This is similar to the facility provided in the IEEE P896 serial bus.

(f) Addressing for broadcast and multidestination messages Multidestination and broadcast messages are needed in our multicomputer station for two types of messages. These are

i) Direct control messages as mentioned above, which are not sent to any of the software modules on the host processor, but only to its Bus Interface, for directly acting on the processor(s) state. Such messages need only a form of **Physical Addressing**, where a flat address representing a single, broadcast or groupcast address recognizable by each Bus Interface is used.

(Here we differentiate between a broadcast or groupcast address by the fact that the former denotes the common address recognized by all devices on the bus and the latter denotes an address recognized by a

group of devices).

ii) Application (control) messages, eg., alarms and sensor readings, which are sent from a software module on one computer to a preset group of modules resident on any other computer on the bus. This requires a form of logical addressing [Mcquill78] to the module's entryport and hence requires a **hierarchical** address for the port as used in CONIC.

Multidestination addressing has been traditionally achieved on a network by sending multiple copies of the message on each outgoing link. But in order to take advantage of the broadcast nature of the message bus we wanted to send a single, multiaddressed message on it. This address could have been a **variable** length one with a list of addresses of the destinations sent along with the message. On the other hand, it seemed appealing to send a **fixed** length groupcast address on the bus for all messages, irrespective of the number of destinations. Of course it is to be noted that it is possible to form 'groups' of destination modules in a process control system because there is a pre-established pattern of communication in it, which is known at configuration time.

Multidestination message passing in our system is implemented in software by a module called the **distribution module**. This is resident in each computer in the station and has the same address in all of them. The details of this module and its working are not relevant here and have not been discussed. They can be found in [Garg83a].

(g) Message reception and signalling Each receiving BI recognizes its address in a message by comparing it with that hardwired onto its board. It also recognizes a broadcast address as discussed above.

We studied a number of existing message reception techniques for networks and multicomputers [Brinkm80, Bass80, Crane81, Estrin81]. It was found that there are two common approaches -

- (i) Provision for DMA by the Bus Interface to and from the host processor's memory, and
- (ii) Existence of a dual port memory accessible to both the Bus Interface Hardware and the host processor software.

Both the methods are equally effective and we chose the latter one. Incoming messages are received by the BI in one of an input queue of

message buffers which form part of the dual port memory. The arrival of the message is signalled to the software by an interrupt. Similarly, outgoing messages are put by the software in an output queue of buffers from this dual port memory and the BI is informed that there is a message to be sent out of the processor.

(h) Message Bus Format The message format for messages on the bus is shown below in Figure 2.4.

PRIORITY	
SOURCE ADDRESS	(processor.module.port)
DESTINATION ADDRESS	(processor.module.port)
TRANSACTION ID	
DATA LENGTH	
APPLICATION DATA	

Figure 2.4 Application message format

2.5 ERROR PROTOCOL FOR MESSAGE BUS

2.5.1 Reliable Datagrams

Currently, the CONIC communication system provides for both simple datagram service as well as virtual circuit service on the DCCS network. For the message bus it was decided to have a Reliable Datagram service in which delivery of a correct message is guaranteed, or the user is informed of failure and duplicates messages are discarded [Belang81].

A Reliable Datagram service was felt to be both adequate as well as better suited for the Message Bus communication for the following reasons -

- * The message bus is a simple shared path and there are no store-and-forward gateways which are a common source of message corruption.
- * Broadcast and multidestination messages on the bus are difficult to implement with a virtual circuit service.
- * The provision of reliable datagrams on the message bus ensures that the reliability achieved by virtual circuits on the remote network is not unduly compromised by communication on the multicomputer station bus.

On a broadcast bus, with no intervening network layer, the above service can be provided either at the transport layer level ie., in software or at the lower data link layer level, ie., in hardware at the Bus Interface itself. Providing error control at the Bus Interface level has the following advantages -

- * Since 'corrupt' messages are filtered out at the Bus Interface itself, the time-critical software processing is not needlessly interrupted by such messages, as it would be, if error control was implemented by the transport layer software.

- * In order to send a message successfully, the bus needs to be requested (and hence arbitrated) only once. This is not possible if the software has to retransmit a message or send an acknowledgement(ACK). In such a case, the bus has to be requested again and again by the software, till the message has been successfully transmitted. This increases considerably the communication overheads in the station. Providing error control at the BI means that the BI can get hold of the bus at the beginning, send a message, allow for an immediate ACK or NAK (Negative Acknowledgement), retransmit if necessary, and release the bus only after it is assured that the message is successfully delivered.

- * One of our main requirements is to keep the error protocol as simple and efficient as possible, but at the same time insure that there is minimum delay on the bus for messages which are to ultimately go out from the station. Speed is thus an important aspect of the protocol chosen for the message bus.

2.5.2 Errors on the broadcast bus

Before considering the suitability of any particular error control protocol, it was necessary to identify the different types of errors that are likely to occur during message transmission on a broadcast bus and the various possible causes for each. These are obviously less in number than would normally occur in a multi-path network and are listed below -

- * corruption of or damage to messages - a message may be corrupted or damaged due to two reasons
 - a) noise on the bus
 - b) fault in the memory of processor

- * loss of message - the message may be lost ie., not reach the destination because
 - a) there is damage to its header and so the destination address is not recognized
 - b) there are no receiver buffers available
- * duplication of message - a message may be sent more than once because the acknowledgement from the receiver is lost
- * out-of-sequence messages - a message may arrive at the receiver out of order because the error protocol allows multiple outstanding unacknowledged messages to be present and one or more is lost.

In order to be able to use a simple protocol, we can reduce further the number of possible errors by

- a) providing error-controlled (Hamming coded) memory chips only in the station. This eliminates errors due to fault in memory.
- b) opting for a STOP and GO protocol ie., having only one unacknowledged message on the bus at any time. This eliminates sequence messages from the bus.

2.6 The INPAR (Immediate Negative Positive Acknowledgement and Retransmissions) protocol

In view of the above reasons, there was a need for an error protocol that provided protection against

- a) corrupt messages
- b) lost messages
- c) unavailability of receiver buffers and
- d) duplicate messages

with the added facility of reporting failure of message transfer, if any, to the user. In the following paragraphs we discuss the important features of the INPAR protocol designed for the Message Bus.

Corrupted or damaged message packets (ie., messages which contain a header and a trailer) are detected by having the usual CRC (Cyclic Redundancy Checks) done on the contents of the packet. If a received packet is found corrupted, a NAK is sent back immediately by the receiving BI.

Similarly, if a packet has been received undamaged, but there are no receiver buffers available, a NAK is sent back immediately.

If the packet has been received undamaged, is not a duplicate and there is a buffer available to receive it, then an ACK is sent immediately. On receiving an ACK, the transaction is complete and the sender BI releases the bus. The Bus Release signal is received by all other BIs on the bus, and they can now request the bus if they want to send.

If the sending BI receives an immediate NAK, or there is neither an immediate ACK nor a NAK (the packet was lost) then the packet is retransmitted. The number of retransmissions allowed, say n , is prefixed and if the packet cannot be sent successfully after n retries, the transaction is considered complete, though unsuccessful, and the Bus Release is sent to indicate that the bus is free. Failure is reported to the user.

The use of NAKs in the protocol helps support diagnostics in the system, since it is possible to keep a record of the number of unsuccessful sends and their causes.

Duplicate packets are detected by attaching a Packet Id (SPI) in each packet sent by a BI. This is the value in an SPI counter maintained at each BI and which is incremented after every Bus Release by the sender. Each BI has a different SPI number and duplicates can only be generated by retransmissions before the sender releases the bus. The BIs also maintain another counter called the Receive Packet Id counter at the receiving side (RPI) which is cleared on reception of every Bus Release. The following actions are taken -

- If on reception of a packet, RPI is found cleared, the packet is considered new. It is received, acknowledged and sent up to the software. The RPI is then set to the value of SPI received in the packet.
- If on reception of a packet, RPI is found equal to the SPI received, the packet is considered a duplicate. It is acknowledged but not sent to the user. (It is assumed that its previous ACK was lost and so it was retransmitted.)
- If on reception of a packet, RPI is found to contain a non-zero value which is not equal to the SPI received, the packet is again considered new, is received, acknowledged and sent up. (In this case it is assumed that the Bus Release signal was lost and a new packet has been sent).

Figure 2.6 shows the Bus Release signal and the 2-bit immediate ACK and NAK signals which give the INPAR protocol its name. The Bus Release signal is a unique condition on the bus and is provided by two STOP signals spaced four clock periods apart. The START and STOP signals shown in the figure have been defined exactly as in [Goldbe82]. Figure 2.7 shows the format of the complete error control packet on the bus, including the fields shown previously for the application message. Figure 2.8 shows the block diagram of the Bus Interface hardware required to implement the INPAR protocol fully.

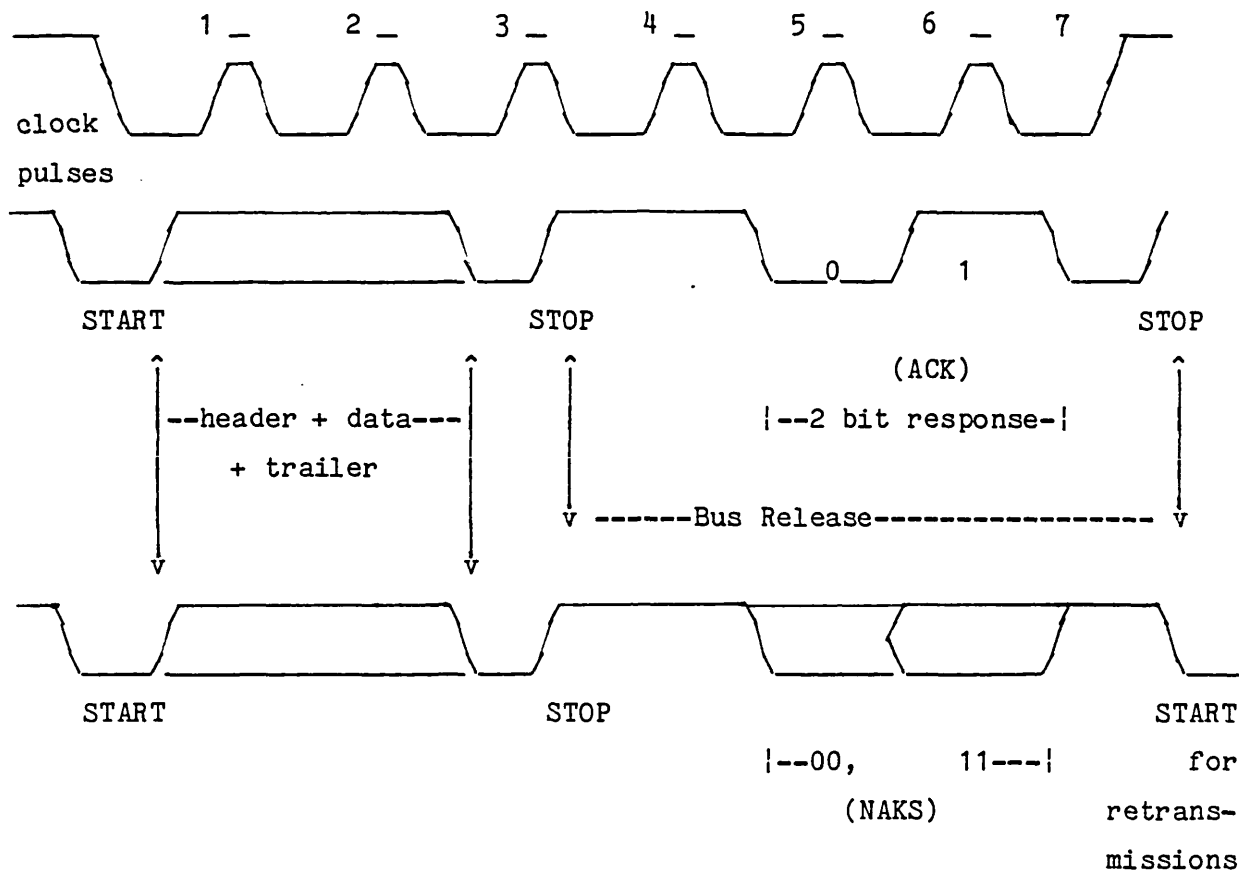


Figure 2.6 Signals on the Message Bus

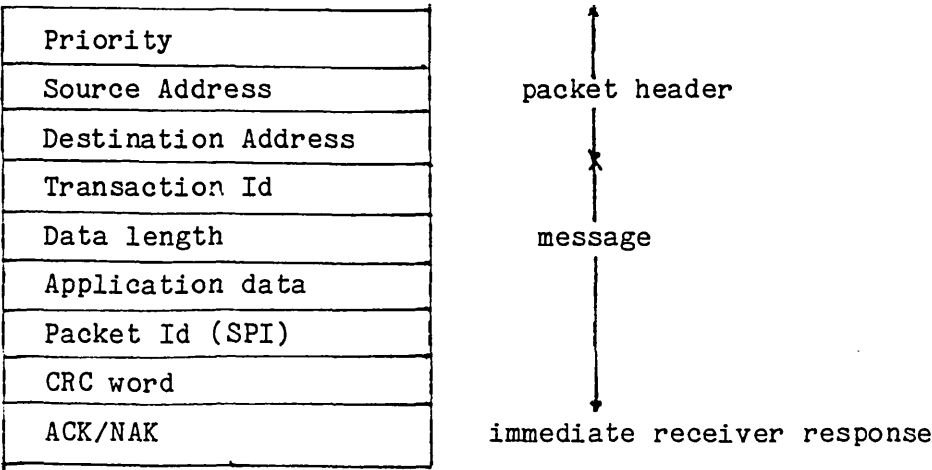


Figure 2.7 The INPAR protocol message format

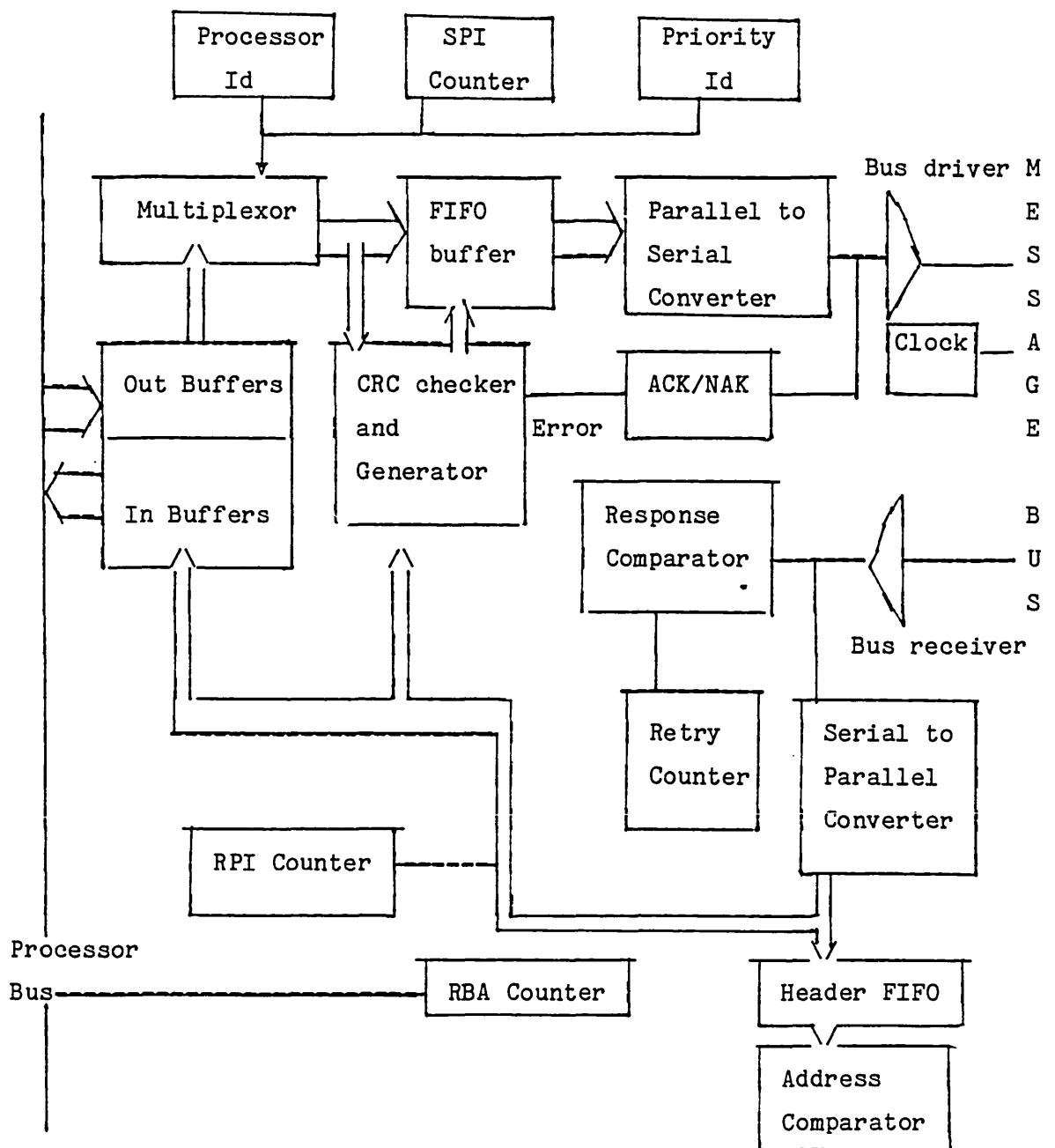


Figure 2.8 Block diagram of the Bus Interface hardware

2.6.1 Reliable Broadcasts

The requirement for multdestination messages on the bus raises the question of how to provide immediate responses (ACK/NAKs) for error control for such messages. It was considered important that even for such messages, there should be the same confidence in their correct transfer as there is for single destination messages, because the frequency of such broadcast and multdestination messages is quite considerable.

There are problems in providing these responses. Should the sender wait for an immediate reply from one receiver only or from all of them? It is obvious that all receivers cannot send such immediate replies if they cannot all acquire the bus and 'send' simultaneously. Which among these receivers should then reply? We have found that this problem has been avoided by most designers of communication systems simply by not providing any form of control for multideestination messages. The Match and Accept bits in the DCS [Mockap76] provide such immediate response for multideestination messages, but that was possible due to the store-and-forward nature of their communication network. On a broadcast bus this cannot be done.

For our Message Bus we have been able to solve this problem quite effectively by using the INPAR protocol and the fact that the bus is a WIRED-AND one. It has become possible to provide a **Reliable Broadcast** service on the Bus. This is explained below.

When a broadcast message is received on the bus, all receivers will try to send responses back. The response has been represented by the bits 01 (ACK) if the message is received correctly and 00 (NAK1) if an error was detected or no buffer was available at any receiver. Due to the WIRED-AND implementation of the Bus, the NAK will override any ACK and be received at the sender as 00. If all receivers are dead, the response detected will be 11 (NAK2). On receiving a NAK, the sender will retransmit the message upto a maximum of N times. For simplicity, the sender does not distinguish between NAK1 and NAK2.

This means that either the message will be received **correctly** by all working receivers or it will be retransmitted if **any one of them** had sent a NAK. The normal duplicate detection mechanism will result in the retransmissions being discarded by the Bus Interfaces that had already received the message. This is what we want for **Reliable Broadcast semantics**.

2.7 GENERAL DISCUSSION OF THE MESSAGE BUS

2.7.1 The Message Bus vs. Local Area Networks (LANs)

If we look at the features provided in the Message Bus, we find a number of similarities between it and a LAN. For example, bus access and arbitration mechanisms and error control protocols are required for both. How, then, is our Message Bus different from a LAN? The differences are not so much technical or topological. Rather the distinctions are more philosophical and conceptual as discussed below.

(i) The Message Bus has been conceived as a 'hardware kernel' which enables hardware 'modules' to coexist in a station and communicate with each other. It is also intended to separate CONIC modules in a DCCS station in order to provide protection and independence, while still maintaining their close interaction. A LAN, on the other hand, is a data communication system which allows a number of independent devices to (merely) communicate with each other [IEEE 81].

(ii) The communication system on the message bus is much simpler than that on most LANs because of the absence of the network layer. Also it is easier to provide broadcast and multidestination messages on the bus.

(iii) Geographically, the message bus gives rise to a much more highly localized multiprocessor system than does a LAN. The length of the message bus is typically between 1/2 and 1m whereas LANs are normally more than 1 km. long.

(iv) The message bus is intended to be much simpler and less expensive than a LAN.

(vi) Finally, one of the main advantages of the message bus concept is that it allows more software functions to be implemented in hardware and thus enables the use of more low cost hardware to alleviate the high cost of software. This is generally not the case in existing LANs.

2.7.2 The Message Bus and Ethernet

It is clear from the above discussion that the Message bus is

different from a LAN. But it is also true that it needs a number of features which are similar to LANs. For example, an efficient bus access mechanism is needed to arbitrate the bus between the different processors connected to it. The Ethernet CSMA/CD mechanism is considered to be very effective for a shared broadcast medium. It was considered as a possible candidate for the message bus, but was rejected on grounds of complexity and speed. The simpler, hardware implemented and priority based arbitration method used in the IEEE P896 Serial Bus was chosen instead as it was adequate for the message bus requirements.

The implementation of the message bus as a Wired-And was initially thought to restrict data transfer rates on it. The maximum rate of data transfer on such a bus is inversely proportional to its length, its line capacitance and the capacitance of each hardware module on it, and directly proportional to the driver sink current, as shown in the graphs of Figure 2.9. But it was found that it is theoretically possible to go upto a data rate of about 11 Mbits/sec with the current state of art in bus Drivers, with bus lengths of 1/2m and 32 modules on the bus. Increasing the bus length to 1m reduces the maximum achievable data rate by half. These data transfer rates are quite sufficient for our purposes anyway. The Ethernet can also transfer data upto 10Mbits/sec, but because of the higher level retransmissions used in its error protocol, the actual information transfer rate [Prince 81] is much less. With the Message Bus protocol, this is not the case as it is implemented in hardware.

2.7.3 Performance characteristics of the Message Bus

In [Patzel82] the author has laid down certain performance requirements for multiprocessor buses. Of these, those relevant in the Message Bus context are Data (throughput) rate, Access rate and Bus operation rate.

Data (or information) rate refers to the transfer of the actual application information via an established connection. It is expressed as the reciprocal of information transfer time.

Access rate or Action rate is defined by the time to gain access to the bus (arbitration time), select another device as receiver (addressing time) and to perform the message transfer itself.

Numerically it is the reciprocal of the sum of the above three times.

Bus Operation Rate is the reciprocal of the time for the full length of the bus operation cycle, including the time to transfer response signals to the sender from the receiver.

The throughput or information transfer rate on the Message Bus is shown in the graphs of Figure 2.10. These have been calculated for a maximum packet length of 144 bytes in which 12 bytes are used for protocol header and trailer. The theoretical throughput rate is shown against increasing packet lengths for various transmission rates on the Bus. It can be seen that even at the lowest allowed transmission rate of 2 Mbits/sec., the throughput rate is much higher than the maximum laid out in [Prince81] for process control systems, for typical control messages of 8 byte length. (The reductions due to retransmission overheads have not been considered here.)

The arbitration time for a message transfer on the bus is only one byte time because of the method used. This time and the data transfer and addressing times add up simply to the time required for the complete packet to be transferred.

For a maximum length packet on the bus, the Access Rate is thus

$$1 / \text{packet transfer time}$$

which is 1.73k/sec., 4.32k/sec. and 8.65k/sec. for transmission rates of 2, 5 and 10 Mbits/sec. respectively. This means that the bus can be accessed, theoretically, for at least 1730 messages every second.

It has been calculated [Garg81] that maximum length application messages can be set up by a processor (of the speed and power of an LSI 11/23) every 6 msecs., i.e. at the rate of 166 messages/sec. For 16 hardware processor modules connected on the bus, the full load offered to the Bus is about 2750 messages every second. This figure is compatible with the Access Rates calculated above for the Message Bus.

The Bus Operation Rate for the Message Bus is the same as Access Rate since the 2 bit response from the receiver has been included in the packet transfer time.

2.8 COMPARISON WITH OTHER MULTIPROCESSOR BUSES AND BUS STANDARDS

The great proliferation in multiple microprocessor systems in recent years has necessitated the use of efficient interconnecting systems to exploit their potential of higher speeds and performance. Shared buses are a natural interconnecting mechanism for localized multiple computing systems and as such have become very popular for use in multiprocessors, though crossbar switches and delta networks are also being used. Attempts are also being made to standardize multiprocessor buses in order to make available support and interface boards that can be widely used. While all multiprocessor buses and standards available to date are very much a 'variation on the same theme [Borill82], providing a mechanical format, a set of protocols and a signal set which attempts to minimize the required amount of circuitry', each has some unique features in order to best meet the functional requirements laid down for its application.

The following is a brief review of some of the better known buses and bus standards in existence today. It serves to bring out the essential features of each one of the buses in order to show how it compares with and differs from the Message Bus and was thus inadequate for use in the multiprocessor proposed by us. The following features of buses were used for comparison:

- * serial vs. parallel structure
- * tight vs. loose coupling
- * heterogeneity or manufacturer dependency
- * decentralized or centralized arbitration and control
- * cost and complexity
- * speed of data transfer
- * broadcast and multidestination transfer facility | these are
- * error control protocol | specific to
- * maximum message length | serial buses.

IEEE GPIB 488 Bus

This General Purpose Interface Bus was designed by Hewlett Packard primarily for interconnection of its test equipment and is considered exceptionally good and cost-effective for this purpose. It is an 8-bit parallel data bus with a maximum data transfer rate of 1 Mbytes/sec., and capable of carrying 15 modules on it. Bus arbitration is centralized and serial (daisy-chain). It has a single master

(controller) on the bus which means it has a single failure point and is really rather limited in functionality and speed as compared to the needs of today's multiprocessor environment. It also suffers from the disadvantage of giving rise to a manufacturer dependent system hardware.

INTEL MULTIBUS

The Intel MULTIBUS (now the IEEE P796 Standard) is a multimaster bus designed for interconnection of processors and devices manufactured by Intel. It is a 16-bit parallel data bus with data transfers occurring with a maximum bandwidth of 5 MHz but typically reaching speeds of 2 MHz due to bus arbitration and memory access delays. It has a centralized form of parallel or serial bus arbitration mechanism and control. It is capable of supporting between 3 to 8 bus masters which may be 8- and 16- bit processors, peripheral controllers, etc. Slave devices include memory and digital and analog I/O expansion boards. Communication is by shared memory on the Bus. While it has the advantage that it is backed by Intel's complete range of Single Board Computers and other off-the-shelf support devices, this very feature makes it manufacturer dependent and unsuitable for use in a heterogeneous multiprocessor environment. Moreover, being a parallel bus, it has been considered rather unwieldy and unsuitable for use in our context.

VERSAbus and VMEbus

The VERSAbus and its Eurocard counterpart the VMEbus, were built by Motorola MicroSystems to cater to the present day needs of multiprocessor systems. Both are essentially a collection of five buses, one each for data transfer, priority interrupt, arbitration, utility and interintelligence bus. The Data Transfer Bus (DTB) is a (upto) 32-bit parallel bus with data rates capable of reaching 5 MHz. The priority interrupt Bus allows interrupt generation from multiple devices. Multiple masters can gain orderly access to slave resources via the DTB arbitration bus. The Bus arbitration mechanism is priority based, centralized and uses serial daisy chaining. The utility bus handles broadcasted system status and control signals, eg., system or power failure, reset/reinitialization, etc. The interintelligence bus was created in order to relieve traffic on the data transfer bus as the number of processors increases. It is a two-wire, 4 MHz synchronous

serial bus used to pass short messages between multiple intelligent boards. The VERSAbus is definitely an improvement on the Intel MULTIBUS in terms of functionality, but has some similar undesirable features and complexity. Its interintelligence bus, with specific improvements, is the type of bus we would like to use in our multiprocessor.

The EUROBUS standard

The EUROBUS standard was developed by the United Kingdom Ministry of Defence, in an attempt to standardize multiprocessor buses and make them manufacturer and processor independent. It has been adopted by the British Standards Institute and also submitted to the International Standards Organization. It is primarily intended for avionic and land-based defense systems but can be used in the real time process control environment also. It is a 16-bit parallel data bus, extendable to 24- and 32- bits and can support upto 20 modules. Various support devices are also available for it, eg., for bus arbitration, bus interface and vector interrupt. It has a centralized bus arbitration mechanism and facility for DMA transfer. But no block transfers or broadcast transfers are possible. Although the EUROBUS encourages a highly modular, card-oriented approach to real time multiprocessor system design, it is unsuitable for us for the same reasons as the other parallel buses considered above.

The Honeywell Experimental Distributed Processor (HXDP) bus

The HXDP is a research-oriented distributed multiple processor system designed at Honeywell specifically to study the science and engineering of processor interconnection and decentralized executive control. It has a bit-serial bus for interconnecting in a loosely coupled manner, upto 64 heterogeneous processing elements (processor/memory pairs). Data transfer speeds of 1.25 Mbits/sec. are possible on it with the type of technology used then. Its main feature is the elaborate decentralized bus arbitration mechanism, called VDPA - for Vector Driven Proportional Access - which was designed at Honeywell. Another important feature of the HXDP processor is that the error control protocol is designed to be implemented totally in hardware, and has the ability to detect and recover from over 40 types of errors, allowing very reliable message transfer. The HXDP bus does not support broadcast or multicast addressing, neither does it have

provision for direct control of one processor from another.

The hardware/software tradeoffs made for the HXDP system are very interesting in our context as we have made many similar decisions for our Multiprocessor. The HXDP bus, besides being a serial one has many desirable features for our purpose.

IEEE P896 (or Future) Bus

This is a comparatively new bus, developed by the EWICS (European Workshop on Industrial Computer Systems) and recommended to the IEEE. It is a very ambitious bus, designed with the future in mind. It is capable of supporting from 8- to 32- bit processors in a symmetrical, tightly coupled multiprocessor environment. It is actually a dual bus, having both a parallel and serial bus (like the VERSAbus discussed above). The parallel bus is the actual data transfer path for up to 32- bit words and supports shared memory. The serial bus is to be used mainly for carrying Operating System messages and interrupts between processors, with a limit of 48 bit length. Speeds on the bus go up to 10 Mbits/sec. Both buses are arbitrated in a completely decentralized manner and the mechanism is very simple (COBUS- like).

A very interesting feature of the serial bus is that it provides for **immediate function** messages from one processor to another. This is very useful for interprocessor communication, whereby one processor can directly halt, interrupt, connect/disconnect, restart another. Both buses support broadcast and multidestination message passing. Limited error control is also performed on the serial bus in the form of immediate Acknowledgements (Acks, but not NAKs) to messages from receivers.

We have found the serial bus of the IEEE P896 very interesting, though of somewhat limited functionality in terms of maximum message lengths allowed and hardware error protocol used. But its other features, like the immediate function capability, are very useful and have been adopted for use in our multiprocessor.

Phillips I2C and D2B buses

These are two very similar buses developed by Phillips, for individual

consumer electronic systems in the industry and home, and modelled as small area networks or SAN. The 10m long I2C bus interconnects integrated circuits in small modular systems such as microcomputer controlled entertainment systems, whereas the 150m D2B bus, with more powerful protocol and high noise immunity, connects upto 50 subsystems like industrial apparatus and instruments. They are both 2-wire, serial data buses operating at data rates from 100 to 10,000 bytes/sec. A very attractive feature in both is the priority based distributed arbitration mechanism implemented by a unique Wired-AND design, thus operating without any overhead whatsoever. Message lengths range from 2 to 75 byte only and the message format resembles asynchronous data link frames. Limited error protocol is performed by the bus interface hardware.

Though speeds on the above buses are very restrictive and so are message lengths, we found their arbitration mechanism particularly simple and effective and hence suitable for real time environments like process control.

2.9 CONCLUSION

A simple but powerful multicomputer Bus has been proposed for use in real-time process control systems. It has been shown how, by implementing the Message Bus as a WIRED-AND and by the use of the INPAR protocol designed for error free transfer across it, it has been made possible to achieve Reliable Broadcasts on it.

A major design decision has been to implement the Message Bus protocol in hardware. Implicit in this decision has been the assumption that VLSI and LSI technology will continue to play an important part and be increasingly used in future to implement functions characteristic of a Bus protocol, irrespective of their complexity and at a greatly reduced overall cost [Ander 81]. The design of some aspects of the message bus have been greatly influenced by the HXDP bus [Jensen 78] and the IEEE serial P896 bus [Kirman81].

It may be noted that the design presented above has concerned only qualitative aspects in detail. Quantitative design factors, like required number of input/output buffers in the Bus Interface, number of allowed retransmissions in the protocol, and the actual criterion for priority allocation to processors, could not be considered. This

is also reflected in the above calculated performance figures for the Bus, which are only very preliminary. They do not take into account these various factors and their effects on performance. This is because it is impossible to consider them without some formal means of representing or modelling them and analysing their effect. In the following chapters we shall see how analysis of the Timed Petrinet models of the various subsystems of the multicomputer help to make this possible and fill the 'gaps' in the above design.

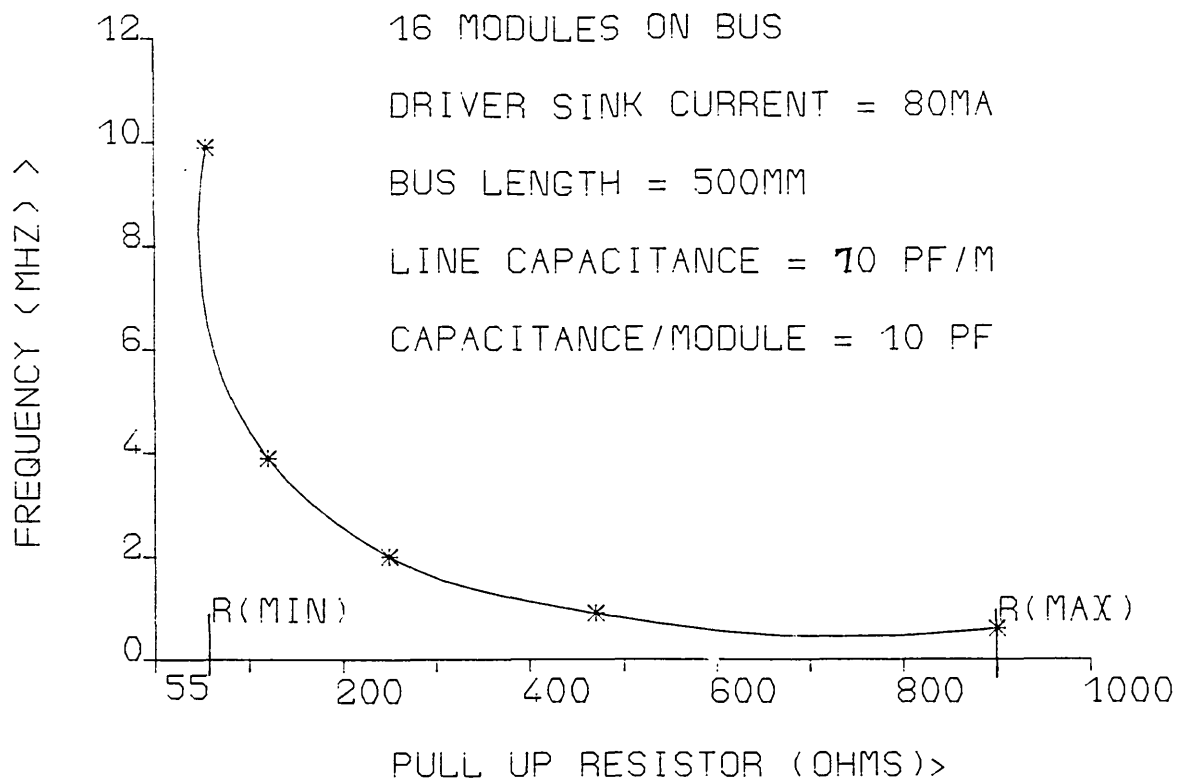


Fig. 2.9a Frequency vs. Pull Up Resistor

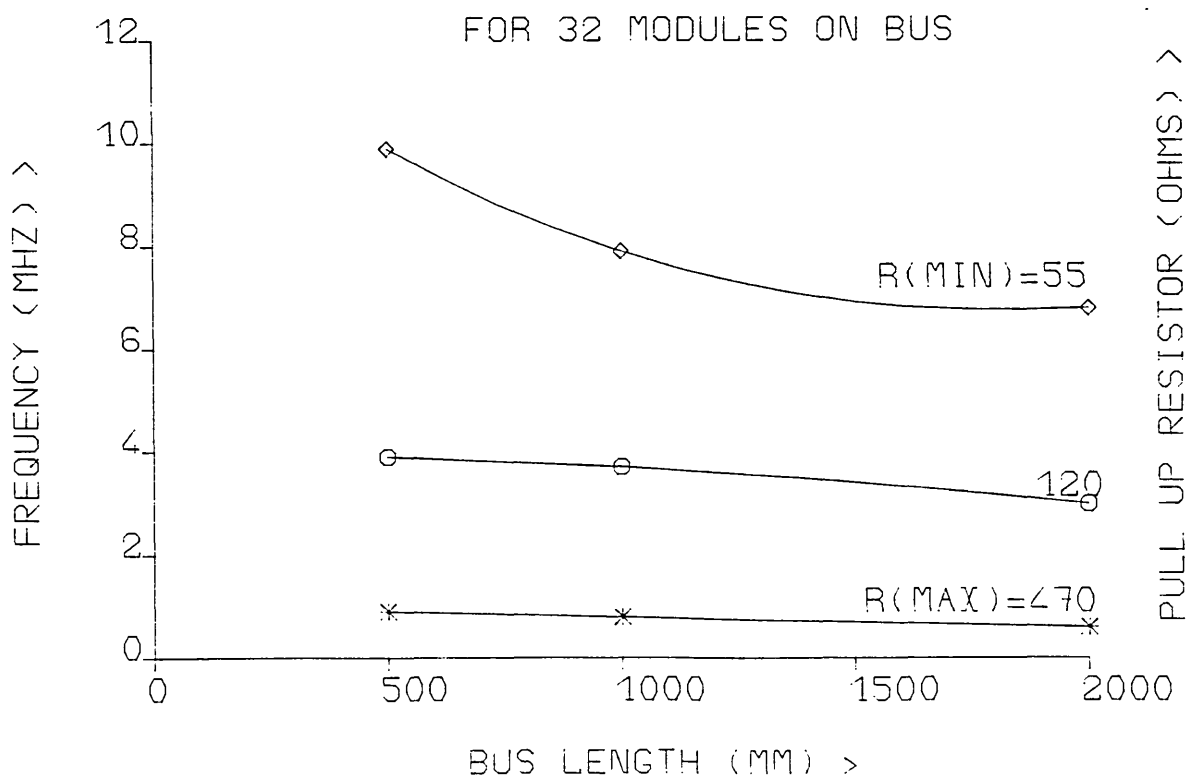


Fig. 2.9b Frequency vs. Bus Length

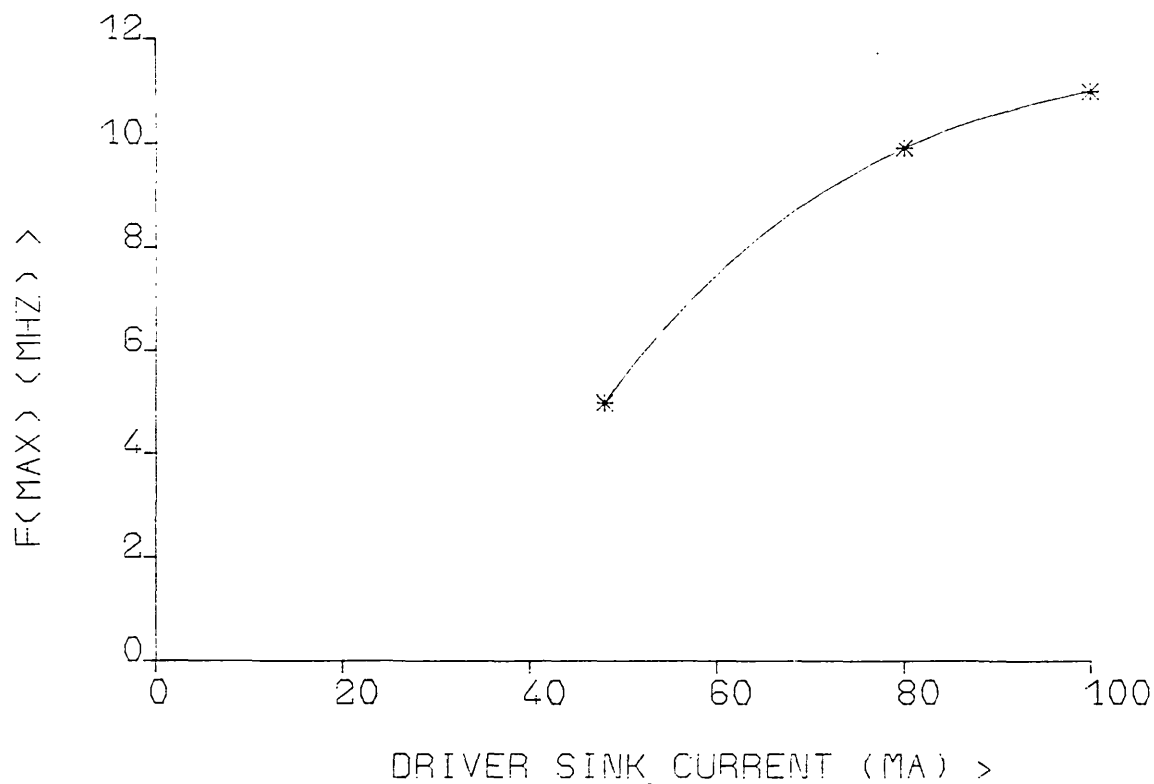


Fig. 2.9c Maximum Frequency vs. Driver Sink Current

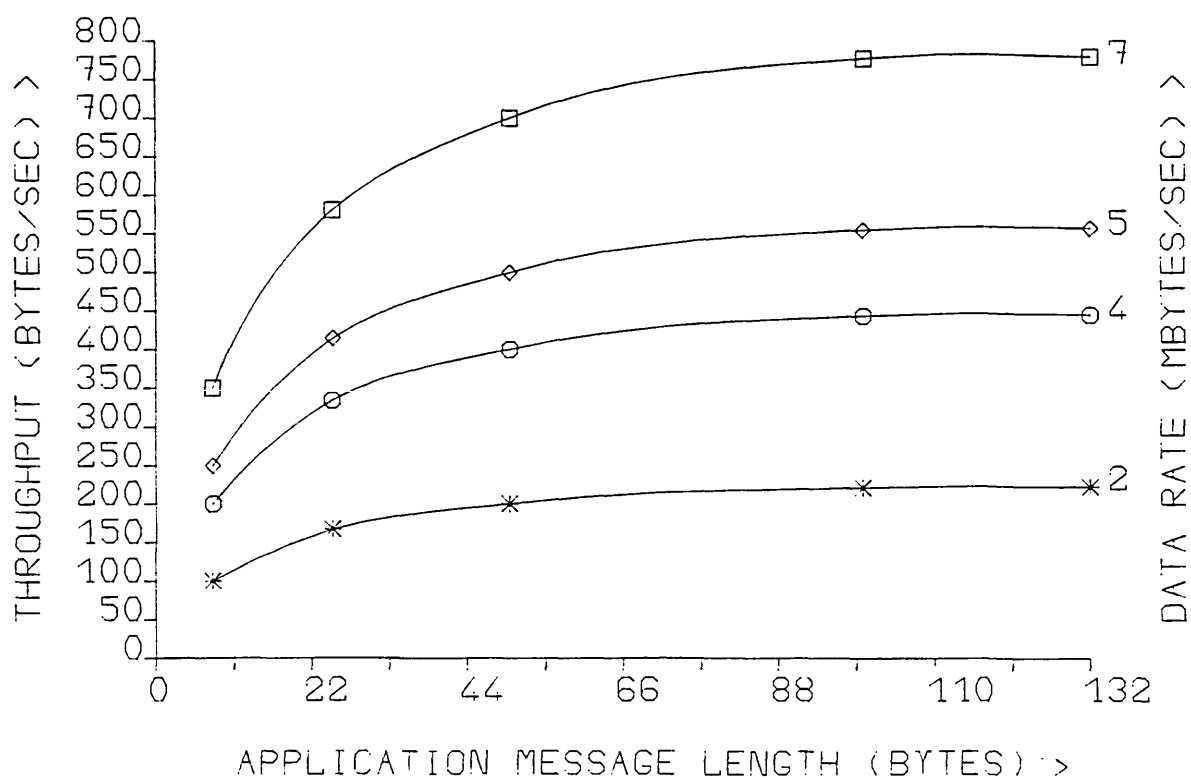


Fig. 2.10 Throughput Rate on Message Bus

CHAPTER 3

BUS CONTENTION SUBSYSTEM ANALYSIS

3.1 INTRODUCTION

This chapter is concerned with developing and analyzing a Timed petrinet model of contention on a single, shared bus by the various processors in a multiple processor system. The system chosen for illustration is the Message bus based multiprocessor designed in Chapter 2. Contention for a shared bus is the first area we have identified for a multiprocessor which depicts conflict, concurrency and synchronization. Memory interference is another such area but we have not considered it since our illustrative multiprocessor has no shared memory. However, we show later in the chapter that our model lends itself easily to various extensions to incorporate this and other general characteristics of multiple processor systems.

Based on its Timed Petrinet analysis, we are not only able to do a performance analysis of the bus contention system in terms of the message throughput on the Bus, given each processor's bus accessing and processing times, but also give a unique criterion by which it is possible to allocate priorities to the processors in order that they may periodically access the bus with minimum delay. This in turn shows that a priority based bus arbitration policy is very useful in multiprocessors, besides being simple to implement. In other words, we are able to **validate** the design proposed for the Message Bus given in the previous chapter.

We start the chapter by discussing the origin and theory of Timed petrinets and also why we chose Sifakis's TPNs for performance analysis of our systems. This is the subject of Section 2. In section 3 we develop and analyze the bus contention model. In Section 4 we show how to maximize throughput on a bus using this analysis. We also equate our maximization problem with the well known Knapsack problem of Dynamic Programming and give a simple computer program to solve it for the case of 16 processors on a bus. In Section 5 we discuss the implication of the analysis on various design factors and consider extensions possible in the model. A discussion of related work is given in Section 6 and conclusions are presented in Section 7.

3.2 TIMED PETRINETTS

3.2.1 Origin of Timed petrinets

As mentioned in Chapter 1, Timed Petrinets were first introduced by Ramchandani [Ramcha73] by associating a firing time to each transition of an ordinary Petrinet in order to be able to study its steady state behaviour. In an independent study, Merlin [Merlin74] investigated the recoverability problems in communication protocols by studying petrinets in which two different time values are associated with each transition. He called these Time Petrinets.

The essential difference between these two is in the process of firing of transitions. In Timed petrinets, the fixed value t is the time taken for the transition to fire, but the firing starts as soon as the transition is enabled. On the other hand, in Time petrinets, the actual firing is instantaneous but this must not happen before time a or after time b from the instant of enabling. Another difference is that whereas the latter have been used mainly for functional design of communication protocols, the former have been used for performance evaluation purposes.

Today, a vast body of research is being conducted into the theory and application of such petrinets. Some researchers have applied the concept of time in Timed petrinets by simple mathematical manipulation on it to their specific systems. But mathematicians, particularly in France, have concentrated on developing a formal theory for Timed Petrinets analysis, in order that it can be applied in a systematic and automatic way to computer systems. Indeed, the Timed Petrinets theory has developed to an extent where it can be usefully applied in the area of distributed systems for performance evaluation and protocol validation. Figure 3.1 shows the various directions taken in the research on Timed Petrinets.

Thus a substantially developed theory for both types of Time(d) Petrinets is now available. Conditions for boundedness and liveness of these nets have been compared with those of ordinary Petrinets [Ghosh77]. Some researchers have also reported different interpretations and extensions, eg., [Sifaki77, Molloy81].

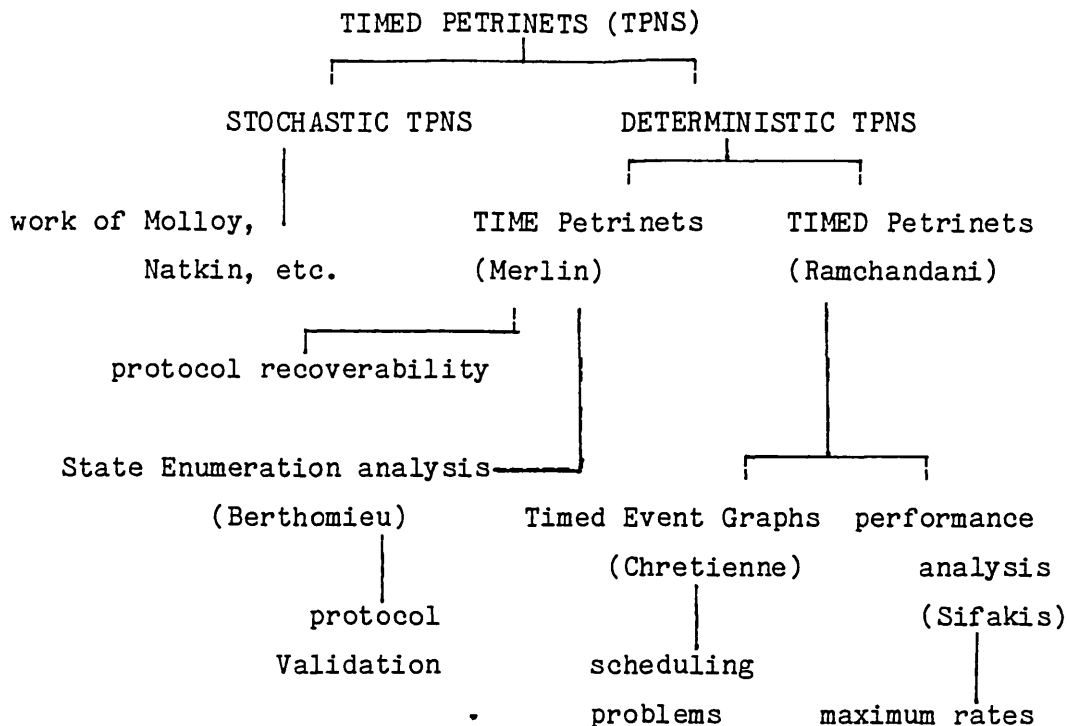


Figure 3.1 Development of Timed Petrinets theory

3.2.2 Choice of Timed Petrinets

Sifakis has associated fixed time values to the places of Timed Petrinets and proved [Sifaki80] that they are mathematically equivalent to those of Ramchandani, ie., those with fixed time values associated to transitions. But we feel that his TPNs are the most suited to model distributed multiple processor systems as real life system 'events' are normally instantaneous but 'activities' are associated with time durations and in Petrinets events are modelled by transitions and activities by places.

In this respect, Merlin's Time petrinets are also suitable for modelling our systems, but we have not chosen them because, firstly, no performance analysis has as yet been done with them and secondly, the presence of two time values, associated with a transition firing, would complicate such analysis.

Molloy and others have studied stochastic Timed Petrinets where random firing times are associated with the firing of transitions. We consider these as suitable for modelling Office Automation systems, Local Area networks and other general purpose multiprocessor systems used by human operators. Here events are controlled by the needs and choices of the operators at a given instant and such systems can be

thought of as nondeterministic systems.

Distributed Computer Control Systems are more deterministic since known fixed times are taken by the processors to perform any computation in the system. For example, when a microprocessor is used to monitor a sensor attached to the 'plant' being controlled, it follows a fixed cycle of monitoring, processing and reporting (sending) this information. There are no probabilities associated with these activities. (Of course, it may be argued that there is a certain nondeterminism in DCCS, as represented by exceptional conditions in the plant and subsequent alarm situations arising. But as we shall see in this and subsequent chapters, these can be easily accommodated even in deterministic models.) Thus we have chosen Sifakis's timed petrinets for our system.

The various Timed Petrinets mentioned above will be discussed in detail later in the thesis to compare their applications with those of Sifakis's models.

3.2.3 Timed Petrinet Theory

Sifakis has defined and represented Timed Petrinets as follows -

Definition 1 A Petrinet (PN) is a quadruple $N = (P, T, \alpha, \beta)$ where

P : is a set of places, $P \neq \emptyset$

T : is a set of transitions, $T \neq \emptyset$, $P \cap T = \emptyset$

$\alpha : P \times T \rightarrow \mathbb{N}$ the forward incidence function

$\beta : P \times T \rightarrow \mathbb{N}$ the backward incidence function

Representation To a PN one can associate a digraph, the nodes of which are the places and transitions, represented respectively by circles and dashes. There is a directed edge from the place p_s to the transition t_j iff,

$\alpha(p_s, t_j) = n_{s,j} \neq 0$, where $n_{s,j}$ is called the weight of the edge.

There is also a directed edge from transition t_r to place p_w iff,

$\beta(p_w, t_r) = n_{w,r} \neq 0$, where $n_{w,r}$ is called its weight.

A marking M of a PN, shown by dots (called markers or tokens) in the place p , is a mapping of P into $\mathbb{N}$,

$$M \\ \text{i.e., } P \rightarrow \mathbb{N}$$

thought of as nondeterministic systems.

Distributed Computer Control Systems are more deterministic since known fixed times are taken by the processors to perform any computation in the system. For example, when a microprocessor is used to monitor a sensor attached to the 'plant' being controlled, it follows a fixed cycle of monitoring, processing and reporting (sending) this information. There are no probabilities associated with these activities. (Of course, it may be argued that there is a certain nondeterminism in DCCS, as represented by exceptional conditions in the plant and subsequent alarm situations arising. But as we shall see in this and subsequent chapters, these can be easily accommodated even in deterministic models.) Thus we have chosen Sifakis's timed petrinets for our system.

The various Timed Petrinets mentioned above will be discussed in detail later in the thesis to compare their applications with those of Sifakis's models.

3.2.3 Timed Petrinet Theory

Sifakis has defined and represented Timed Petrinets as follows -

Definition 1 A Petrinet (PN) is a quadruple $N = (P, T, \alpha, \beta)$ where

P : is a set of places, $P \neq \emptyset$

T : is a set of transitions, $T \neq \emptyset$, $P \cap T = \emptyset$

α : $P \times T \rightarrow \mathbb{N}$ forward incidence function

β : $P \times T \rightarrow \mathbb{N}$ backward incidence function

($\mathbb{N}$ represents the set of natural integers)

Representation To a PN one can associate a digraph, the nodes of which are the places and transitions, represented respectively by circles and dashes. There is a directed edge from the place p_s to the transition t_j iff,

$$\alpha(p_s, t_j) = n_{s,j} \neq 0$$

There is also a directed edge from transition t_r to place p_w iff,

$$\beta(p_w, t_r) = n_{w,r} \neq 0$$

A marking M of a PN, shown by dots in the place p , is a mapping of P

Definition 2 A Timed Petrinet (TPN) consists of

- a Petrinet $N = (P, T, \alpha, \beta)$
- $\mathcal{T} = (\tau_0, \tau_1, \tau_2, \dots, \tau_i, \dots)$ an increasing sequence of real numbers called a time base
- a mapping $v : P \times \mathcal{T} \rightarrow \mathcal{T}$ such that
for all $(p, \tau_i) \in P \times T : v(p, \tau_i) = \tau_j \Rightarrow \tau_j \geq \tau_i$

Rules

- a) A marker in a TPN may be in one of the two following states : **available** or **unavailable**. Initially, each place p contains $M_0(p)$ available markers.
- b) A transition t is enabled iff every place p_s , which has a directed edge to t , contains at least $\alpha(p_s, t)$ available markers.
- c) The firing of a transition t has to take place instantaneously as it is enabled. It consists in removing $\alpha(p_s, t)$ available markers from each of the places p_s and in placing $\beta(p_w, t)$ markers in each place p_w which has a directed edge from t . In the case of conflicting transitions, at least one transition fires instantaneously.
- d) A marker remains unavailable in a place p_s during the time interval between the instant of its arrival τ_i and the instant $v(p_s, \tau_i)$; then it becomes available.

Sifakis has studied the behaviour of pure TPNs such that

for all $(p_i, \tau) \in P \times \mathcal{T} : v(p_i, \tau) - \tau = z_i = \text{constant}$,
i.e., each marker is delayed by a time z_i in the place independently of its arrival.

Steady state functioning of a Timed Petrinet

For a TPN with n places and m transitions, the incidence matrix C is defined as

$$C = [c_{ij}]_{n \times m} \text{ with } c_{ij} = \begin{cases} \beta(p_i, t_j) & \text{if } \neq 0 \\ -\alpha(p_i, t_j) & \text{if } \alpha(p_i, t_j) \neq 0 \\ 0, & \text{if not} \end{cases}$$

In order to study the functioning of the TPN, two temporal vector variables are introduced: (A^t denotes the transpose of A)

into IN,

M
P $\rightarrow$ IN

Definition 2 A Timed Petrinet (TPN) consists of

- a Petrinet $N = (P, T, \alpha, \beta)$

- $\mathcal{T} = (c_1, c_2, \dots, c_i, \dots)$

an increasing sequence of real numbers called time base

- a mapping $v : P \times \mathcal{T} \rightarrow \mathcal{T}$ such that

$$\forall (p, c_i) \in P \times \mathcal{T} : v(p, c_i) = c_j \Rightarrow c_j \geq c_i$$

Rules

- a) A marker in a TPN may be in one of the two following states : **available** or **unavailable**. Initially, each place p contains $M_0(p)$ available markers.
- b) A transition t is enabled iff every place p_s , which has a directed edge to t , contains $\alpha(p_s, t)$ available markers at least.
- c) The firing of a transition t has to take place instantaneously as it is enabled. It consists in removing $\alpha(p_s, t)$ available markers from each of the places p_s and in placing $\beta(p_w, t)$ markers in each place p_w which has a directed edge from t .
- d) A marker remains unavailable in a place p_s during the time interval z_i after which it becomes available. Thus each marker is delayed by a time z_i in the place p_i where

$$\forall (p_i, c) \in P \times \mathcal{T} : c(p_i, c) - c = z_i = \text{constant}$$

Definition 3 For a TPN with n places, a vector $Q^t(c)$ is defined (Q^t denotes the transpose of Q), where

$$Q^t(c) = [q_1(c), q_2(c), \dots, q_n(c)]$$

and is called the charge variable at the instant c . Each entry $q_i(c)$ represents the marking at p_i at the instant c . For an interval $\Delta c = c - c_0$, the mean variation of the charge of the net is given by the vector

$$I(c) = Q(c) / \Delta c \quad \text{and is called the current vector.}$$

Each entry i_k of the vector I represents the mean frequency of firing of transition t_k during Δc .

Sifakis has shown that in order for the total charge in the net to

$Q^t(\tau) = [q_1(\tau), q_2(\tau), \dots, q_n(\tau)]$ representing the marking at p_i at the time τ .

$R^t(\tau) = [r_1(\tau), r_2(\tau), \dots, r_m(\tau)]$ representing the firing vector at the time τ .

The relation $M = M_0 + CR$ for a petrinet, where M_0 is the initial marking of the net, can now be written as

$$Q(\tau) = Q(\tau_0) + CR(\tau)$$

For an interval $\Delta\tau = \tau - \tau_0$, this gives

$$\Delta Q(\tau) / \Delta\tau = (Q(\tau) - Q(\tau_0)) / \Delta\tau = C R(\tau) / \Delta\tau = CI(\tau)$$

where

vector $Q(\tau) / \Delta\tau$ gives the mean variation of the number of tokens in the interval $\Delta\tau$

vector $I(\tau) = R(\tau) / \Delta\tau$ is called the current vector and each entry i_k of the vector I represents the mean frequency of firing of transition t_k during $\Delta\tau$.

Sifakis has studied the case where the firing frequencies of the transitions are constant, and the corresponding TPN is bounded. Then the TPN has a periodic functioning and I is a solution of

$$CI = 0, \quad I > 0 \quad \dots(1)$$

The current vector I depends on the initial marking and the delays associated with the places. Another relation that I must satisfy is

$$\{J_s^t Q(\tau_0) \geq J_s^t Z C^+ I\}_{s=1}^k \quad \dots(2)$$

where J is a place-invariant of the net,

$[J_1, J_2, \dots, J_k]$ is the set of place invariants corresponding to the set of elementary subnets of the net,

$$C^+ = [c_{ij}] \quad \text{with} \quad c_{ij} = \begin{cases} \beta(p_i, t_j) & \text{if } \beta(p_i, t_j) \neq 0 \\ 0, & \text{if not} \end{cases}$$

and Z is the (square) delay matrix given by $\text{Diag} [z_1 \ z_2 \ \dots \ z_n]$

The relations (1) and (2) describe the steady state behaviour of the TPN. For every periodic functioning, the current vector I satisfies

remain bounded, constant currents are to be assigned to the transitions. This results in **periodic** firings in the net.

The maximum computation rate corresponding to this periodic firing can be calculated by solving a set of n linear equations given as

$$CI = 0, \quad I > 0 \quad \dots(1)$$

$$\{ J_s^t Q(c_0) \} \geq \{ J_s^t Z C^+ I \}_{s=1}^k \quad \dots(2)$$

where C = incidence matrix of the TPN

$$= [c_{ij}] \quad \text{with} \quad c_{ij} = \begin{cases} \beta(p_i, t_j) & \text{if } \neq 0 \\ -\alpha(p_i, t_j) & \text{if } \alpha(p_i, t_j) \neq 0 \\ 0, & \text{if not} \end{cases}$$

$$\text{and } C^+ = [c_{ij}^+] \quad \text{with} \quad c_{ij}^+ = \begin{cases} \beta(p_i, t_j) & \text{if } \neq 0 \\ 0, & \text{if not} \end{cases}$$

$\{J^t\}$ is a generator of $\mathcal{C}$ where $\mathcal{C}$ is the set of non-negative solutions of $CX = 0$.

$Q(c_0)$ is the initial marking vector and

Z is the (square) delay matrix given by $\text{Diag}[z_1 \ z_2 \ \dots \ z_n]$

The net is said to function at a natural rate if the inequality (2) is changed to equality. The above linear equations can be written easily if the TPN is decomposed into subnets. In this respect, Sifakis has shown that searching for nonnegative solutions of $CI = 0$ (resp. $J^t C = 0$) amounts to decomposing the TPN into consistent (invariant) components, which correspond to State Graphs (Marked Graphs).

3.3 DEVELOPMENT AND ANALYSIS OF THE BUS CONTENTION MODEL

3.3.1 Timed Petrinet model

As mentioned earlier, Sifakis's TPNs can be used to great advantage in modelling for performance evaluation, the bus contention subsystem of the Message Bus multiprocessor. For simplicity, we shall discuss a two-processor system first. Later, it will be shown how this can be easily extended to an n-processor multiprocessor.

The following **assumptions** are made in the subsequent analysis -

these relations but the converse is not always true: a solution I_0 of these equations does not necessarily correspond to a feasible periodic functioning from M_0 . This is due to the fact that the relation (2) takes into account only the imposed delays z_i . In fact the delay of a token in a place p_i is the sum of the delay z_i and of a variable 'synchronization' delay due to the waiting of a token for other tokens to become available.

Functioning of a TPN at its natural rate

Let I_0 be a current vector of the TPN. Then I_0 is said to correspond to a functioning at a **natural** rate if I_0 satisfies the set of n linear equations given by

$$CI = 0, \quad I > 0 \quad (1)$$

$$\{ J^t Q(0) = J^t Z C^+ I \}_{s=1}^k \quad (3)$$

where $\{J_s\}_{s=1}^k$ is a base of the space of solutions of $C^t X = 0$.

Functioning at natural rate correspond to functioning at maximal rate: the delays of the tokens in the places are exactly equal to their unavailability times z_i . Obviously, every solution of this system of equations is a maximal solution of (1) and (2) and consequently for every vector I corresponding to a feasible functioning there exists a solution of (1) and (3) such that

$$I_0 \geq I.$$

In the following sections, we have interpreted this I as corresponding to the maximum (possible) computation rate of the net, with the given values of the delays z_i .

3.3 DEVELOPMENT AND ANALYSIS OF THE BUS CONTENTION MODEL

3.3.1 Timed Petrinet model

As mentioned earlier, Sifakis's TPNs can be used to great advantage in modelling for performance evaluation, the bus contention subsystem of the Message Bus multiprocessor. For simplicity, we shall discuss a two-processor system first. Later, it will be shown how this can be easily extended to an n -processor multiprocessor.

The following **assumptions** are made in the subsequent analysis -

they are **accessing** the bus, ie., sending information. Obviously, only one processor at a time can so access the bus. We assume that the accessing time of the processors are A1 and A2 resp. The values of A1 and A2 depend on the message being sent on the Bus and also the normal overheads incurred due to the error-control protocol operating on the bus.

The place p7 models the availability or nonavailability of the Bus at any instant. This is represented by the presence or absence of the marker in p7. The places p2 and p5 are 'dummy' places added in the TPN to complete the net between transitions t1 to t2 and t4 to t5 resp. There is no time delay associated with these places, the markers in them become available instantly on their arrival.

The transitions t1 and t4 model the event Bus Request by the processors P1 and P2. These fire as soon as the processor has completed its processing and wants to send a message on the bus next. The transitions t2 and t5 model the event Bus Acquired by the processors. The conflict between these transitions is clearly seen - only one will fire (if the bus is available) disabling the other. The transitions t3 and t6 correspond to Bus Release by P1 and P2. When any of these fire, the corresponding processor goes back to processing and the bus becomes available.

Each processor may have to wait for a certain time between the Bus Request and the Bus Acquired events, if the Bus is not available due to its use by the other. It will be shown that this waiting time is fixed for the processors when the TPN is in the steady state.

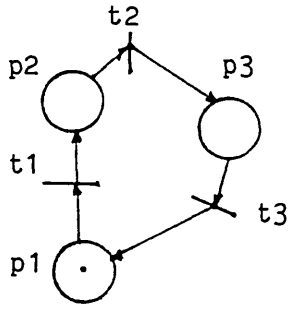
3.3.2 Calculation of **maximum** computation rate

For the TPN of Figure 3.2, we have $Z = \text{diag} [z1 \ 0 \ z3 \ z4 \ 0 \ z6 \ 0]$, where $z1 = P1$, $z3 = A1$, $z4 = P2$ and $z6 = A2$.

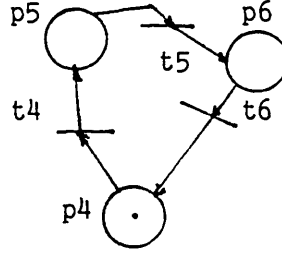
$$C = \begin{bmatrix} -1 & 0 & 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \\ 0 & 0 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & -1 & 1 & 0 & -1 & 1 \end{bmatrix} \quad Q^t(\text{co}) = [1 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1]$$

Solving $CI = 0$, we find $I = [i1 \ i1 \ i1 \ i2 \ i2 \ i2]$

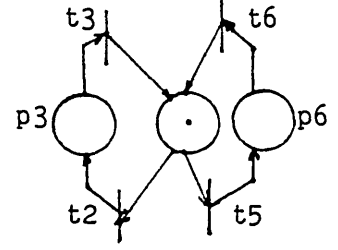
The cover by state graphs is given by the following figures:



SG1



SG2



SG3

and $J1^t = [1110000]$ $J2^t = [0001110]$ $J3^t = [0010011]$

The solution of the equations (1) and (2) give the maximum values of $i1$ and $i2$ as satisfying the conditions

$$1 \geq i1(P1 + A1) \quad \dots\dots (3)$$

$$1 \geq i2(P2 + A2) \quad \dots\dots (4)$$

$$1 \geq A1.i1 + A2.i2 \quad \dots\dots (5)$$

3.3.3 Interpretation of the maximum rate

The above calculation shows that each processor periodically processes and accesses the Bus and thus transfers a message at the maximum possible rate of $i1$ and $i2$ respectively. Thus the minimum time period for one such cycle is given by $Tp1 = 1/i1$ and $Tp2 = 1/i2$, where

$$Tp1 = P1 + A1 \quad \text{and} \quad Tp2 = P2 + A2.$$

The above 'minimum' signifies the fact that the actual periodic time is larger than $Tp1$ and $Tp2$ and takes into account the 'waiting' times for each processor between requesting the bus and acquiring it in each cycle, because the other processor is accessing it.

Let us denote by T the time in which each processor can actually access the bus once and let this be the common bus acquisition cycle time for the processors. Then, from the inequalities (3)-(5), we have

$$i1 \leq 1 / (P1 + A1) \quad \Rightarrow \quad i1 = 1 / (P1 + A1 + d1)$$

$$i2 \leq 1 / (P2 + A2) \quad \Rightarrow \quad i2 = 1 / (P2 + A2 + d2)$$

such that $T = P1 + A1 + d1 = P2 + A2 + d2 \Rightarrow i1 = i2$

$$\text{Then (5) } \Rightarrow \quad 1 \geq A_1 / (P_1 + A_1 + d_1) + A_2 / (P_2 + A_2 + d_2)$$

where d_i models the delay for each processor p_i in getting the Bus in each cycle. Two messages in all are transferred on the bus in this cycle time T .

Thus, for the given values of P_1 , A_1 , P_2 and A_2 there can be found a minimum value of T in which one processor accesses the Bus only once, but the other (with a smaller value of $P+A$) can access it m times in T , $m > 1$. This gives rise to $m+1$ messages being transferred in the time T , and $(1+m)$ is larger than 2.

Let us define the **Message Throughput Rate** M_t on the Bus as the total number of messages sent on the Bus in a cycle of time T . Then we can find the value of M_t in terms of the P and A times of each processor, using the relations given by the TPN. The delays d_1 and d_2 associated with each processor when it waits for the Bus to be acquired can also be found using the above equations.

Similarly, we define the **% Bus Utilization** B of the Bus as the total time in every cycle T that the Bus is being usefully accessed by the processors. In the next section we will see how the above three performance measures are calculated for the general case of n processors on the Bus.

3.4 TECHNIQUE FOR MAXIMIZING MESSAGE THROUGHPUT

The TPN model for a system with more than two processors is a simple extension of above case and is shown in Figure 3.3. It yields the following relations between the processing and accessing times

$P_0, P_1, P_2 \dots P_n$ and $A_0, A_1, A_2 \dots A_n$ of the $(n+1)$ processors:

$$i_0 \leq 1 / (P_0 + A_0), i_1 \leq 1 / (P_1 + A_1), \dots, i_n \leq 1 / (P_n + A_n).$$

and

$$A_0.i_0 + A_1.i_1 + A_2.i_2 + A_3.i_3 + \dots + A_n.i_n \leq 1 \quad \dots\dots\dots(6)$$

where $i_0, i_1 \dots i_n$ are the maximum possible rates for $P_0, P_1, \dots, P_n$ resp.

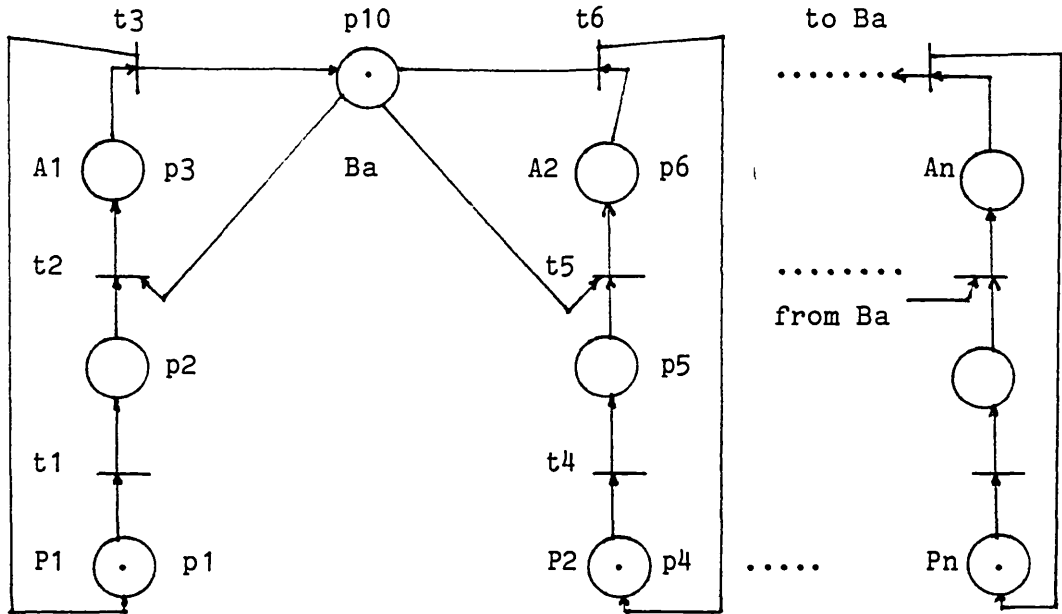


Figure 3.3 Bus contention model for n processors

Let T be the period of one complete cycle of bus acquisition. We will define it in the general case as **the time when each processor access es the bus at least once.**

Let us assume that the values of the P and A times are such that the processor with the largest value of $(P+A)$, designated P_0 , accesses once and the others access it $m_1, m_2, \dots, m_n$ times resp., ie., $i_0 = i_1/m_1 = \dots = i_n/m_n$.

$$\begin{aligned} \text{Let } T_0 &= P_0 + A_0 + d_0 \\ &= (P_1 + A_1 + d_{11}) + (P_1 + A_1 + d_{12}) + \dots + (P_1 + A_1 + d_{1m_1}) \\ &= m_1(P_1 + A_1) + d_{10}, \end{aligned} \quad \dots (7.1) \text{ etc., and}$$

$$\begin{aligned} T_0 &= (P_n + A_n + d_{n1}) + (P_n + A_n + d_{n2}) + \dots + (P_n + A_n + d_{nm_n}) \\ &= m_n(P_n + A_n) + d_{n0} \end{aligned} \quad \dots (7.n)$$

$$\text{where } d_{10} = d_{11} + d_{12} + \dots + d_{1m_1}$$

$$\dots \dots \dots$$

$$d_{n0} = d_{n1} + d_{n2} + \dots + d_{nm_n}$$

$$\text{where all } d_s \geq 0$$

$$m_1, \dots, m_n \geq 1$$

$$\text{Hence } (6) \Rightarrow A_0 + A_1.m_1 + \dots + A_n.m_n \leq (P_0 + A_0 + d_0) \dots (8)$$

$$\begin{aligned} \text{which gives } T &= \max. \{ (P_0 + A_0 + d_0), (A_0 + A_1.m_1 + A_2.m_2 + \dots + A_n.m_n) \} \\ &= P_0 + A_0 + d_0 \end{aligned}$$

The conditions (inequalities) (7) and (8) give the relations between the processor times P and A and the cycle time T and $m_1, \dots, m_n$. Hence

the Message Throughput Rate M_t is given by

$$M_t = (1 + \sum_{i=1}^n m_i) / T$$

The delays experienced by the processors in each cycle of the Bus acquisition are given by the values

d_0 for processor 1, $d_{11}, d_{12}, \dots, d_{1m_1}$ for $P_1, \dots$ etc., and $d_{n1}, d_{n2}, \dots, d_{nm_n}$ for processor P_n .

The % Bus Utilization factor B in each cycle T is given by

$$B = (A_0 + \sum_{i=1}^n A_i \cdot m_i) / T \times 100$$

The value of M_t as given by its definition can be maximized when T is minimum and $m_1 \dots m_n$ are maximum, provided the conditions (7)-(8) are satisfied. This is an optimization problem closely resembling problems in Operations Research [Christ79]. In fact, it can be rewritten and found equivalent to the well-known 'Knapsack' problem of Dynamic Programming [Taha81] as follows -

$$\text{Maximize } M(T) = (1+n) + \sum_{i=1}^n m_i$$

with the constraints

$$\sum_{i=0}^n A_i + \sum_{i=1}^n m_i \cdot A_i \leq T$$

and

$$0 \leq m_i \leq T/C_i - 1 \quad \text{where } C_i = P_i + A_i \text{ for all } i = 1, \dots, n.$$

which can be further simplified to

$$\begin{aligned} \text{Maximize } M(A) &= \sum_{i=1}^n m_i & \text{where } A &= T - \sum_{i=0}^n A_i \\ & & \text{and } M(A) &= M(T) - (1+n) \\ \text{with the constraint } & \sum_{i=1}^n m_i \cdot A_i \leq A \end{aligned}$$

This is the wellknown Knapsack problem with bounded variables and is solved recursively by the equation

$$\begin{aligned} M_k(A) &= \max. [m_k + M_{k-1}(A - A_k \cdot m_k)] \\ m_k &= 0, 1, \dots \end{aligned}$$

$$0 \leq m_k \leq \min \{ \lceil T/C_k - 1 \rceil, \lceil A/A_k \rceil \}$$

A simple computer program was written to maximize M_t for a given value of T corresponding to an arbitrary set of values of the processors' P and A times and the resulting values of $m_1 \dots m_n$ were obtained. The program is given in Appendix I.

(pgs. 63-64)

Figures 3.4a-b illustrate the case for 16 processors on a bus and show the resulting cycle followed by the processors to minimize their delays and achieve the computed Message Throughput.

Table 3.1 gives 2 sets of values used for P_0, A_0 and $P_i, A_i, i=1, \dots, 15$

Processor number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Processing times	78	19	14	4	6	15	8	10	17	16	12	12	8	15	17	12
Accessing times	6	3	4	4	5	3	3	2	7	3	5	6	5	5	3	6
Processing times	9	7	7	6	2	3	4	2	4	5	4	5	6	3	3	8
Accessing times	2	2	1	2	3	1	2	8	2	2	3	1	1	4	5	1

Table 3.1

In Figure 3.4a, $T = 84 = P_0 + A_0$,

$m_7 = 5$ and $m_1 = 3$ and all other m are $= 1$,

$d_0 = 0$,

$d_{11} = 2, d_{12} = 2$ and $d_{13} = 12$,

$d_{71} = d_{72} = d_{73} = d_{74} = 0$ and $d_{75} = 24$,

all other d 's are 0,

giving $M_t = 22/84 = 0.261$

and $B = 100\%$

(If all m were 1, the value of M_t would be $= 16/84 = 0.19$)

In Figure 3.4b, $T = 52 = P_0 + A_0 + d_0$ where $d_0 = 41$

$$= \sum_{i=1}^{15} A_i + 12$$

For this case, we have chosen a larger value of T than required, to show that the value of M_t can be maximized to the extent desired by simply increasing the cycle time of bus acquisition, thereby increasing the individual processor delays.

$T = 52$ gives $m_5 = 9$, $m_2 = 5$ and all other m_s are 1, so that

$$M_t = 28/52 = .638$$

$$\text{If } T = \sum_{i=0}^{15} A_i = 40, \quad M_t = 16/40 = .40$$

For both values of T , $B = 100\%$.

3.5 DISCUSSION

3.5.1 Implications on choice of bus arbitration policy

The above results indicate a very important implication regarding the choice of the bus arbitration policy to be used in a shared bus multiprocessor. It shows that in order to maintain the highest possible message throughput rate on the bus, it is necessary to maintain the cyclic acquisition shown by Figure 3.4. In other words, each processor should be able to access the bus in the correct order in each cycle. The bus arbitration policy chosen for the multiprocessor should ensure that this cycle is followed even in the face of any contention that is likely to arise.

The simplest arbitration policy used for multiprocessors is priority based. But priorities are usually allocated to the processors on some ad-hoc basis, eg., message priority. No consideration is given to the processing and accessing times of the processors. It is clear from Figure 3.4 that such an adhoc- priority based access policy may give rise to long delays and sometimes even starvation to some low-priority processors. No periodic bus acquisition will then result.

Figure 3.4a shows that the priorities p to be allocated to the processors in our chosen example are related as follows -

$$\begin{aligned} pP7 &> pP1 > pP8 > pP2 > pP11 > pP9 > pP3 > pP4 > \\ pP10 &> pP5 > pP6 > pP12 > pP13 > pP14 > pP15 > pP0 \end{aligned}$$

For the Message Bus multiprocessor station, we have chosen a priority based bus access mechanism. By allocating the priorities in such a manner that the cyclic bus acquisition resulting from the solution of the maximization problem is ensured, we can achieve minimum delays on

the bus.

These priorities can be modelled in the TPN diagram by the use of Inhibitor arcs [Peters81]. Figure 3.5 shows such a TPN diagram where only the processors P1, P7 and P8 are shown for simplicity.

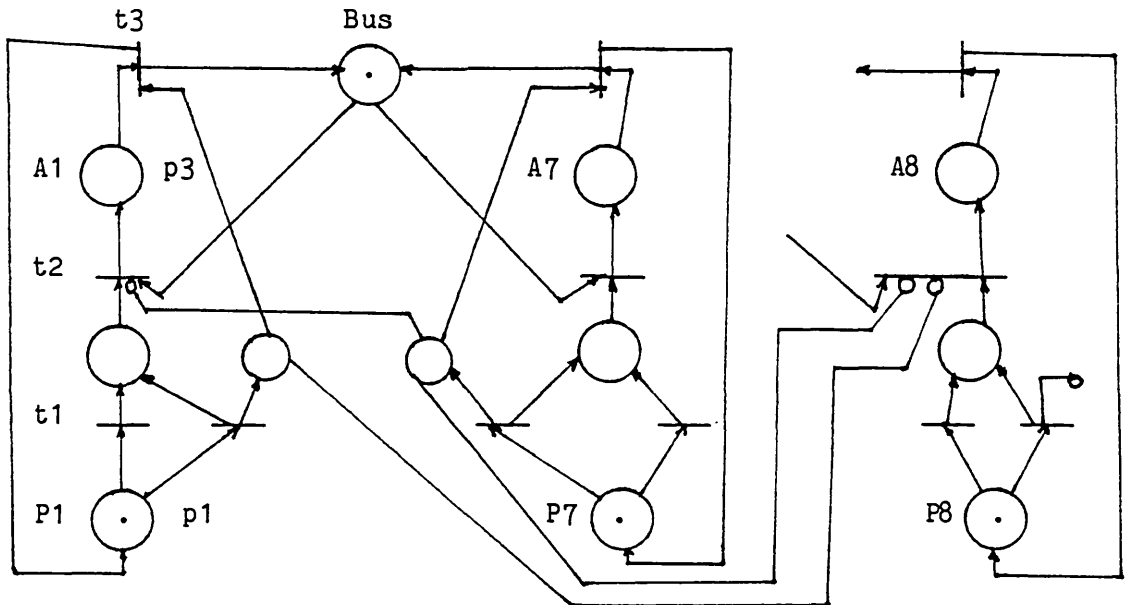


Figure 3.5 Bus contention model for n processors with priority

It is clear from the foregoing discussion that if any other bus arbitration policy, eg., CSMA/CD [Metcal76] is followed for our DCCS multiprocessor, large delays will result, or possibly starvation.

The VDPA bus allocation policy suggested for the HXDP distributed processor [Jensen78], could also be used to ensure maximum throughput and cyclic acquisition, by allocating the pattern of 1's in the circular vector in each processor Bus Interface properly. But the implementation of such a bus arbitration mechanism is much more complex than the simple priority based one chosen for the Message Bus. More importantly, the former does not cater to the situation where an 'alarm' message may have to be sent on the bus. It is not clear how such infrequent but important, run-time dependent situations are dealt with by the VDPA scheme.

3.5.2 Implementation considerations

The above assigned priorities can be easily implemented on the Message Bus by providing them in the Bus Interfaces. The priority vector in the BI is simply hardwired for the above priority. In addition, the

bus access cycle corresponding to the computed value of T is fixed by two constraints on each transmitting side. These are -

- a) the processor p_i can access the Bus only m_i times in each cycle
- b) each cycle corresponds to and is culminated by M_t Bus Release signals.

These two values can either also be jumpered onto the BI hardware or programmed into it.

The above implementation does not violate the flexibility requirement of the Bus Interface system because -

- a. By representing one Bus cycle in terms of a number of Bus Release signals rather than actual time, the length of the cycle can vary at any instant to include protocol overhead changes, without disrupting the periodicity of accesses.
- b. When a new processor is added (or removed) from the bus, the values of m_i and M_t and hence the priorities will change and can be simply rewired or reprogrammed on to the BI hardware.

3.5.3 Extensions to the model

The bus contention model shown in Fig. 3.2 was chosen because it best represents the characteristics of the Message Bus multiprocessor. The versatility of such a TPN model is easily demonstrated by showing that a change in the assumptions can be immediately represented with a slight modification of the net, resulting in a different system being modelled. We will discuss two such extensions in the following paragraphs.

- a. **Non zero bus acquisition time** The place p_7 in Figure 3.2 was not allocated any time delay, ie., $z_7 = 0$ there. This models the fact that there is no delay in the next processor acquiring the bus after the previous one has released it. This is truly representative of the situation on the Message Bus. Due to its WIRED-AND implementation, bus acquisition is instantaneous after the last one. Bus arbitration by sending the priority byte on the bus first takes only one byte time and can be considered negligible or at most, part of the bus accessing time.

In some multiprocessors, especially those that use a Central Arbiter for allocating the bus amongst the different processors, a certain

non-zero time is taken up for arbitrating the bus. This is also characteristic of Local Area Networks like Ethernet [Metcal76], where the CSMA/CD mechanism takes a large time to acquire the network for the processor, in a high load environment.

The above bus characteristic is easily incorporated in the model by letting $z_7 > 0$. This changes the inequality (5) of the existing TPN model to

$$1 \geq \sum_{p=1}^n (A_p + z_7) \cdot i_p \cdot m_p \quad \text{for } n \text{ processors on the bus.}$$

The net effect is to decrease the maximum allowable values of the computation rate for each processor. The time of one common cycle T increases by an amount equal to $n \cdot z_7$. Thus delays for each processor increase, bringing the message throughput rate down by a factor of $T/(T+nz_7)$. An important observation here is that in such cases, the % bus utilization is always less than 100, since the bus is idle for a time z_7 after each release.

b. Multiple buses The TPN model of Figure 3.2 allows a single marker in the place p_7 , representing the fact that there is only a single bus available to the processors. This is of course the Message Bus in our multiprocessor.

Many multiprocessor systems have more than one bus between the processors to increase reliability, eg., the SIFT multiprocessor [Wensel78]. This fact can be easily represented in our TPN model by allowing the number of markers in place p_7 to be greater than one. This number b will normally be less than the total number of processors n in the multiprocessor. The inequality (5) is now modified to

$$b \geq \sum_{p=1}^n A_p \cdot i_p \cdot m_p$$

This means that the computation rates i_p now approach more closely the maximum allowable value of $i/(P_p + A_p)$ for each processor p .

The corresponding delays for each processor are considerably reduced. The message throughput rates have to be redefined in order to accommodate all the available buses in the system. But the technique outlined above can still be used to find the maximum possible message throughput for the multiple buses.

3.5.4 Performance prediction of the multiprocessor

The calculation of the delays associated with each processor in acquiring the bus gives us a valuable insight into many other related performance measures for the Message Bus multiprocessor. It is possible to evaluate its performance in terms of the achievable **Speed up** and **Efficiency**. These performance indices are being increasingly used to evaluate and predict multiprocessor performance as compared to a uniprocessor [Agarwa83, Zubere80].

Speed up Sp (or speed gain factor as it is sometimes called) is a measure of the speed gain in a p -processor multiprocessor configuration over a uniprocessor one. It is defined as

$$Sp = T1/Tp \quad \text{where } Tp \text{ is the time taken to perform any computation using } p \text{ processors.}$$

Efficiency E of the multiprocessor is then given by the ratio $E = Sp/p$

Obviously, Sp is always less than p and E is less than 1, because of the loss in performance in a multiprocessor due to bus contention, memory interference and communication protocol overheads. The value of these indices can be increased considerably by proper allocation of the processing 'tasks' to the various processors in the multiprocessor as given in [Chu80, Price81 and Ma82]. This allocation is mainly concerned with minimizing the inter-task communication on the bus.

Let us assume that in our n -processor model of the Message Bus, the tasks have been so allocated to the processors that the processing times for each are the given values P_k , $k = 1, 2, \dots, n$. These tasks execute concurrently in the respective processors. In order to perform the overall computation, each (task) processor has to communicate on the bus and let the bus accessing times be A_k , $k = 1, 2, \dots, n$.

Then the value of the bus cycle time period T , in which each processor has completed its own task once (and hence the overall computation), represents the time Tp mentioned above. If, on the other hand, the same computation was to be done by a uniprocessor, the time taken would be equal to the sum of the processing times, since the tasks

would be executed sequentially.

The values of Speed Up and Efficiency are thus given by

$$S_n = \frac{\sum_{k=1}^n P_k}{T} \text{ and } E = S_n / n$$

For the 16 processor configuration of Figure 3.4b, with the values of P_k and A_k as given in Table 3.1,

$$S = 78/40 = 1.95 \text{ and } E = 1.94/16 = 0.12$$

The case where the processors can access the bus more than once in T can be easily accommodated to give

$$S = (P_0 + \sum_{i=1}^n m_i \cdot P_i) / T$$

and for the case under study, $S = 130/52 = 2.5$

$$E = 2.5 / 16 = 0.155$$

It is to be emphasized here that the poor values of S_p and E calculated above are due to the purely arbitrary choice of values taken for processing and accessing times in the example considered. Our purpose in doing so has only been to show S_p and E can be calculated using the results of TPN analysis. More realistic values will be considered in Chapter 6 where the composite multiprocessor model is discussed.

3.5.5 Discussion of assumptions

a) **Effect of protocol overhead changes** We assumed earlier that the message transmission times of each processor are fixed. This is not strictly true because the situation on the bus at any instant may necessitate retransmissions of messages and thus change the effective transmitting time for it. This change will increase bus acquisition delays in that cycle for all other processors, and hence reduce the message throughput for that cycle. But the periodic nature of the access will still be maintained because of the priorities allocated to processors and the steady state will be restored in the subsequent cycles.

b) **Inclusion of alarm messages** Exceptional conditions occur in any system, requiring the sending of 'alarms' which are considered high

priority messages. These have not been included in our discussion, but can be catered for, separately. One way of implementing these is that the alarm sending processor be allowed to temporarily raise its priority to a previously set maximum (which is higher than those allocated to the processors) and acquire the bus to send the alarm.

These can be modelled in the TPN diagram by having another transition indicating the presence of the 'alarm ready' condition for every processor. This is shown in Figure 3.6 for a two processor system. Normally transition t_i is fired for the processor at the rate i_i , say. But when an alarm has to be sent, transition t_{ia} fires instead. The rate for this is i_{ia} where

$$i_i \gg i_{ia}$$

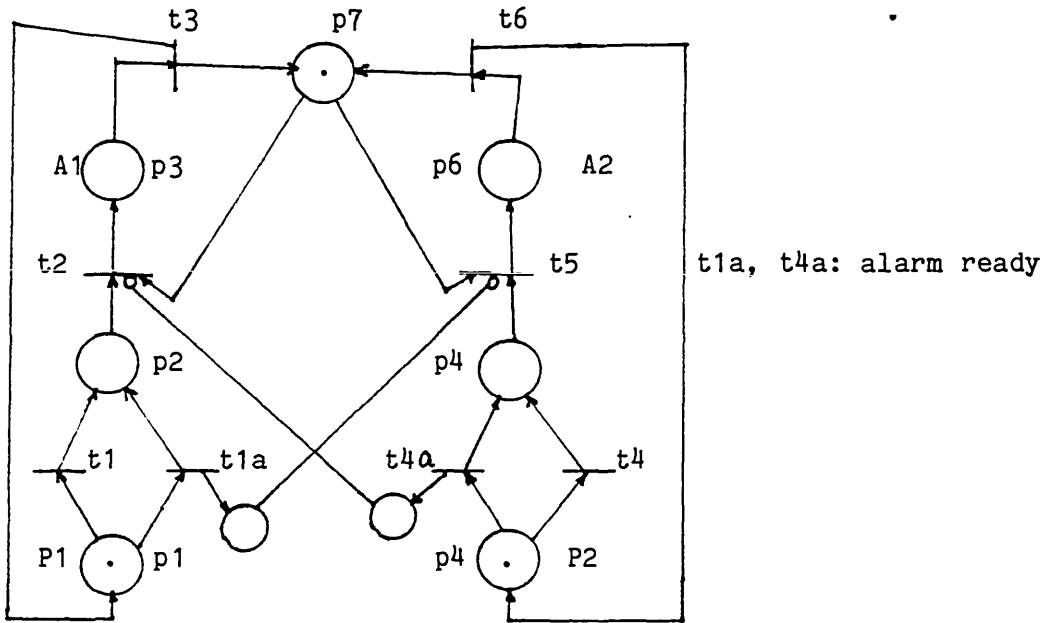


Figure 3.6 Bus Contention TPN model for two processors with alarm message sending included.

A little thought will show that the analysis will not be affected. The only change that occurs is that transition t_{ia} has priority over t_i and this is shown by the inhibitor arcs. Note that the normal priority arcs of Figure 3.5 are not shown here.

Again, only the current bus access cycle will be disrupted because the priorities are changed, but the steady state is restored subsequently.

3.6 COMPARISON WITH RELATED WORK

3.6.1 Stochastic TPN models for multiprocessor bus contention

Marsan et al [Marsan82] have proposed a Stochastic TPN model for representing the bus contention and memory interference in tightly coupled multiprocessors and have used their TOMP system to demonstrate its use. TOMP is an experimental, general purpose multiprocessor in which each processor has access to its private memory, a local common memory and also some nonlocal common memory via its private bus, local bus and global bus respectively. The bus arbitration units use a fixed priority policy for the global bus and a FIFO one for the local bus.

The Stochastic Petrinet (SPN) which models this system uses TPNs in which random firing times are associated with transitions. These times are determined by the assumptions on the multiple processor workload.

Marsan et al have used the concept that SPNs are isomorphic to continuous time Markov Chains (MC); SPN markings correspond to MC states, making it possible to obtain the steady state probability of each marking and from it to derive performance indices. The index chosen is P , the **processing power**, which is the average number of active processors.

A two processor SPN model has been analysed using its associated MC. The processing power has been found for varying load factors which are defined as the ratio of the average accessing to processing times of the processors. A Markov model and a Queueing Network model have also been studied for the system and compared with the SPN model.

Marsan et al argue that although the SPN models give powerful and precise representations of distributed systems, their main disadvantage is the explosion of states that occurs in the related MC, when larger systems are represented. For a 3-processor system, the number of states increases to 395, as compared to only 25 in a 2-processor system. In our TPN model, this explosion does not occur, as the number of states, (in terms of the set of allowable markings) increases from 8 in the 2-processor case to only 20 in the 3-processor case. One of the reasons for this is that the Message Bus multiprocessor exhibits loosely coupled characteristics, and has a single bus,

which considerably simplifies the problem.

Again, whereas Marsan et al have used the theory of Markovian Stochastic processes in arriving at the performance index evaluation, we have done the same using the comparatively simpler fixed-delay TPN theory of Sifakis and have been able to use standard Non-linear Programming methods to simplify our problem. Moreover, we do not need any state exploration analysis in fixed delay TPN theory, whereas it is the mainstay of Markov Process analysis.

3.6.2 Analysis of real time multiprocessor systems [Marken82]

The author has come to many similar conclusions as ours in this work, but she has not used petrinets for her analysis of real time multiprocessor systems. Two main conclusions which are the same as ours are -

- real time systems are best analyzed and approximated by deterministic models,
- the real time system operation should be considered as becoming periodic with constant cycle times.

She has discussed Direct Shared Bus (DSB) architectures, including both single and multiple bus systems, and for the multiple bus case, she has proposed two bus scheduling schemes that can be used on the buses so as to maximize throughput on them.

The main difference, as compared with our work, is that for the single bus case, it has been proved that the throughput is independent of the order in which the processors transmit data on the bus. We, on the other hand, have argued that message throughput can be maximized by allowing a certain priority based order for processors, in arbitrating the bus.

Actually, these seemingly opposed views are both correct. This is because in our case, we have defined an overall task to be one in which some processors may transmit more than once. If we consider the case where execution of an overall task implies only one access by each processor, then the same situation, and hence conclusion, arises.

Another difference is that the Non Fixed Allocation Bus Scheduling (NFABS) problem proposed for the multiple bus case, has been proved to

be NP-complete and hence requires a heuristic method for its solution. Our problem, which we have identified as equivalent to a bounded Knapsack problem, can be proved to be solvable in pseudo-polynomial time, ie., it is not strongly NP-complete and hence it was possible to write a program to solve it.

3.6.3 Time Valued Petrinets for control algorithm decomposition

In [Nader79] a slightly different approach is taken in the use of Timed Petrinets for multiprocessor performance analysis. Nader has shown how to model, decompose and analyze central real time control algorithms for parallel processing on multiprocessors, using what he has called Time Valued Petrinets (TVPNs). These TVPNs are similar to Sifakis's TPNs in that fixed times are associated with places. But places have been interpreted as being of two different types - as process execution places and data transfer places. Each has been assigned a time value t_{ei} or t_{ti} respectively. The firing of transitions can be made in two ways - using time dependent events, ie., after a predetermined time has elapsed, as in Sifakis's TPNs, or time independent events using correctness criteria, ie., post or pre conditions for completeness or start of a process execution or data transfer.

Using the above concepts, he has used simple arithmetic to find the minimum execution times for purely sequentially processing an algorithm on a single processor, using time dependent firings. This has been compared to the times taken to execute the algorithm when it is decomposed in various different ways for parallel processing on more than one processor. The method has been applied to a real time multivariable model reference adaptive control algorithm to show its ease and power of use.

The above approach leads to the same performance index ie., speed up that we have used in our model. But whereas we have used a bus contention system and its associated waiting times to arrive at Speed up, disregarding task allocation criteria, Nader has based his model only on the proper decomposition of algorithms being executed on multiprocessors. He has not taken into account the interconnection system between the processors. His minimum times are only dependent on process execution times, not on data transfer times also, as in our case.

3.7 CONCLUSION

A Timed Petrinet model of the Message Bus contention system has been developed and analysed to illustrate how Timed Petrinets can be used for the performance analysis of such systems. We have shown how TPN theory leads to a procedure to calculate the least possible cyclic time period of bus acquisition in a multiprocessor where each processor accesses the bus at least once. This results in the maximum achievable message throughput rates on the bus for given values of processor processing and accessing times.

The throughput maximization problem is actually NP-complete, but not in the 'strong sense'; hence we have been able to solve it in psuedo-polynomial time using a simple computer 'Knapsack' program.

We have also shown that it is possible to arrive at the usual performance measures of multiprocessors, ie., Speed Up and Efficiency, using the results for cycle time from the TPN analysis. While this chapter only uses purely arbitrary values of processing and accessing times for processors in calculating Speed Up etc., Chapter 6 will consider realistic values for heterogenous processors, to show how an analysis of the contention subsystem can be used for the overall performance validation of a multiple processor system.

We have used Message Throughput as a performance measure that we want to maximize in the system. It is usually argued that for Distributed process control systems, Response Time is the better measure. While we agree to this, we would like to stress that we are dealing here with an environment in which multiple processors cooperate to perform a given task by passing messages in minimum time, rather than one in which two processors communicate for a query-response transaction, which necessitates minimum response time requirements.

One of the main results obtained from the foregoing study is that the bus arbitration policy chosen for a real time multiple processor system should not be ad hoc, but compatible with cyclic bus acquisition. As such we have validated the priority based bus arbitration mechanism chosen for the Message Bus multiprocessor as being the correct design choice, since it allows such a cyclic bus access.

In contrast to the generally prevailing concept, it was found that Timed Petrinet analysis does not always lead to an explosion in the number of states, complicating the study of large scale systems. Our three processor case, for example, would have only 20 possible states; but even these play no part in the theoretical analysis of the model.

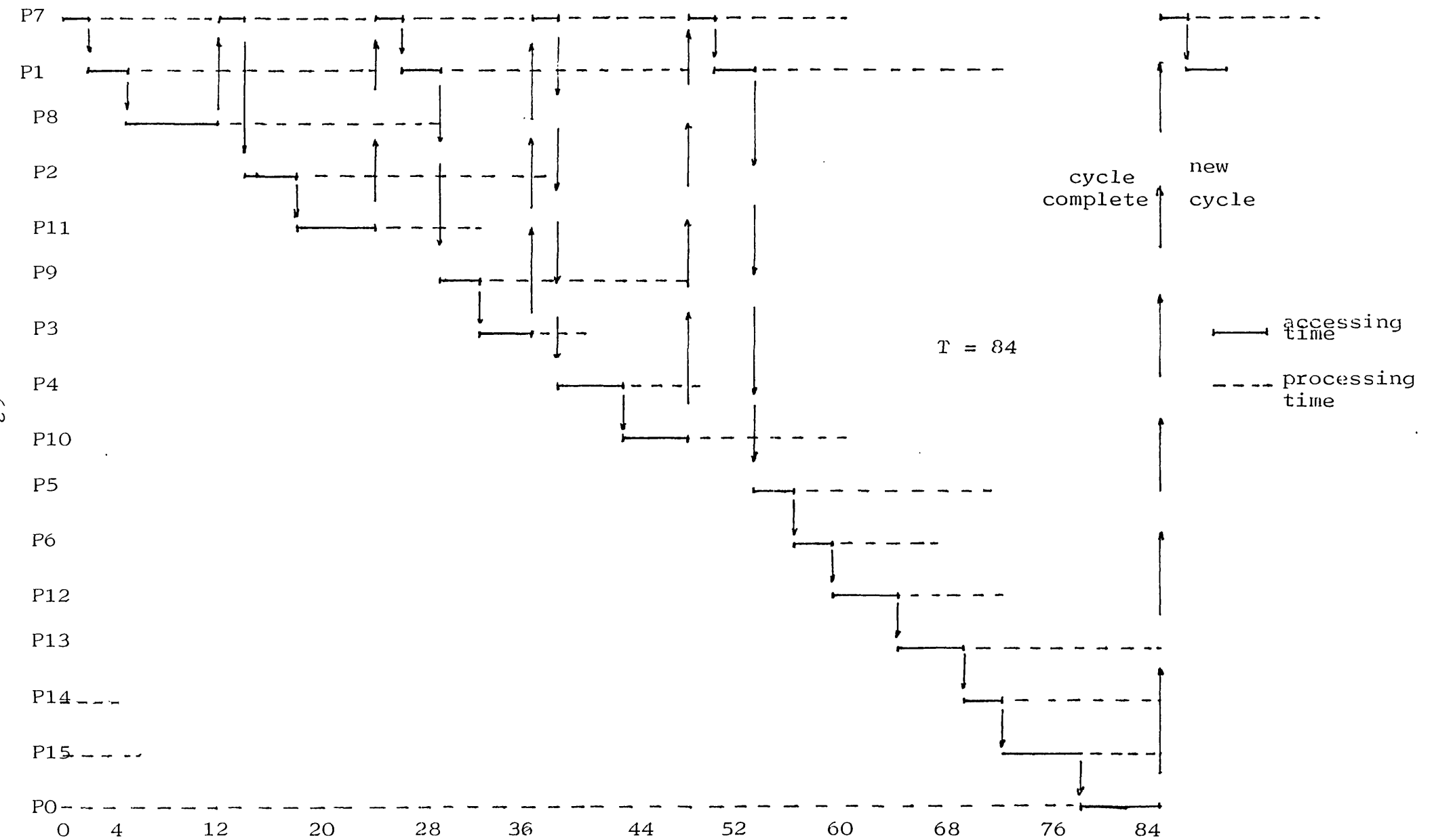


Figure 3.4a Bus Access Diagram for 16 processors

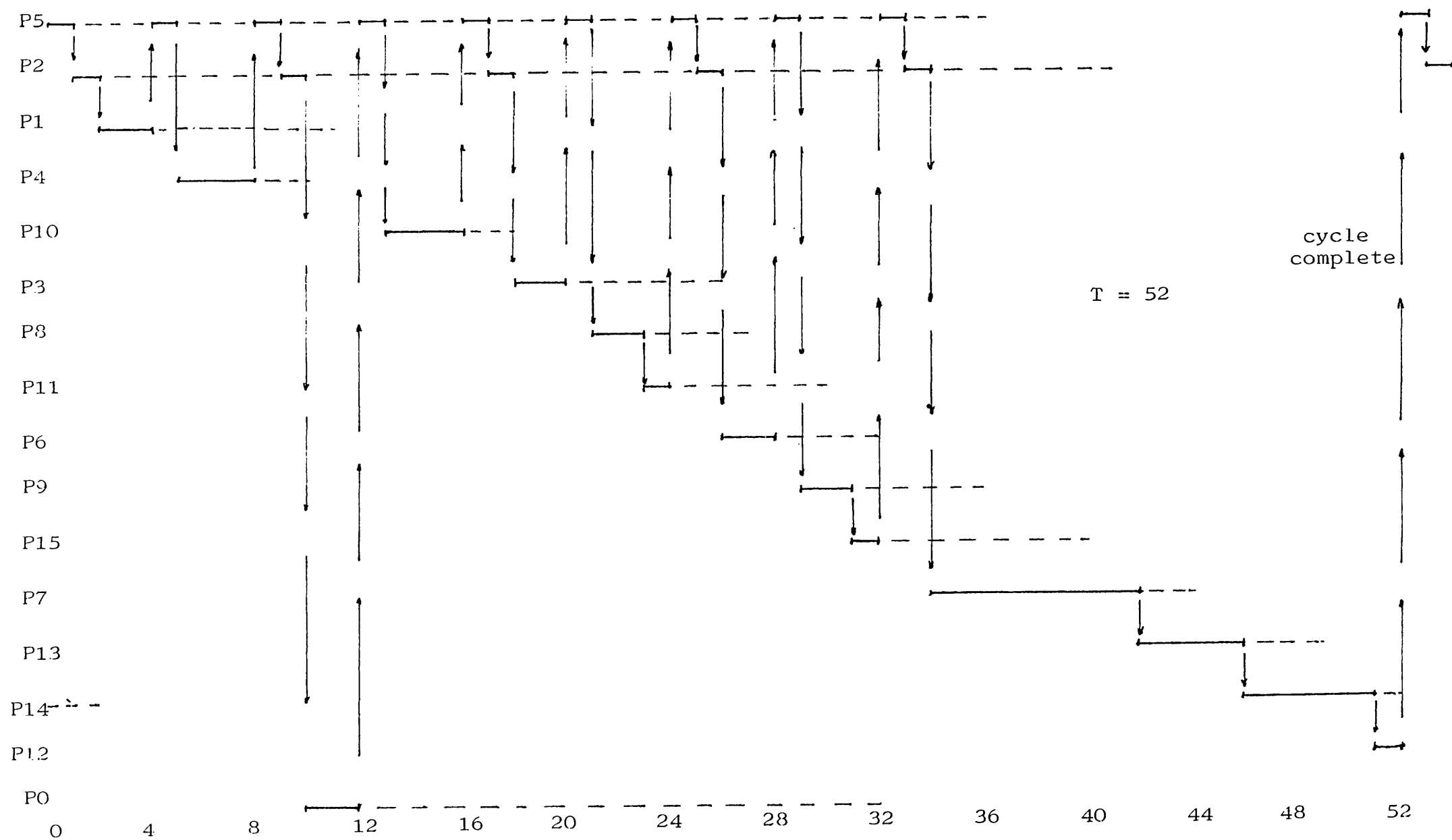


Figure 3.4b Bus Access Diagram

FORMAL SPECIFICATION AND VERIFICATION OF INPAR PROTOCOL

4.1 INTRODUCTION

This chapter is concerned with the formal specification and verification of a communications protocol, using its Timed Petrinet model. In other words, it shows how Timed Petrinets can be used to **validate the functional design** of a protocol.

A protocol functional design can be validated by proving that it functions correctly and provides the intended services to its user, and by proving certain **Safety** (correct delivery), **Liveness** (effective work being done), and **Performance** (work being done fast enough) properties of the protocol [Sunshi82b].

To do this, it is necessary to give a formal and rigorous specification for the protocol and to verify that it meets its specification. This is because it is now widely recognized that it is not enough to merely give an informal narrative description for a communication protocol or to validate it in an ad hoc manner, since faults have been shown to arise in such systems.

Specification and Verification Assuming that the communication architecture of a distributed system is normally structured as a hierarchy of different protocol layers [Sunshi82a], with each layer providing a particular set of **Services** to its users above, then the description of the input/output behaviour of the protocol layer constitutes a **Service Specification** of the protocol.

In a network environment with physically separated users, a protocol layer must be implemented in a distributed fashion, with 'entities' local to each user communicating among one another via the services of the lower layer. The interaction among entities in providing the layer's service constitutes the actual protocol. Hence a **Protocol Specification** must describe the operation of each entity within a layer in response to commands from its users, messages from the other entities (via the lower layer service) and internally initiated actions (e.g., timeouts).

Both the specifications described above must embody the key concept of **Abstraction** if they are to be successful. In other words, they must include only the essential requirements that an object must satisfy and omit the inessential.

Verification is essentially a demonstration that an object meets its specification. For the two major classes of objects requiring specification for a protocol layer, ie., services and protocol entities, the basic verification problem that must be addressed is: verification of the protocol **design** by analyzing the possible interactions of the entities of the layer, each functioning according to its (abstract) protocol specification and communicating through the underlying layer's service and to see whether this combined operation satisfies the layer's service specification.

The service specification itself cannot be verified, but rather forms the standard against which the protocol is verified. But it can be checked for consistency. It must also properly reflect the users' desires, and provide an adequate basis for the higher levels using it.

In addition, there are certain implicit requirements for a protocol, which are given as a set of **properties**. These are concerned with its safety, liveness and performance and include:

- * freedom from deadlock, or arrival at some desired final state(s);
- * completeness of the protocol in handling all situations that may arise during execution;
- * progress, or the absence of cycles where no useful work is done;
- * stability, meaning that the protocol will return to a normal mode of operation after an abnormal condition occurs;
- * minimal delay in performing the functions of the protocol.

There are a number of ways of formally specifying and verifying protocols. A variety of general formalisms such as state diagrams, Petri nets, grammars and programming languages and their adaptations and extensions have been used for specification. These may be classified into two main categories: transition models and programming languages, but combinations and extensions of these are also followed. Most approaches to protocol verification are closely coupled with some

specification method while others are broadly applicable. They include, for example, reachability analysis or state exploration techniques which are straightforward to apply to transition models of protocols. Program proofs are also used in which assertions are formulated and proved for the desired properties of the protocol. Another method used is that of Structural Induction which is the basic proof method used with abstract data type specifications. We will discuss this in detail in the next section.

While a large body of work exists on the formal specification and verification of communication protocols using ordinary petrinets, various extensions of these petrinets are being studied [Diaz82] in order that both control and data transfer aspects of protocols may be analysed formally. But it is only recently [Bertho83] that Timed Petrinets have been used for this purpose. These are being used, for example, to model effectively, the 'timeout' mechanism and its effect on protocol operation. Earlier, Merlin had used the time concept in Petrinets [Merlin74], to state timing conditions for recovery of protocols after faults.

We have used Timed Petrinets to model the protocol behaviour in order to prove its correctness, and to show that they are ideal tools for modelling a protocol's liveness and performance characteristics also, and for this reason, are better than state transition machines.

We have used the notation of the AFFIRM system [Sunshi82b] to formally specify the protocol service and to verify that it meets its requirements. The need for using AFFIRM arose because, firstly, it is not possible to verify a protocol service against its specification using only Timed petrinet analysis; and secondly, because the abstract data type approach for specification, and the associated Structural Induction method of verification used in AFFIRM, help avoid the state explosion usually encountered with a purely transition model approach. The AFFIRM method has been modified slightly for use with Timed Petrinets. The protocol chosen for illustration is the INPAR protocol designed in Chapter 2 for the Message Bus based multiprocessor.

In Section 2 we give a brief description of the AFFIRM system and its method for formal specification and verification for protocols. In Section 3 the Message Bus Service is modelled by a Timed Petrinet and specified in AFFIRM. This is followed by its partial validation.

Section 4 gives the INPAR protocol specification, followed by its verification against the required service in Section 5. We prove its safety property only in this chapter. This chapter can thus be considered as laying a basis for the work on Protocol Liveness and performance specification, using Timed Petrinets, which will follow in the next chapter. A discussion and conclusion follows in Section 6.

4.2 THE AFFIRM SYSTEM

AFFIRM [Sunshi82b] is a general purpose experimental system for the algebraic specification and verification of user defined abstract data types, being developed at ISI, California. It uses the State Transition Machine for modelling the behaviour of systems, rendering these in the AFFIRM specification language, expressing its properties algebraically as theorems and proving them using an interactive theorem prover. Most of their work has focussed on the 'Safety' property of protocols, but techniques for specifying its 'Liveness' and 'Performance' have not been considered. This, we believe is due to the shortcomings of State Transition Machines in modelling these characteristics.

In the AFFIRM notation, an **abstract data type** is specified as follows-

- (i) define its three functions, namely, constructors, selectors and extendors, [Guttag82]
- (ii) express the extendors and selectors by equational axioms, relating how each function behaves when applied to each of the constructors,
- (iii) treat these axioms which look like equations, as left to right rewriting rules and use them to state and prove certain invariant properties of the data type being specified,
- (iv) prove the theorems by Structural Induction, ie., based on the constructors of the data type.

The various steps involved in Protocol specification and verification using the AFFIRM system are given briefly below. Details of AFFIRM and its method for Protocol Specification and Verification can be found in [Sunshi82b].

- (i) Produce a service specification of the protocol by writing down its state transition machine description in AFFIRM.
- (ii) Validate that the service specification at least partially meets

- the user requirements by proving some invariant properties.
- (iii) Produce the protocol specification by writing its state transition machine representation in AFFIRM.
 - (iv) Verify that the protocol specification implements the service specification.
 - (v) Specify an algorithm implementing the protocol specification.
 - (vi) Verify that the algorithm implements the protocol.

We have found that the algebraic notation of AFFIRM can be used, with suitable extensions to the data types defined, to formally state and prove not only the safety properties, but also liveness and performance properties of a protocol. This is done by first giving a complete specification of the protocol using Timed Petrinets instead of State Transition Machines. All of the AFFIRM notation is still valid for a Petrinet which is actually a general case of a State Transition machine. In fact, a state transition machine is a petrinet if the number of input and output places for a transition is always equal to one.

4.3 THE MESSAGE BUS SERVICE SPECIFICATION

An Informal description of the Message Bus service We have described the Message Bus service in detail in Chapter 2 while considering the design of the bus. It is repeated briefly here for completeness. Its Timed petrinet model is shown in Figure 4.1.

- * The message bus provides for transmission of one message at a time from one of many senders to one of many receivers
- * Every potential sender requests for the bus and may have to wait before it can acquire the bus, before sending
- * The address of the receiver is sent on the bus along with the message and the receiver recognizes this address to receive the message
- * After the message has been received, the sender releases the bus to indicate end of a transaction
- * If the message cannot be correctly sent or received, this fact is reported to the sender.

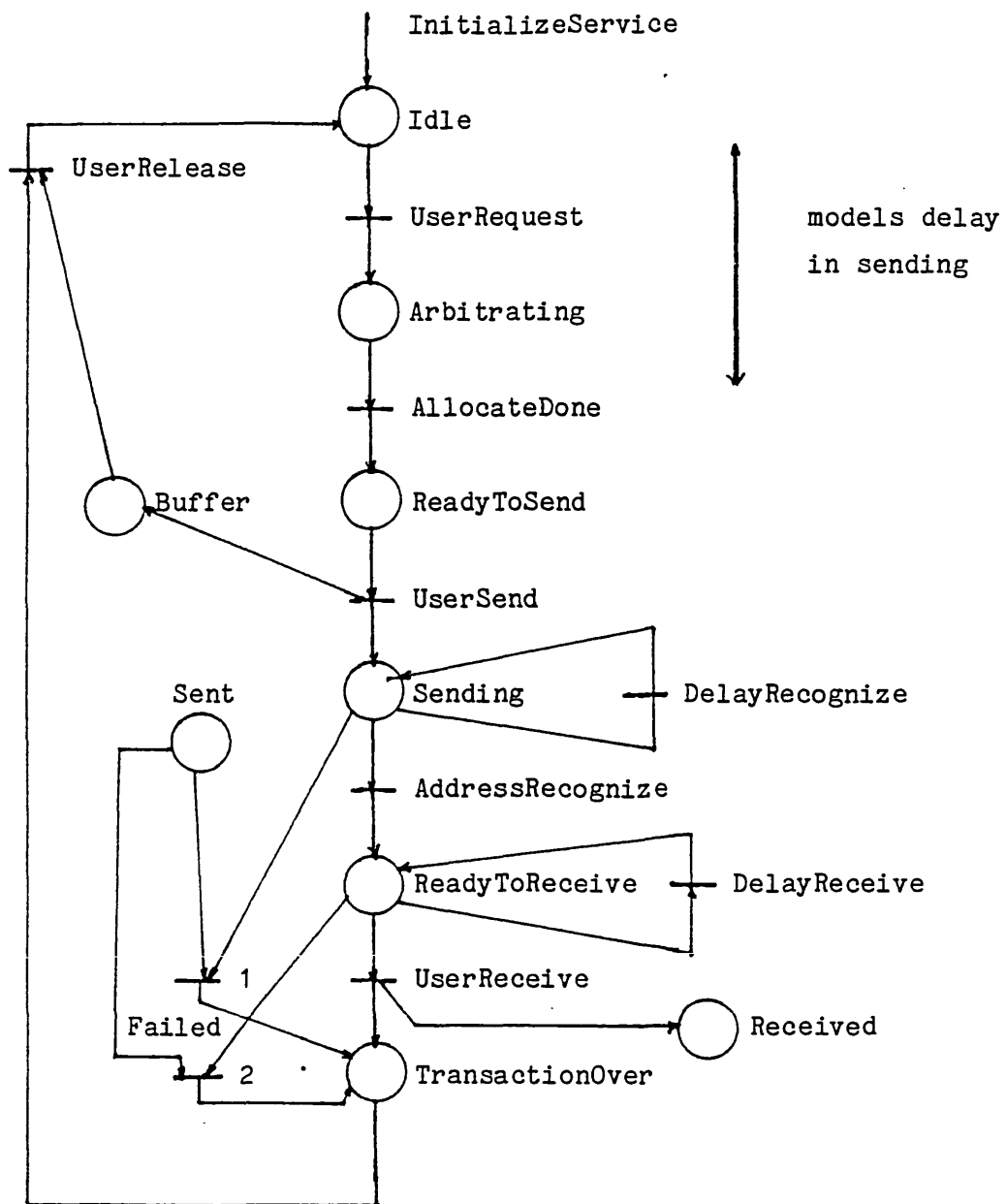


Figure 4.1 Timed Petrinet Machine for Message Bus Service

The formal description of the Timed Petrinet machine of the Message Bus system follows next. It uses the following data structures -

State Variables

State - ControlState: The current status of the service. This state variable simply cycles through the six states (values) of the enumerated type **controlState**. These values are **Idle**, **Arbitrating**, **ReadyToSend**, **Sending**, **ReadyToReceive** and **TransactionOver**.

Sent - QueueOfMessage: The queue of messages that have been sent to the receiver.

Received - QueueOfMessage: The queue of messages that have been

received by the receiver.

Buffer - QueueOfMessage: The queue of messages that have been sent by the sender but not yet received by the receiver.

State Transitions Most state transition functions are requested by the user, but some appear to occur spontaneously to the user. These are used to model the 'delay' associated with sending the message.

InitializeService: initialises **state** to **Idle** and **Sent**, **Received** and **Buffer** to the empty queue.

UserRequest: applicable only when **State** is **Idle** and sets it to **Arbitrating**.

AllocateDone: A spontaneous event, applicable only when **State** is **Arbitrating** and sets it to **ReadyRoSend**.

UserSend: applicable only when **State** is **ReadyToSend** and sets it to **Sending**. Adds a message to **sent** and **Buffer**.

DelayRecognize: A spontaneous event, applicable only when **State** is **ReadyToSend**. It models the delay involved in the receiver recognizing its address.

AddressRecognize: applicable only when **State** is **Sending** and sets it to **ReadyToReceive**.

DelayReceive: A spontaneous event, applicable only when **State** is **ReadyToReceive**. It models the delay involved in receiving the message.

UserReceive: applicable only when **State** is **ReadyToReceive** and sets it to **TransactionOver**. Adds the message at the head of **Buffer** to **Received**

Failed: applicable only when **State** is **ReadyToReceive** or **Sending**. and sets it to **TransactionOver** in each case.

UserRelease: applicable only when **State** is **TransactionOver** and sets it to **Idle**. A message is removed from **Buffer**.

Service representation in AFFIRM: The following is a stylized representation for the service in AFFIRM. State transition functions that leave a state variable unchanged are not explicitly specified. The auxiliary types **message** and **ControlState** are also listed here.

Petrinet machine **MessageBus;**

declare b:Messagebus;

declare m: Message;

constructors

InitializeService, UserRequest(b), AllocateDone(b), UserSend(b,m),
AddressRecognize(b), UserReceive(b), DelayRecognize(b), UserRelease(b),
DelayReceive(b), Failed(b);

selector State(b) : ControlState;

Selectors Buffer(b), Sent(b), Received(b) : QueueOfMessage;

axioms {InitializeService}

State(InitializeService) == Idle, ... (a)
Buffer(InitializeService) == NewQueueOfMessage, ... (b)
Sent(InitializeService) == NewQueueOfMessage, ... (c)
Received(InitializeService) == NewQueueOfMessage; ... (d)

axioms {UserRequest}

State(UserRequest(b)) == if State(b)=Idle
then Arbitrating
else State(b); ... (e)

axioms {AllocateDone}

State(AllocateDone(b)) == if State(b)=Arbitrating
then ReadyToSend
else State(b); ... (f)

axioms {UserSend}

State(UserSend(b,m)) == if State(b)=ReadyToSend
then Sending
else State(b), ... (g)
Buffer(UserSend(b,m)) == if State(b)=ReadyToSend
then Buffer(b) Add m
else Buffer(b), ... (h)
Sent(UserSend(b,m)) == if State(b)=ReadyToSend
then Sent(b) Add m
else Sent(b); ... (i)

axioms {AddressRecognize}

State(AddressRecognize(b)) == if State(b)=Sending

```

                                then ReadyToReceive
                                else State(b);                                ...(j)

axioms {UserReceive}
    State(UserReceive(b)) == if State(b)=ReadyToReceive
                                then TransactionOver
                                else State(b),                                ...(k)
    Received(UserReceive(b)) == if State(b)=ReadyToReceive
                                then Received(b) Add Front(Buffer(b))
                                else Received(b);                                ...(l)

axioms {DelayRecognize}
    State(DelayRecognize(b)) == if State(b)=Sending
                                then Sending
                                else State(b);                                ...(m)

axioms {DelayReceive}
    State(DelayReceive(b)) == if State(b)=ReadyToReceive
                                then ReadyToReceive
                                else State(b);                                ...(n)

axioms {Failed}
    Sent(Failed(b)) == if State(b)=Sending or ReadyToReceive
                                then Delete(Sent(b))
                                else Sent(b),                                ...(o)
    State(Failed(b)) == if State(b)=Sending or ReadyToReceive
                                then TransactionOver
                                else State(b);                                ...(p)

axioms {UserRelease}
    State(UserRelease(b)) == if State(b)=TransactionOver
                                then Idle
                                else State(b),                                ...(q)
    Buffer(UserRelease(b)) == if State(b)=TransactionOver
                                then Remove(Buffer(b))
                                else Buffer(b);                                ...(r)

no change State, Sent, Received, Buffer;

end {Message Bus};

type ControlState;
interfaces Idle, Arbitrating, ReadyToSend, Sending, ReadyToReceive,
    TransactionOver:ControlState;
distinct Idle, Arbitrating, ReadyToSend, Sending, ReadyToReceive,
    TransactionOver;
end {ControlState};

```

Properties of a Specification The next step in this method for Protocol Specification and Verification is to validate that the service specification at least partially meets the user requirements. This helps to increase our confidence in the reasonability of the specified machine.

For the Message Bus specification, our requirement for the **Safety** property is that the **messages received are equal to those sent, except for those in transit**. This can be formally stated in AFFIRM as theorem MessageBusService, all $b(\text{Sent}(b)=\text{Received}(b)\text{join Transit}(b))$;
define $\text{Transit}(b) == \text{if State}(b)=\text{TransactionOver}$

$\text{then NewQueueOfMessage}$
 $\text{else Buffer}(b)$;

This theorem can be proved by the method of Structural Induction as given in [Stan77]. The proof without an interactive theorem-prover like that of AFFIRM is rather laborious, and is given in [Garg83a].

Liveness properties can be established by stating and proving theorems about the **progress of the service**. We know that the allowed progression of states involves three distinct cycles, such that

(i) in the first cycle, a single message is transferred

(ii) in the second and third cycles, no message can be transferred i.e. either the six operations, **UserRequest, AllocateDone, UserSend, AddressRecognize, UserReceive** and **UserRelease**, in that order, add the correct message to those received, or
the six operations, **UserRequest, AllocateDone, UserSend, DelayRecognize, Failed** and **UserRelease**, in that order and/or
the eight operations, **UserRequest, AllocateDone, UserSend, DelayRecognize, AddressRecognize, DelayReceive, Failed** and **UserRelease**, in that order do not add any message.

These can be stated formally as two theorems as follows -

theorem ServiceProgress,

all $b_1, b_2, m(b_2=\text{UserRelease}(\text{UserReceive}(\text{AddressRecognize}(\text{UserSend}$
 $\text{(AllocateDone}(\text{UserRequest}(b_1)), m)))$
 $\text{and State}(b_1)=\text{Idle}$
imp $\text{State}(b_2)=\text{Idle}$
 $\text{and Sent}(b_2)=\text{Sent}(b_1)\text{Add } m$
 $\text{and Received}(b_2)=\text{Received}(b_1)\text{Add } m)$;

that the transaction is over

- If a NAK1 or NAK2 or NAK3 is received by the sender, it retransmits upto 'n' times. If all retries fail, the sender 'gives up' and releases the bus, indicating end of a (unsuccessful) transaction

* The packet-id SPI is used to check if the message received is a duplicate. This is as done as follows -

- The receiver maintains a packet-id of its own (RPI) whose value is set to zero on every Bus Release signal. On reception of a correct message, the receiver checks the received SPI with its RPI
- If the value of RPI is zero, the message is considered new, is ACKed and delivered to the user. The value of RPI is then changed to that of SPI
- If RPI is not zero, but equal to SPI, the message is a duplicate. It is ACKed but not delivered to the user
- If RPI is neither equal to SPI nor zero, the message is considered new (the Bus Release signal is assumed lost in this case), the message is ACKed and delivered and RPI made equal to SPI.

In the next section we shall see how the above characteristics of the protocol can be represented effectively by the use of Timed Petrinet models.

4.4.2 The TPN model for the INPAR protocol

Figure 4.2 shows the complete TPN for the INPAR protocol. The places User, SenderToReceiver and ReceiverToSender have time delays Z_1 , Z_2 and Z_3 associated with them, representing the fact that a time delay is encountered by the protocol when it is in either of these places. These represent the time taken by the User to prepare a packet, to send the packet on the bus and to receive the NAKs/Ack on the bus, respectively. All other places are assumed to be instantaneous, ie., the markers in them become available as soon as they arrive at these places.

Some places are not labelled at all. These are dummy places needed only because each transition should be followed and preceded by places. They do not represent any datastructure of the protocol. They will be useful in defining the control state of the protocol.

theorem NoMessage,

```
all b1,b2,m(      b2=UserRelease(Failed(DelayRecognize(UserSend
                  (AllocateDone(UserRequest(b1)),m)) or
                  b2=UserRelease(Failed(DelayReceive(AddressRecognize
                  (DelayRecognize(UserSend(AllocateDone
                  (UserRequest(b1)),m))))))
and State(b1)=Idle
imp      Sent(b2)=Sent(b1)
and Received(b2)=Received(b1));
```

Both the above theorems can be proved by Structural Induction. We show next that the above mentioned services are provided by the INPAR protocol.

4.4 THE INPAR PROTOCOL

4.4.1 Informal description

The INPAR protocol provides a 'Reliable Datagram' type of service on the Message Bus, ie., it guarantees delivery of the correct message, or the user is notified of failure, and duplicates are discarded.

The protocol can be described informally as follows -

- * A packet-id (SPI) and a CRC-word is attached to each message sent
- * Since the medium is unreliable, the message may be lost or corrupted and not reach the receiver. Moreover, if there are no available buffers at the receiver, a correct message may still not be received at the receiver
- If the message is lost, no ACK is received at the sender, but this also amounts to receiving a NAK (called NAK3) from the receiver. Thus there is no 'timeout' in the protocol, as the sender transmits immediately after the NAK/ACK is received.
- If the message is received but is found corrupted (by a CRC check) the receiver sends a NAK, (NAK1)
- If no buffers are available with the receiver, it sends a NAK (NAK2)
- If all is well and a correct message is delivered to the receiver, an ACK is sent to the sender
- If an ACK is received by the sender, it releases the Bus, indicating

The tokens shown with some places describe their initial conditions (values) in each cycle of the protocol operation. (The P in the SPI place represents P markers.)

The inhibitor arc from the place Retry to the transition GiveUp shows that the GiveUp transition will fire only when there are no tokens left in Retry. This models the fact that if after N retries, the message is still not delivered correctly, the transaction is given up. Similarly, the inhibitor arc from RPI to Deliver models the fact that the transition Deliver fires only when RPI has zero tokens (a value of zero). Inhibitor arcs in Petrinets are used for 'Zero testing' [Peters81]. They enhance the modelling power of the Petrinet. They shall not, however, be used for performance analysis.

The unlabelled place and transition on the left of Retry represents the fact that the ReleaseBus activates this transition repeatedly to result in the removal of all remaining tokens from place Retry. This is also the case for the place RPI as shown. These have again been drawn only to model the conditions under which the protocol operates in order to transfer messages correctly.

Finally, the TPN as shown in Figure 4.2 depicts clearly one complete cycle of the protocol from the transition ProtocolSend to ReleaseBus. We will see that such a cyclic representation makes it easy to specify the performance of the protocol.

4.4.3 TPN Markings and control state

A significant property of a Petrinet (and hence of a TPN) is that it has state information built into the net, as represented by the positions of the 'tokens' in the places of the net, ie., by its Marking vector.

The number of distinct states possible in a net is usually very large. This number depends on the number of places in the net and also on the value of the forward and backward incidence functions α and β . If the number of places is p, then for $\alpha = \beta = 1$, the number of distinct states is 2^p . If α and β are greater than 1, then the number of distinct states increases considerably.

However, the state information is very useful in formally specifying the working of the protocol, because it acts as the **controlling variable** (called control state) on which depends the firing of the transitions of the Petrinet. It thus makes it easy and simple to state enabling conditions for transitions while writing the complete formal specification of the protocol. This fact was brought out when we compared the AFFIRM specifications of our protocol using State Transition Machines with Timed Petrinet models. The latter is more compact and easy to understand. It is also especially useful in specifying and proving formally that the protocol is deadlock-free and live.

Because of the large number of possible distinct states of the Petrinet, it is usually impractical to enumerate and formally specify all of them. We have found a simple but effective way out of this problem and that is to **use only those places of the net for specifying the state information, which actually affect the firing of the transitions of the net.**

For example, for the INPAR protocol TPN, the places Sent, SPI and Received need not be considered for defining the state of the protocol, as they are not enabling conditions for its transitions, but only certain data structures required for its formal specification. The place RPI affects the firing of transitions, but again need not be considered for defining the state. Wherever necessary, it can be explicitly and separately specified.

For the INPAR protocol then, we are left with the eight places, p1(User), p2(SenderToReceiver), p3(ReceiverToSender), p4(Receive-Buffer), p5(Pending) and p7 and p8(dummy places), which will collectively define the control state. The presence or absence of a token in these places will give the state of the protocol. Figure 4-2 shows the TPN corresponding to these places.

The control state variable (selector) to be used for the INPAR protocol 'type' in its formal specification will be called **Marking**. It can be conveniently specified as being of the type **Register** as shown in the Appendix. The different values of the type Register (States) are given by the Reading of this Register. The Reading(i1,i2,...) thus describes the state when tokens are present in places i1,i2,... of the TPN.

For example, the Initial State, with a token in place p1 (User) is represented by Reading(1). Table 4.1 gives a list of all (only 13) possible, distinct states of the INPAR protocol as defined in this manner.

The variable Marking (the present state) is modified (into a new state) by the firing of each of the protocol transitions. This fact is written in the form of axioms for the selector Marking.

4.4.4 Time elapsed and Control state

A little thought into the Marking variable of a TPN, shows that at any instant, it does not fully represent the 'state' of the net. For example, the Marking given by Reading(1), which is also the initial state, is the same at the instant when the protocol is first started (as also when the transition ReleaseBus fires) and just before ProtocolSend is enabled. But the 'state' is not the same since the time elapsed is different.

Thus the complete state of a Timed Petrinet can only be specified by specifying both Timeelapsed as well as Marking at that instant. Thus

$$\text{state} : [\text{Marking}(s), \text{Timeelapsed}(s)]$$

The value of the variable Timeelapsed is shown along with the Marking, in Table 4.1, to give the instantaneous state of the Timed Petrinet of the protocol. (N is the total number of retransmissions possible, n ranges from 0 to N and m ranges from 1 to N).

In the next section, we shall use the Timed Petrinet model to write down a formal specification of INPAR in AFFIRM, using Marking and Timeelapsed as the controlling variables.

Table 4.1 List of INPAR TPN states

TRANSITION	STATE	
	Marking	Timeelapsed
Initial State	Reading(1)	0
ProtocolSend	Reading (2,5,6)	$z1$
ReceiveRecognize	Reading(3,4,5,6)	$z1+n(z2+z3)+z2$
	Reading(3,4,5)	"
Deliver or LosePacket	Reading(4,5)	"
	Reading(4,5,6)	"
GiveUp	Reading(5,8)	$z1+(N+1)(z2+z3)$
ReceiveByte	Reading(5,8)	$z1+m(z2+z3)$
	Reading(5,7)	"
	Reading(5,6,7)	"
LoseByte	Reading(3,5,6,7)	$z1+n(z2+z3)$
	Reading(3,5,7)	"
Retransmit	Reading(2,5)	$z1+m(z2+z3)$
ReleaseBus	Reading(1)	$z1+(n+1)(z2+z3)$

4.5 INPAR PROTOCOL FORMAL SPECIFICATION

4.5.1 Data types used in the specification

Message: this type is a minimally defined data type as in the service specification

Byte: an enumerated data type with four distinct elements ie., ACK, NAK1, NAK2 and NAK3

Packet: a record with three components: a value of type message, a

value of type counter (as defined below) and a value of type CRCofMessage (as defined below)

Counter: a data type with operations to decrement and increment its count and to clear, set, initialize it etc.

CRCofMessage: a minimal data type with a single operation to compute CRC on the type message

SMedium: this is the sending channel of the protocol and is really of type QueueOfPacket, with additional operations to 'lose' packet and 'corrupt' packets

RMedium: this is the returning channel of the protocol and is of type QueueOfByte, with the additional operation to 'lose' a byte.

Register: this defines the marking vector and hence the control state of the protocol.

The above data types can be individually specified in the AFFIRM notation as given in the Protocol specification in the Appendix. The types message, byte, packet, counter, CRCof message, Queueofmessage and Queueof-packet can be written directly as in their STM specification, since they do not carry any timing information. In other words, we assume that passage through these places of the TPN is instantaneous. But the three types User, SMedium and RMedium are associated with time delays Z1, Z2 and Z3 respectively. This is specified by the selector **timein** in each.

State Variables (selectors) of the protocol

SenderToReceiver - SMedium: channel from sender to receiver

ReceiverToSender - RMedium: channel from receiver to sender

Pending - QueueOfPacket: the packet currently being transmitted if any. This is either empty or has only one packet

ReceiverBuffer - QueueOfPacket: the packet received but not yet delivered to the user. It is also either empty or has only one packet

Sent - QueueOfMessage: the queue of all messages sent but not necessarily acknowledged yet

Received - QueueOfMessage: the queue of all messages successfully received

RBA - Counter: count of number of input buffers available at the receiver. This count is decremented on every delivered message and is incremented by the user

SPI - Counter: packet-id of the currently sent packet, incremented on each successful transmission

RPI - Counter: packet-id maintained at the receiver

Retry - Counter: current count of the number of retries left at sender.

Timeelapsed - integer: this represents the total time that has elapsed since the current protocol cycle was started. This function is modified by every transition of the protocol, but strictly it is not a datastructure of the protocol. It will be used to specify the performance property of the protocol.

Marking - Register: this represents the control state of the protocol at any instant. It is tested by most transitions for enabling conditions and is also modified by them.

Constructor (Transition) Functions of the protocol

InitializeProtocol: set the counters and queues to their initial values.

ProtocolSend: Given a message m, try to send it as a packet

- (i) if no message is waiting to be ACKed (**Pending=NewqueueOfPacket**) then accept m (by appending it to **Sent**)
- (ii) construct a **Packet** (by adding **SPI** and **CRCofMessage** to the message and transmit it (by adding to **SenderToReceiver**)
- (iii) wait for an ACK (put **Packet** in **Pending**)
- (iv) set **Retry** to its initial value

ReceiveRecognize: receive a packet if one is available

- (i) remove and examine the first packet, if **SenderToReceiver** is not empty
- (ii) if no input buffers are available (**RBA=0**) send NAK1
- (iii) else if packet is corrupt, send NAK2
- (iv) else send an ACK

Deliver : Try to deliver a new message

- (i) if it is a new message, add it to the **Received** queue, set **RPI = SPI** and clear **ReceiverBuffer**
- (ii) if it is a duplicate, clear **ReceiverBuffer**

ReceiveByte: receive an ACK/NAK if it is there

- (i) If **ReceiverToSender** is not empty then remove the first byte and examine it
- (ii) if NAK and if **Retry** is not zero, decrement it

Retransmit: resend the message in **Pending** by adding it to **SenderTo-Receiver**

GiveUp: if **Retry** is zero, delete the last message added to **Sent**

LoseByte: lose a reply by removing front of **ReceiverToSender**

LosePacket: lose a packet by removing front of **SenderToReceiver**

ReleaseBus: indicate transaction is over by setting **Retry** and **RPI** to zero, incrementing **SPI** and emptying **Pending**.

As an example, a typical 'state' of the protocol may be

```
ReceiveByte(Deliver(ReceiveRecognize(ProtocolSend(Initialize
                                                    Protocol,m))));
```

which represents the sequence of transitions

```
InitializeProtocol; ProtocolSend(s,m); ReceiveRecognize(s); Deliver(s)
and ReceiveByte(s) in that order of firing.
```

(This state is represented by the Marking variable having a value **Reading(5,6,8)** and **Timeelapsed = TimeinUser + TimeinSenderToReceiver + TimeinReceiverToSender**).

4.5.2 AFFIRM representation of the protocol specification

The stylized version of the INPAR protocol in AFFIRM is given in the Appendix IIa. Also defined are the auxiliary types Message, Packet, RMedium, SMedium, Counter, CRCofMessage, Byte, QueueOfMessage, QueueOfPacket, Register and Byte.

The selector Timeelapsed(s) represents the time that has elapsed at the instant when the previous transition was fired and which resulted in the present state s of the system. This takes into account the fact that in a TPN, time is not continuous but measured in terms of the discrete time bases c defined for Timed Petrinets.

4.6 THE INPAR PROTOCOL VERIFICATION

Now we will attempt to verify that our lower level (protocol) TPN specification actually implements the higher level (Message Bus) one. To do this, we have to convert each of the higher level axioms into a theorem at the lower level and then prove these theorems using the lower level specification. This can be done in two steps. First, we define -

- (i) a **static** correspondence between each state of the higher level and the state(s) implementing it at the lower level, showing in fact that every higher level state is implemented, and
- (ii) a **dynamic** correspondence between the transitions of the two levels, showing that the sequence of states reachable in the two levels are the same.

The above definitions are achieved by giving a representation function **rep** from the state variables of the lower level to the state variables of higher level. These are shown informally in Figure 4.3 (pg. 87) and in a formal way in Appendix IIb.1. It can be seen that there are two sets of protocol operations -

- (i) the first set has six operations corresponding to each of the six user operations at the service level
- (ii) the second set of protocol operations (which has four operations in all) collectively accomplish the two spontaneous operations of the service level

(The service level state variables **Sent** and **Received** are implemented transparently, while **Buffer** is implemented as the text of the first packet in **Pending**.)

Appendix IIb.2 defines the protocol level state classes and the predicates in the protocol specification defining these states (in AFFIRM).

Next, we replace each occurrence of the service machine state in the axioms of the service specification by the **rep** of its corresponding protocol states and use other rewrite rules displayed in Appendix IIb.1 until the expression is reduced to terms involving only the protocol level selectors and constructors. The resulting theorems are shown in Appendix IIc.

It can be shown that the axioms defining the results of the **user** operations on state variables are the easiest to convert to theorems. This is because each service level operation is directly implemented by a single protocol operation, and the state variables also have a simple correspondence.

For the **spontaneous** service operations, there is no fixed correspondence of one protocol operation for each service operation. We have to show instead that **any** sequence of the four spontaneous protocol operations will have the specified effect. The theorems show that each such individual protocol operation has no effect on the state variables. So we can also conclude that any sequence of these operations will have no effect, which is what we want from each service operation.

All the theorems in Appendix IIc can be proved easily by using the axioms of the protocol specification and the correspondence of states between service and protocol. This shows that the protocol correctly implements the service.

The final set of theorems in Appendix IIc (13 to 48) covers the effect of the operations on the service level state. The first group of theorems (13 to 32) states that for the user operations, the correct next state is generated by the corresponding protocol operation for each of the six states of the system. The second group of theorems (32-48) states that for the spontaneous operations, either no state change or only the correct one, takes place due to any such individual operation. Hence their combined effect gives no change or only the correct change of state.

Thus as the protocol progresses from one operation to another, the right state changes (if any) occur as specified in the service. This in fact proves the **safety** of the protocol.

Service	Protocol
InitializeService	InitializeProtocol
UserSend	ProtocolSend
AddressRecognize	ReceiveRecognize
UserReceive	Deliver
UserRelease	ReleaseBus
Failing	GiveUp
Sent	Sent
Received	Received
Buffer	Text(Front(Pending)))
State Idle	S1
Arbitrating	S2
ReadyToSend	Reading(1)
Sending	Reading(2,5,6)
ReadyToReceive	Reading(3,4,5,6)
TransactionOver	Reading(5,8)
DelayRecognize	any possible sequence of the operations {LosePacket,ReceiveByte,Retransmit}
DelayReceive	any possible sequence of the operations {ReceiveByte,LoseByte,Retransmit ReceiveRecognize}

Figure 4.4 The correspondence between service and protocol state variables

4.7 DISCUSSION AND CONCLUSION

Timed Petrinets have been shown to successfully model the behaviour of a protocol in order to formally state its behaviour as required and prove its safety property. While many other researchers have modelled protocols using Finite State Machines, our work using Timed Petrinets brings to light two main points. These are -

- i) Timed Petrinets are better suited to model protocol behaviour than State Transition Machines for the following reasons:

- a) they allow representation of
- * the number of allowed retransmissions in a protocol by the use of multiple arcs from a transition to its output place
 - * the initial condition (state) by the presence or absence of tokens in the places
 - * the GiveUp condition by the use of inhibitor arcs
 - * the Timeout condition using the Timeelapsed variable, though we have not used this as the INPAR protocol does not have an explicit Timeout
- b) for the above reasons, the formal specification of the Protocol rendered much more compact and easy to follow.
- c) ease of specification of Liveness and Performance for protocol. We will show in the next chapter that the biggest advantage of using Timed Petrinet models is that it is possible to consider these properties using the variable Timeelapsed as defined in Section 4.4.
- ii) **State explosion need not be a limiting factor with Timed Petrinet models** It is well known that one of the biggest disadvantages of using Finite State Machines to model complex protocols, is that combinatorial state explosion occurs, making it difficult to formally specify each state in the model as a controlling variable. We have shown, however, that this need not always be true and that all the distinct possible states of the protocol do not have to be explicitly specified; only those that contribute directly to the firing of transitions need be enumerated. Thus places which only model data structures (as opposed to control structures) can be disregarded when specifying state information, reducing the total number of states. We will see in the next chapter that further reductions are possible in the model when its performance characteristics are to be specified.

A major drawback that still exists with the above Timed Petrinet model is that it is not possible to identify the type and order of messages received or sent. For example, in the INPAR protocol considered above, we are unable to identify whether an ACK or NAK was received. Place coloured and Predicate/Transition nets [Diaz82], in which tokens have identity and relationship, have been proposed to overcome these limitations of petrinets, but they still have no time

factor associated with the places or transitions. We feel that if Predicate/Transition nets can be 'timed' and a theoretical analysis made possible for them, they would provide an ideal tool for complete protocol specification.

This chapter has provided a detailed specification of the functional verification of a protocol using Timed Petrinet models and must be considered as having laid the basis for the specification of a Protocol's Liveness and Performance properties, which will be the subject of the next chapter.

CHAPTER 5

PROTOCOL LIVENESS AND PERFORMANCE

5.1 INTRODUCTION

This chapter is concerned with formally specifying and verifying the liveness and performance of a communication protocol using its Timed Petrinet model. It can be viewed as an extension of the last chapter in which we presented a means of specifying and validating the 'functional' behaviour of a protocol using Timed Petrinets. In other words, whereas the last chapter was concerned with **design validation**, this chapter deals with **Performance validation** of protocols using TPNs.

Liveness properties for protocols have been considered difficult to state and prove using State Transition Machines [Sunshi82b]. This is because it is not possible to specify and hence test state machine specifications for progress properties. Temporal Logic [Schwar81] has been thought to be better in this respect since progress and eventuality can be specified using its constructs. We will show in this chapter that the concept of time in Petrinet models, as we have used it, can help to make it possible to specify progress and hence eventuality in protocol operations.

It has now been established [Sunshi82b] that without validation of its performance properties, a protocol is only partially correct, even when it has been proved to be safe and live. According to Sunshine, 'eventually is better than never, but it may still be too long'.

Performance validation of a protocol is concerned with stating and proving algebraically that the protocol works in such a manner as to accomplish its job within a time acceptable to its higher level user service. Indeed, this should be the case for all possible conditions of the message transfer medium and for all sequences of transitions needed to do the message transfer correctly. Thus the time taken by the protocol for correct message transfer should be less than or equal to that allowed for the transfer by the user service. This is important in order to meet the real-time requirements of the system. It is also important that the time taken by the protocol be known, as

it controls the message buffer holding times and hence the buffer requirements in the system.

In distributed computer controlled real time systems, a certain 'information rate' is to be maintained in order for the system to be safely controlled [Prince81]. This information rate depends not only on the actual data transfer rate on the medium, but also on the time taken by the protocol overheads, ie., conversion of messages to packets, retransmissions and timeouts.

We present our methodology for performance specification of protocols and illustrate it using the INPAR protocol. Since INPAR is a relatively simple protocol, we also model and analyze a typical real-life protocol, the ECMA standard, using the proposed methodology. This protocol exhibits typical properties like conflict, parallelism and synchronization, and the effect of these on the calculation of performance is also presented.

In Section 2 we state and prove liveness for the INPAR protocol, using its Timed Petrinet specification of Chapter 4. In Section 3 we show how the TPN model can be analyzed to give its performance in terms of the cycle time needed to complete its various operations and how this can be formally specified. Our methodology for performance validation is given in Section 4. In Section 5 it is illustrated using the ECMA protocol. We discuss our methodology in terms of shortcomings and possible extensions in Section 6, as also some related work. Conclusions are given in Section 7.

5.2 PERFORMANCE ANALYSIS OF INPAR PROTOCOL

5.2.1 Reduction of the Timed Petrinet

Once the protocol model has been proved to be safe, we can reduce its Timed Petrinet for performance analysis. Thus, in Figure 4.2 we use only those places and transitions which affect its performance characteristics. The resultant net is shown in figure 5.1 where the places and transitions have been labelled in a different way. The places Sent, Received, RPI, SPI, ReceiverBuffer and Pending and the transition Deliver are not drawn in Figure 5.1 as they do not affect the firing of the transitions and are in fact needed only to show that the protocol functions 'correctly'. Also, in this figure we have

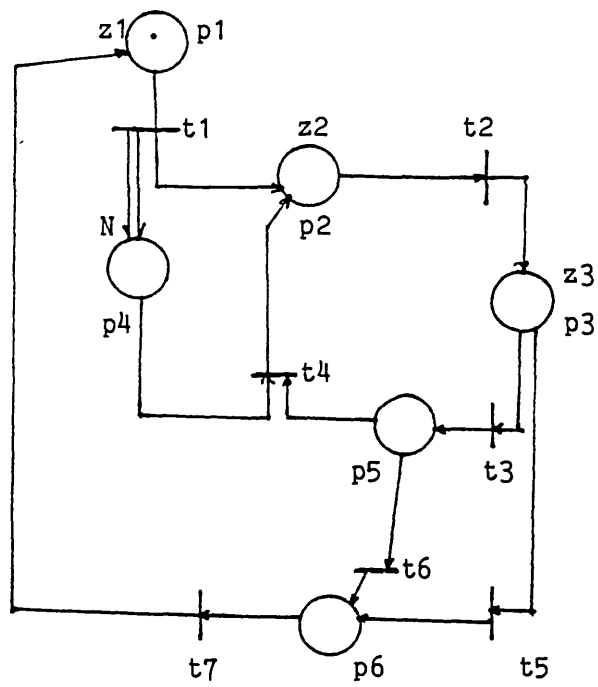


Figure 5.1 INPAR protocol Timed Petrinet for performance

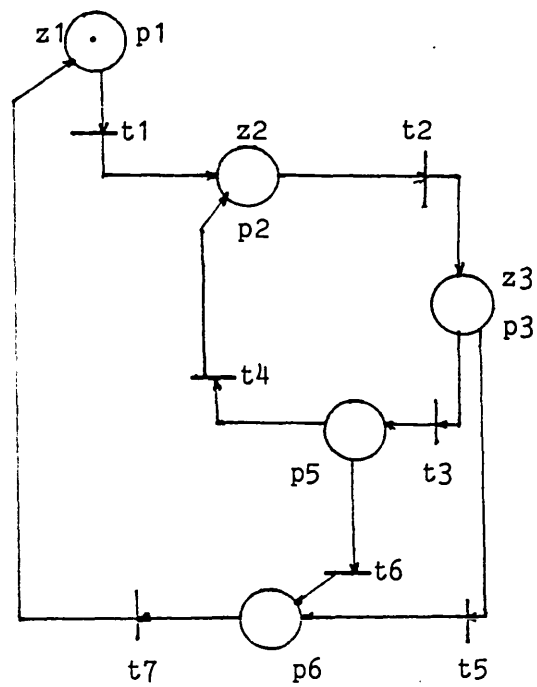


Figure 5.2 INPAR protocol subnet for J

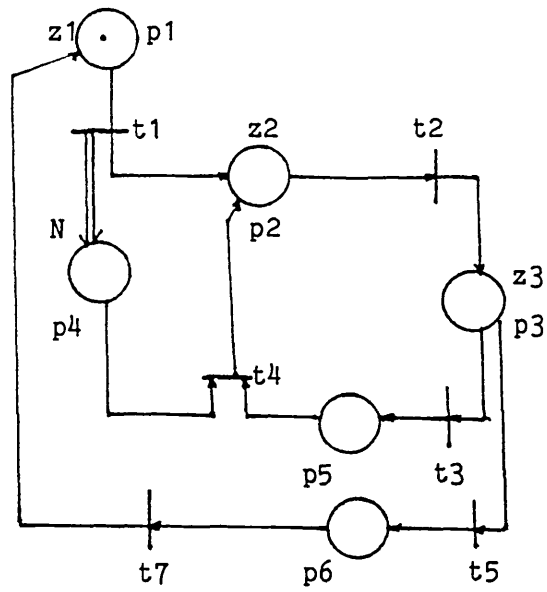


Figure 5.3a INPAR protocol subnet for sequence I1

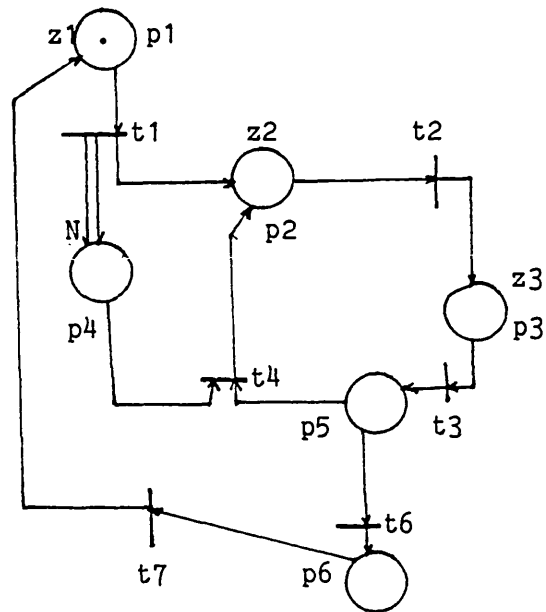


Figure 5.3b INPAR protocol subnet for sequence I2

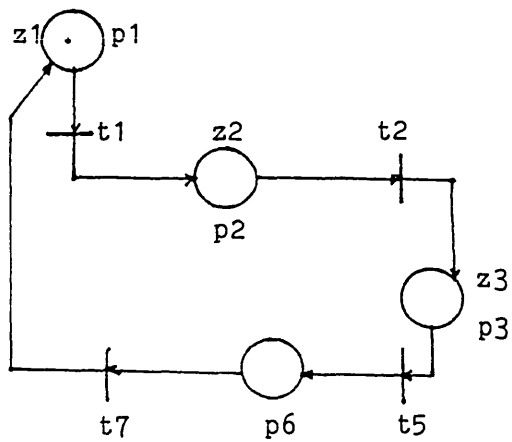


Figure 5.3c INPAR protocol subnet for sequence I3

combined transitions t_2 and t_8 , and t_2 and t_5 into t_2 and t_3 respectively, since each pair has the same effect.

The rules that we have used for reduction of the Timed Petrinet, are

- a) The places representing only data structures of the protocol can be discarded.
- b) Places which do not carry any timing information can also be discarded, if possible, by combining their input and output transitions.

These reduction rules are in addition to those given in [Berthe80].

Performance analysis for the reduced Timed Petrinet model can now be done by finding the maximum computation rate of the net, as given in [Sifaki77]. For the TPN of Figure 5.1 we have,

$$Q^t(\tau_0) = [1 \ 0 \ 0 \ 0 \ 0 \ 0], \quad Z = \text{diag} [z_1 \ z_2 \ z_3 \ 0 \ 0 \ 0] \text{ and}$$

$$C = \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & -1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & -1 & 0 & 0 \\ N & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & -1 \end{bmatrix}$$

The equation $CI = 0$ has three linearly independent solutions given by the vectors

$$\begin{aligned} I1 &= [1 \ N+1 \ N \ N \ 1 \ 0 \ 1] \\ I2 &= [1 \ N+1 \ N+1 \ N \ 0 \ 1 \ 1] \\ I3 &= [1 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1] \end{aligned}$$

$J^t C = 0$ has only one solution given by $J^t = [1 \ 1 \ 1 \ 0 \ 1 \ 1]$

$$J^t Q(\tau_0) \geq J^t Z C^+ I1 \Rightarrow 1 \geq [z_1 + (N+1)(z_2+z_3)].i1 \dots(1)$$

$$J^t Q(\tau_0) \geq J^t Z C^+ I2 \Rightarrow 1 \geq [z_1 + (N+1)(z_2+z_3)].i1 \dots(2)$$

$$J^t Q(\tau_0) \geq J^t Z C^+ I3 \Rightarrow 1 \geq [z_1+z_2+z_3].i1 \dots(3)$$

It is to be noted that the vectors $I1$, $I2$ and $I3$ represent the three possible sequences of operations of the protocol and correspond to the operations of transferring a message with N retransmissions [$I1$], without retransmissions [$I3$] and no possible transfer (transaction given up) [$I2$]. The subnets of the protocol TPN corresponding to

these vectors are given in Figures 5.3a-c respectively. Similarly, the subnet corresponding to the J vector is given in Figure 5.2.

The inequalities (1) - (3) give the maximum computation rates for the transition t_1 ie., ProtocolSend in the three cases. Since functioning at natural rate is possible, we can rewrite these as

$$i_1 = 1 / [z_1 + (N+1)(z_2+z_3)] \text{ for } I_1 \text{ and } I_2 \text{ and}$$

$$i_1 = 1 / [z_1+z_2+z_3] \text{ for } I_3$$

Alternatively, it can be said that the time elapsed between two ProtocolSend operations or the time taken to complete one cycle of the protocol in each case, whether a message is transferred or not is given by

$$T_1 = z_1 + (N+1)(z_2+z_3) \text{ for } N \text{ retransmissions and}$$

$$T_2 = z_1+z_2+z_3 \text{ for no retransmissions.}$$

The above equations can also be written down by inspection from Fig. 5.2 by simply adding the time delays of the places involved in each of the sequences I of figure 5.3. These equations can be used to specify formally the performance properties of the protocol.

5.2.2 Formal specification of protocol performance

Informally, it can be said that the time required for any 'send' operation is dependent only on the message sending time (effectively $\text{Timein}(\text{SenderToReceiver})$), the response receiving time ($\text{Timein}(\text{ReceiverToSender})$) and the number of retransmissions necessary.

The above property for the performance of the INPAR protocol can be stated in the following AFFIRM theorems, using the variables Timeelapsed and the constants Timein for the User and the sending and receiving mediums in the protocol specification given in the Appendix. TPN theory will enable us to write down these theorems.

The time taken for any protocol operation to be performed is dependent on the sequence of transitions to be fired in order to complete that operation. We have identified two cases corresponding to the I vectors I_1 (or I_2) and I_3 as given in the Figures 5.3 a-c.

In Figure 5.3c, where a message can be sent correctly without the need for any retransmissions, we have the theorem,

theoremBesttime,

```
for all s,m ( Timeelapsed(ProtocolSend(ReleaseBus(ReceiveByte(Deliver(
    ReceiveRecognize(ProtocolSend(s,m)))))),m)
    = Timeelapsed(ProtocolSend(s,m)) + Timein(user) +
    Timein(SenderToReceiver)+Timein(ReceiverToSender));
```

which states that the time that elapses between two consecutive ProtocolSends, when no retransmissions are necessary, is simply equal to the sum $z_1+z_2+z_3$.

For Figures 5.3a&b where the message may (or may not) be sent successfully after n retransmissions, such that $n \leq N$, we have the theorem,

theoremretriedtime,

```
for all s,m ( Timeelapsed(ProtocolSend(ReleaseBus(ReceiveByte(Deliver
    (ReceiveRecognize(Retransmit(ReceiveByte(Receive
    Recognize(ProtocolSend(s,m)))))*))),m)
    = Timeelapsed(ProtocolSend(s,m)+Timein(User) + (n+1)
    (Timein(SenderToReceiver)+Timein(ReceiverToSender)) );
```

{* represents n retransmissions at most}

which simply states that the time that elapses between two 'sends' when retransmissions are necessary, is equal to the sum $(z_1+(n+1)(z_2+z_3))$.

In the second theorem, if we replace the sequence ReceiveByte(Deliver) by GiveUp, it would be applicable to the case where no message can be transferred and the transaction is given up. We can also use any possible combination of the transitions LosePacket, LoseByte, ReceiveByte and ReceiveRecognize to consider all possible conditions of the mediums.

Both the theorems stated above can be proved by Structural Induction, by using the axioms of the protocol specification as rewrite rules.

The statements for the performance properties for the INPAR protocol are relatively simple. This may not be the case for most other protocols. We intend next to show how our method of protocol performance specification can be easily extended to well known or 'standard' communication protocols and we illustrate this using the ECMA standard protocol [ECMA80].

5.3 PROVING LIVENESS FOR INPAR PROTOCOL

Liveness (something eventually happens) is concerned with stating and proving that the protocol

- (i) normally does effective work, ie., transfers a correct message - this is related to the PROGRESS property
- (ii) returns to its initial state after each cycle, even if no message could be transferred in that cycle - this is related to the DEADLOCK FREENESS property.

In other words, we prove that for all possible conditions of operation of the protocol (represented by the medium at any time), it completes a cycle and returns to its initial state, whatever be the intervening transitions.

In order to formally state the above liveness properties in a language like AFFIRM, we have to write theorems about the protocol type, in which state invariance is maintained for all possible sequences of transitions.

Timed Petrinet theory provides the means for defining all possible sequences of transitions which, starting from an initial state, result in the same state being reached again. Such transitions are called 'neutral sequences' and can be identified by finding the solution of the equation $CI = 0$, where C is the incidence matrix of the net.

As discussed earlier, these are given by the vectors I_1, I_2 and I_3 and represented in Figures 5.3 a)-c). These can be enumerated as the sequences -

$I_1 = (t_1; (t_2; t_3; t_4)^*; t_5; t_7)$

$I_2 = (t_1; (t_2; t_3; t_4)^*; t_6; t_7)$

$I_3 = (t_1; t_2; t_5; t_7)$

for which we can write

$I_1 = (\text{ProtocolSend}; ((\text{ReceiveRecognize}, \text{LosePacket}); (\text{ReceiveByte}, \text{LoseByte}); \text{Retransmit})^*; \text{ReceiveRecognize}; (\text{Deliver} // \text{ReceiveByte}); \text{ReleaseBus})$

$I_2 = (\text{ProtocolSend}; ((\text{ReceiveRecognize}, \text{LosePacket}); (\text{ReceiveByte}, \text{LoseByte}); \text{Retransmit})^*; (\text{ReceiveRecognize}, \text{LosePacket}); (\text{ReceiveByte}, \text{LoseByte}); \text{Giveup}; \text{ReleaseBus})$

$I_3 = (\text{ProtocolSend}; \text{ReceiveRecognize}; (\text{Deliver} // \text{ReceiveByte}); \text{ReleaseBus})$

(where $t;t'$ means transition t followed by t' , $t//t'$ means t and t' in any order and t,t' means transition t or t') [Berthe83].

Using the above, we can write down the following theorems about the liveness property of the INPAR protocol -

TheoremProtocolProgress {for I1 and I3}

```
all s1,s2,m (s2 = ReleaseBus(ReceiveByte(Deliver(ReceiveRecognize(
    Retransmit(LoseByte(LosePacket(
        ProtocolSend(s1,m)))))*)))
    and Marking(s1)=Reading(1)
    imp Marking(s2)=Reading(1)
    and Sent(s2)=Sent(s1)Add m
    and Received(s2)=Received(s1)Add m )
```

which simply states formally the fact that the above operations in the right order will deliver a correct message and return to the initial state. The transitions LoseByte and/or LosePacket can be exchanged with ReceiveByte and/or ReceiveRecognize respectively to take into account all possible sequences contained in I1.

TheoremMessagefailure {for I2}

```
all s1,s2,m (s2 = ReleaseBus(GiveUp(ReceiveByte(Retransmit(LoseByte
    (RecevieRecognize(ProtocolSend(s1,m)))))*)))
    and Marking(s1)=Reading(1)
    imp Marking(s2)=Reading(1) )
```

which states that the above sequence of operations will only complete and return to the initial state but no message will be transferred. As before, any possible combination of the transitions LoseByte, LosePacket, Receive- Recognize, ReceiveByte can be used in the theorem.

Both the above theorems can be proved by Structural Induction.

5.4 AN OVERVIEW OF OUR METHOD FOR PERFORMANCE SPECIFICATION

Our method of specifying communication protocol performance in terms of the properties of its abstract data type representation, can be summarised as follows:

- a) Draw a detailed TPN diagram of the functioning of the protocol. Use 'places' to represent its states, medium, buffers, and other data structures, and 'transitions' to represent events or operations.
- b) Reduce the TPN as much as possible, by disregarding, for example, the places and transitions not concerned with performance issues.
- c) Analyze the TPN model of the protocol and obtain the various possible distinct transition sequences of operation of the protocol. This is done by solving the equation $CI = 0$ for the net, where C is its incidence matrix. Find also all the independent solutions of the equation $J^t C = 0$.
- d) Draw the relative subnets of the TPN, one for each I vector and J vector.
- e) Write a formal specification of the protocol TPN as an abstract data type in a suitable specification language. Include a variable (Timein) to specify the time delay associated to each place. Also include a variable (selector of the protocol type) to specify the time elapsed (Timeelapsed) after each operation (transition) of the protocol. Write the axioms for this selector showing when and how it is modified (if at all) by every transition.
- f) For each sequence of transitions I , write down the time taken to complete the sequence by inspection from each J vector subnet. This can be done by simply adding the time delays z_i of the places p_i of the subnet for J , which are involved in completing this sequence. The time taken for each sequence I is the largest of the times given by the J subnets. If there is only one possible J vector, then the time taken for each I sequence is equal to the time found from that J subnet.
- g) Rewrite the above formally as theorems in the specification language for the selector variable Timeelapsed using the values of the variables 'Timein' for the places. One theorem can be written for each I vector.

5.5 PERFORMANCE SPECIFICATION OF ECMA PROTOCOL

To illustrate that the methodology described in the last section is applicable to a relatively complex communication protocol, we have chosen the European Computer Manufacturer's Association (ECMA) [ECMA80] transport protocol. This is an end-to-end protocol at the Transport layer level of the ISO Reference Model [ISO81].

Four classes of protocols are defined allowing network layer errors of increasing severity. Each class has three phases of operation: establishment of a transport connection (^{dis}connection), data transfer, and termination of a transport connection (disconnection).

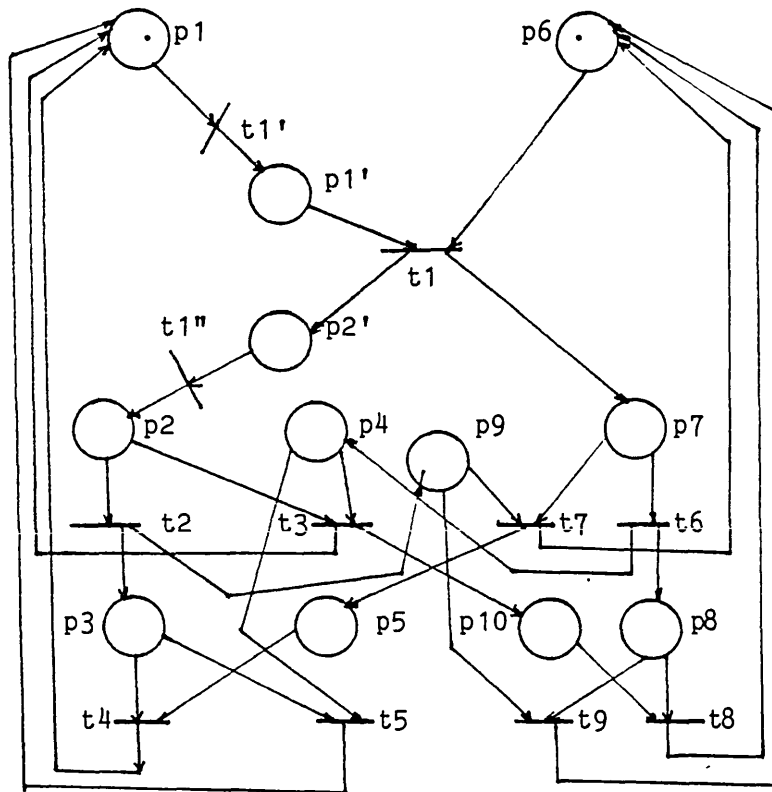
We shall concentrate only on the connection and disconnection phases for classes 1 and 2 of this protocol, since this will suffice for our purpose of illustration. For these classes, there are no mechanisms for recovery from errors, and when an error occurs, each end returns to the initial disconnected state where all but connect request messages are ignored.

5.5.1 Connection and Disconnection Phase and its TPN model

The transport layer provides the means for users to communicate by establishing a transport connection between two transport entities. Transport entities are managed locally and a connection can only be set up between entities in the disconnected state. Hence, simultaneous attempts for connection result in the creation of two transport connections, and there is always an initiating entity and an accepting entity in a transport connection.

Transport entities are initially disconnected. Their behaviour from this state is described by the net shown in Figure 5.4. In the initial state (p1), the initiating entity sends (t1') a connect request (p1'). The accepting entity (p6) accepts the request (t1) by sending the connect confirm (p2') and the connection is established (t1''). Both entities now pass to the data transfer (p2 and p7) phase. Once in this phase, each entity may end a transport connection by sending (t2 and t6) a disconnect request (p4 and p9) and pass to the waiting disconnect confirm state (p3 and p8). In this state, data messages are discarded.

On receipt of a disconnect request, the remote entity returns to the initial state either by sending (t3 and t6) a disconnect confirm (p5 and p10) if it was in the data transfer state, or not (t5 and t9) if a disconnect collision has occurred and it was in the waiting for disconnect confirm state. The role of each entity can eventually be switched. In every state, an error signal causes both entities to return to the initial state, where all but connect request messages are ignored (not shown).



<u>places</u>		<u>transitions</u>	
p1 and p6:	idle	t1'	:connect request
p1'	:waiting connect	t1''	:connect establish
	confirm	t1	:connect accept
p2'	:connection confirm	t2 and t6:	disconnect request
p2 and p7:	data transfer	t3 and t7:	disconnect accept
p3 and p8:	waiting disconnect	t4 and t8:	disconnect end
	confirm	t5 and t9:	disconnect accept
p4 and p9:	request disconnect		
p5 and p10:	confirm disconnect		

Figure 5.4 Timed Petrinet for ECMA connection-disconnection phase

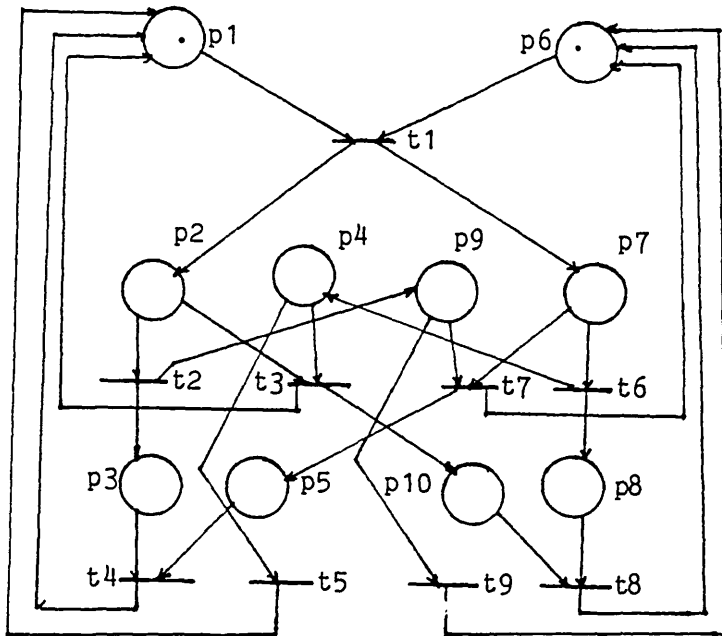


Figure 5.5 The reduced model

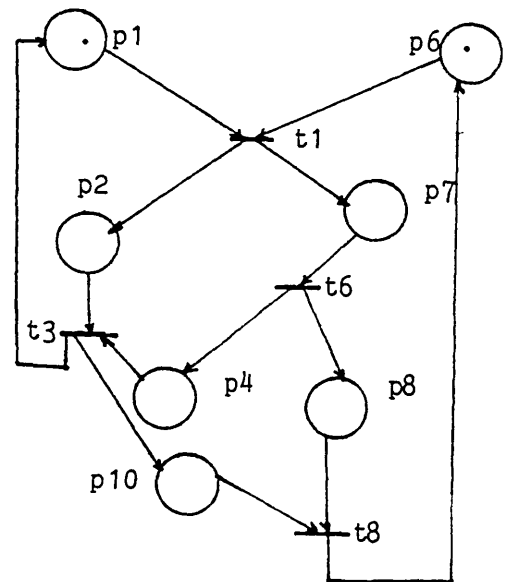


Figure 5.6a Subnet for I1

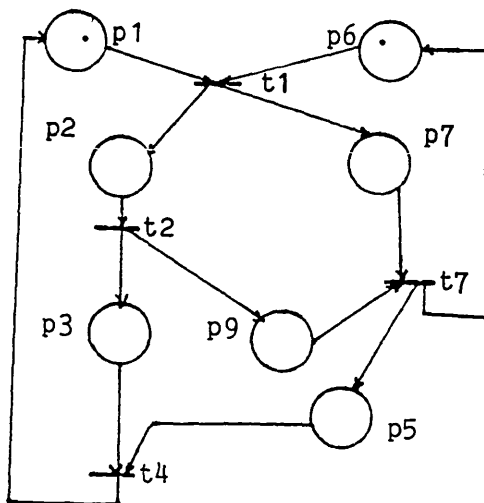


Figure 5.6b Subnet for I2

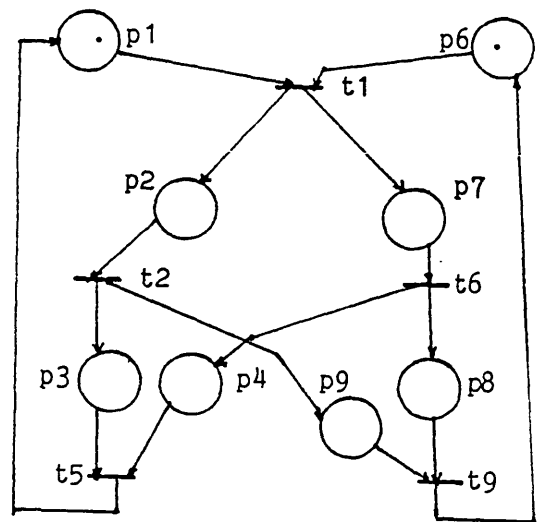


Figure 5.6c Subnet for I3

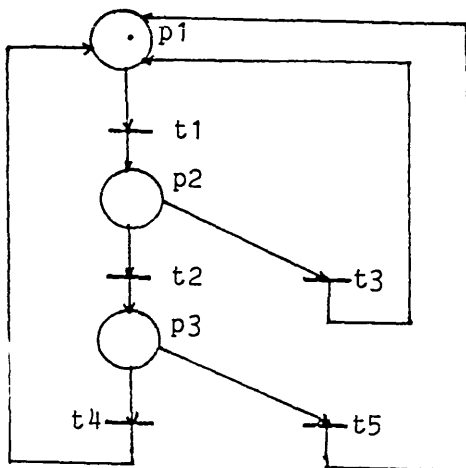


Figure 5.7a Subnet for J1

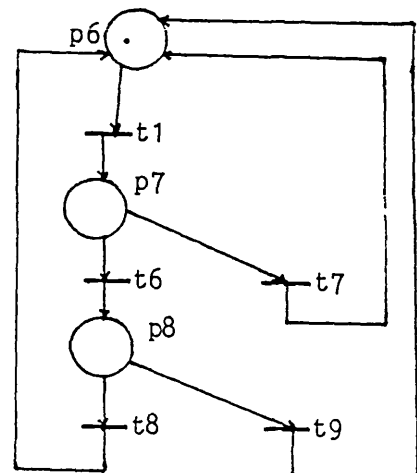


Figure 5.7b Subnet for J2

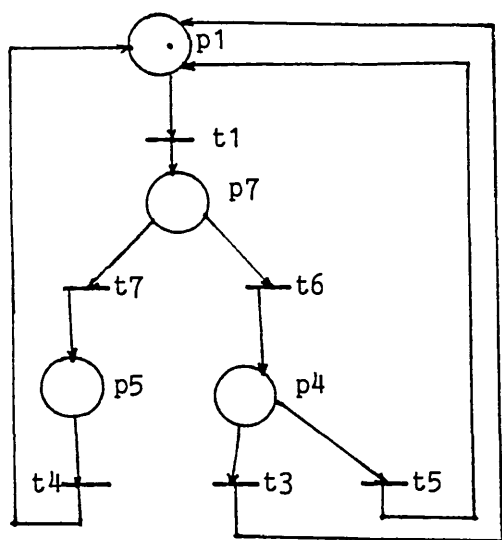


Figure 5.7c Subnet for J3

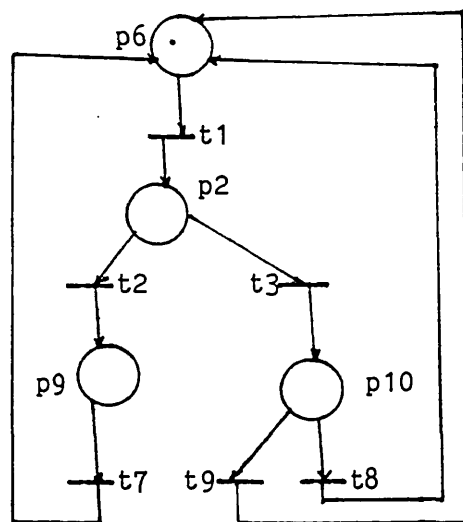


Figure 5.7d Subnet for J4

A time delay z_i is assumed associated with each place p_i (except p_3 and p_8). Because of the serial nature of the transitions $t1', t1''$ and $t1$ and the places $p1' & p1$ and $p2' & p2$ we can simplify the figure to that of Figure 5.5 and assume that the time delays associated with $p1'$ and $p2'$ have been included in z_1 and z_2 respectively. This in fact means that the connection phase is now represented only by the transition $t1$. Figure 5.5 will henceforth be used to model our protocol.

5.5.2 Analysis of the TPN model

We have, for Figure 5.5, $Z = \text{diag} [z_1 \ z_2 \ \dots \ z_{10}]$

$$Q^t(\tau_0) = [1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]$$

$$C = \begin{bmatrix} -1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & -1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 \end{bmatrix}$$

CI = 0 has three solutions given by

$$I1 = [1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 1 \ 0]$$

$$I_2 = [1 \ 1 \ 0 \ 1 \ 0 \ 0 \ 1 \ 0 \ 0]$$

$$I_3 = [1 \ 1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 0 \ 1]$$

$J^t C = 0$ has four solutions given by

$$J^{t_1} = [1 \ 1 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$

$$J^{t_2} = [0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 1 \ 1 \ 0]$$

$$J^{t_3} = [1 \ 0 \ 0 \ 1 \ 1 \ 0 \ 1 \ 0 \ 0]$$

$$J^{t_4} = [0 \ 1 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 1]$$

The various subnets representing these vectors are given in Figs. 5.6a-c and 5.7a-d respectively. For each I vector, the maximum rate of computation can be written from the inequalities

$$J_i^t Q(\tau_i) \geq J_i^t Z C^+ I \quad \left| \begin{array}{l} 4 \\ i=1 \end{array} \right.$$

Thus we have, for the sequence of transitions given by I1,

$$ilmax. = \min. \text{ of } [1/(z_1+z_2) \quad 1/(z_6+z_7) \quad 1/(z_1+z_4+z_7) \quad 1/(z_2+z_6+z_{10})]$$

For the sequence given by I2 we have

$$ilmax. = \min. \text{ of } [1/(z_1+z_5+z_7) \quad 1/(z_2+z_6+z_9) \quad 1/(z_1+z_2) \quad 1/(z_6+z_7)]$$

and for the sequence given by I3 we have

$$ilmax. = \min. \text{ of } [1/(z_1+z_4+z_7) \quad 1/(z_2+z_6+z_9) \quad 1/(z_1+z_2) \quad 1/(z_6+z_7)]$$

Conversely, we can state that the minimum times taken for one cycle of each sequence to be completed are

$$Tmin. = \max. \text{ of } [(z_1+z_2) \quad (z_6+z_7) \quad (z_1+z_4+z_7) \quad (z_2+z_6+z_{10})] \text{ for } I_1 \quad (1)$$

$$Tmin. = \max. \text{ of } [(z_1+z_2) \quad (z_6+z_7) \quad (z_1+z_5+z_7) \quad (z_2+z_6+z_9)] \text{ for } I_2 \quad (2)$$

$$Tmin. = \max. \text{ of } [(z_1+z_2) \quad (z_6+z_7) \quad (z_1+z_4+z_7) \quad (z_2+z_6+z_9)] \text{ for } I_3 \quad (3)$$

Alternatively, we can get these values from inspection of the Figures 5.7a-d for each of the sequences I. We interpret these as follows-

For the sequence I1 (Fig 5.6a), which corresponds to the firing of the transitions (t1;t6;t3;t8), the time taken is the same as that taken for the sequence (t1;t3) - Figure 5.7a

sequence (t1;t6;t8) - Figure 5.7b
sequence (t1;t6;t3) - Figure 5.7c and
sequence (t1;t3;t8) - Figure 5.7d

These times can be obtained from each figure by simply adding up the time delays of the places involved in firing the constituent transitions of the sequence I1.

Thus from Figure 5.7a, the time is $(z1+z2)$
from Figure 5.7b, it is $(z6+z7)$
from Figure 5.7c, it is $(z1+z4+z7)$
and from Figure 5.7d, it is $(z2+z6+z10)$

The largest of the above gives (1). Again, we know that the transition t3 in sequence I1 fires only if $z2 \geq (z4+z7)$

Thus, (1) reduces to $T_{min.} = \max. [(z1+z2), (z2+z6+z10)] \dots(4)$

Similar arguments lead to the following equations for I2 and I3.

$$T_{min.} = \max. [(z6+z7), (z1+z5+z7)] \dots(5)$$

$$T_{min.} = \max. [(z1+z4+z7), (z2+z6+z9)] \dots(6)$$

5.5.3 Formal specification of the ECMA protocol

In order to formally specify the performance properties of the ECMA protocol, we must first write down its complete formal specification as a data type with axioms describing the relations between selectors and constructors. We need to identify only the axioms concerned with 'time' and these are given below. For convenience we use the labels pi and ti for the places and transitions, instead of their actual names.

Each of the axioms for Timeelapsed states that the time that has elapsed when a transition is fired can be stated in terms of the time elapsed since some previous transition fired. Its value then depends on the intervening 'Timein' variable(s). The previous transition(s) is identified by the Marking resulting from it. These markings are given in Table 5.1, and the timing diagrams which help to write down axioms for the variable Timeelapsed are given in Figures 5.8, 9, 10 for each of the three sequences I1, I2, I3, respectively.

The above is a consequence of the fact that in the ECMA TPN, a transition may be enabled by tokens becoming available at different times in each of its (more than one) input places. This is what gives

rise to the various possible J vectors for the TPN. This is in contrast to the situation in the relatively simpler INPAR protocol where each transition is enabled by only one input place with a time delay.

Table 5.1 Resulting Markings for the ECMA TPN

Transition	Resulting Marking
(Initial state)	Reading(1,6)
t1	Reading(2,7)
t2	Reading(3,7,9) or Reading(3,4,8,9)
t3	Reading(1,8,10)
t4	Reading(1,6)
t5	Reading(1,8,9) or Reading(1,6)
t6	Reading(2,4,8) or Reading(3,4,8,9)
t7	Reading(3,5,6)
t8	Reading(1,6)
t9	Reading(3,4,6) or Reading(1,6)

(Here, the marking after transitions (t2 // t6) is Reading(3,4,8,9) and after transitions (t5 // t9) is Reading(1,6))

Timing Diagrams

for I1, ie., (t1;t6;t3;t8;t1)

here $z2 \geq z7+z4$

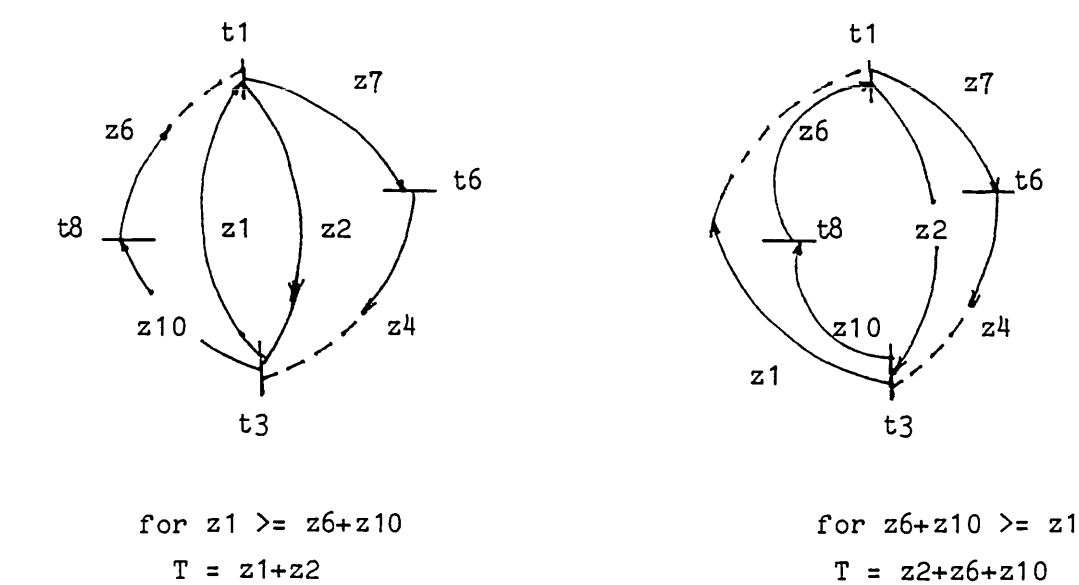
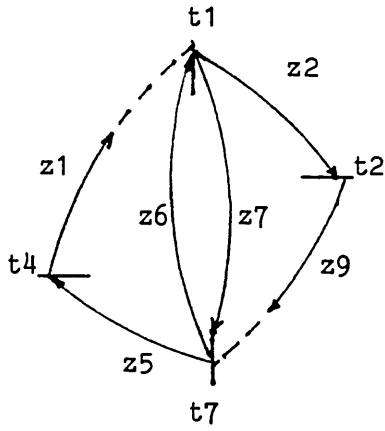


Figure 5.8 Timing Diagram for I1

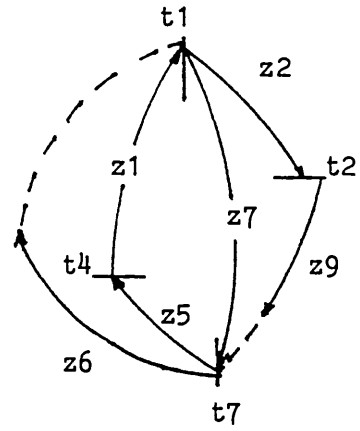
for I2, ie.. (t1;t2;t7;t4;t1)

here $z7 \geq z2+z9$



for $z6 \geq z1+z5$

$$T = z7+z6$$



for $z5+z1 \geq z6$

$$T = z7+z5+z1$$

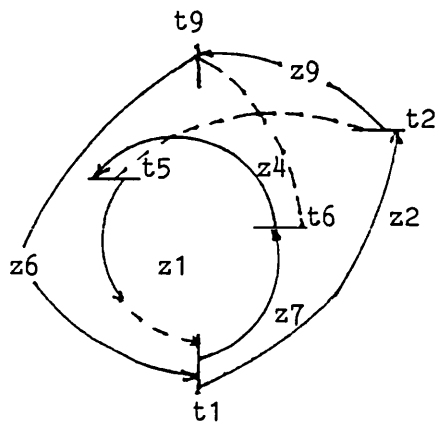
Figure 5.9 Timing Diagram for I2

for I3, ie.. (t1;t6//t2;t9//t5;t1)

(here $z4+z7 > z2$ and $z2+z9 > z7$)

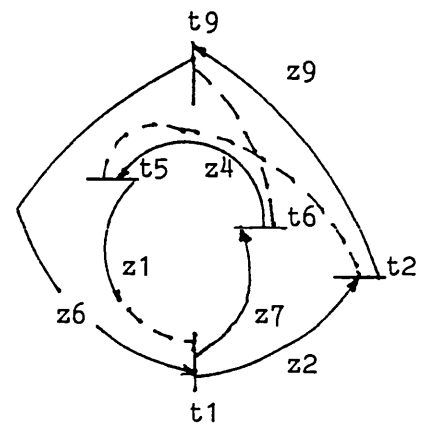
(a) (t1;t6;t2;t9;t5;t1)

(b) (t1;t2;t6;t9;t5;t1)



for $z2 > z7$

$$T = z2+z6+z9$$

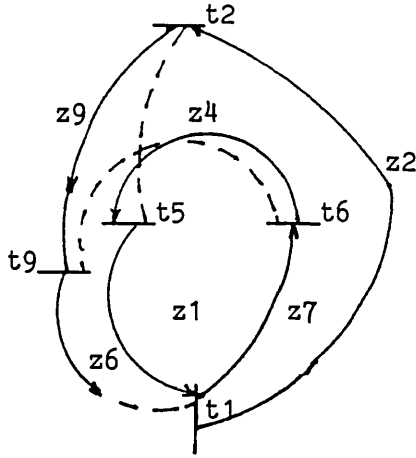


for $z7 > z2$

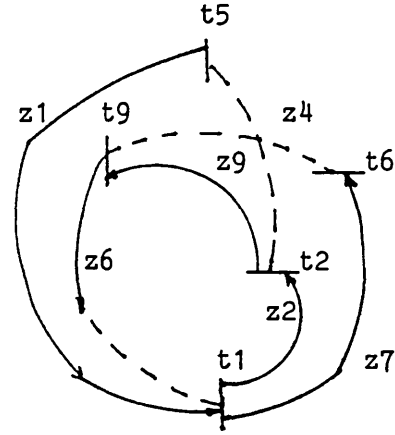
Figure 5.10 Timing diagram for I3

(c) (t1;t6;t2;t5;t9;t1)

(d) (t1;t2;t6;t5;t9;t1)



for $z2 > z7$



for $z7 > z2$

$$T = z1 + z4 + z7$$

Figure 5.10 Timing diagram for I3 (contd.)

The axioms for the selector variable 'Timeelapsed' are :-

```
Timeelapsed(t1(s)) == if Marking(s) = Reading(1,10,8)
    then Timeelapsed(s) + GreaterOf( Timein(p1) or ((Timein(p6)
        + Timein(p10)) )
    else if Marking(s) = Reading(1,6)
        then Timeelapsed(s) + GreaterOf( Timein(p6) or (Timein(p1)
            - Timein(p10)) )
    else if Marking(s) = Reading(3,5,6)
        then Timeelapsed(s) + GreaterOf( Timein(p6) or (Timein
            (p1) + Timein(p5)) )
    else Timeelapsed(s),
```

```
Timeelapsed(t2(s)) == if Marking(s) = Reading(2,7)
    then Timeelapsed(s) + Timein(p2)
    else if Marking(s) = Reading(2,8,4)
        then Timeelapsed(s) + Timein(p2) - Timein(p7)
    else Timeelapsed(s),
```

```
Timeelapsed(t3(s)) == if Marking(s) = Reading(4,8,2)
    then Timeelapsed(s) + Timein(p2) - Timein(p7)
    else if Marking(s) = Reading(2,7)
        then Timeelapsed(s) + Timein(p2)
    else Timeelapsed(s),
```

{by definition, the transition t3 can fire only if $z2 \geq z4+z7$ }

```
Timeelapsed(t4(s))==if Marking(s)=Reading(3,9,7)
    then Timeelapsed(s)+Timein(p5)+Timein(p7)-Timein(p2)
    else if Marking(s)=Reading(5,6,3)
        then Timeelapsed(s)+Timein(p5)
        else Timeelapsed(s),
```

```
Timeelapsed(t5(s))==if Marking(s)=Reading(3,9,7) or Reading(3,4,8,9)
    then Timeelapsed(s)+Timein(p4)+Timein(p7)-Timein(p2)
    else if Marking(s)=Reading(2,4,8) or Reading(3,4,9,8)
        then Timeelapsed(s)+Timein(p4)
        else if Marking(s)=Reading(3,4,6)
            then Timeelapsed(s)+(Timein(p4)+Timein(p7))
                -(Timein(p2)+Timein(p9))
            else Timeelapsed(s),
```

{the transition t5 can only fire if $(z4+z7) > z2$ }

```
Timeelapsed(t6(s))==if Marking(s)=Reading(2,7)
    then Timeelapsed(s)+Timein(p7)
    else if Marking(s)=Reading(3,7,9)
        then Timeelapsed(s)+Timein(p7)-Timein(p2)
        else Timeelapsed(s),
```

```
Timeelapsed(t7(s))==if Marking(s)=Reading(2,7)
    then Timeelapsed(s)+Timein(p7)
    else if Marking(s)=Reading(3,9,7)
        then Timeelapsed(s)+Timein(p7)-Timein(p2)
        else Timeelapsed(s),
```

{transition t7 can fire only if $z7 \geq z2+z9$, by definition}

```
Timeelapsed(t8(s))==if Marking(s)=Reading(4,8,2)
    then Timeelapsed(s)+Timein(p2)+Timein(p10)-Timein(p7)
    else if Marking(s)=Reading(1,8,10)
        then Timeelapsed(s)+Timein(p10)
        else Timeelapsed(s),
```

```
Timeelapsed(t9(s))==if Marking(s)=Reading(3,9,8,4) or Reading(2,4,8)
    then Timeelapsed(s)+Timein(p2)+Timein(p9)-Timein(p7)
    else if Marking(s)=Reading(3,4,9,8) or Reading(3,7,9)
        then Timeelapsed(s)+Timein(p9)
```

```

else if Marking(s)=Reading(1,8,9)
then Timeelapsed(s)+(Timein(p2)+Timein(p9))
-(Timein(p4)+Timein(p7))
else Timeelapsed(s);
{transition t9 can fire only if  $z_2+z_9 > z_7$ }

```

5.5.4 Statement of ECMA performance

The performance of the ECMA connection-disconnection phase can now be stated formally, using equations (4)-(6), as follows. The performance measure is the same as that considered for the INPAR protocol, ie., the time taken to complete any particular cycle of operations.

theoremperformance1

for all s, $\text{Timeelapsed}(t_1(t_8(t_3(t_6(t_1(s))))) = \text{Timeelapsed}(t_1(s)) +$
 $\text{GreaterOf}((\text{Timein}(p_1) + \text{Timein}(p_2)) \text{ or } (\text{Timein}(p_2) +$
 $(\text{Timein}(p_6) + \text{Timein}(p_{10})))$

The theorem performance1 states formally that the time taken for the connection-disconnection phase to be completed, if the accepting entity requests for disconnection, is equal to the greater of the times (z_1+z_2) and $(z_2+z_6+z_{10})$.

theoremperformance2

for all s, $\text{Timeelapsed}(t_1(t_4(t_7(t_2(t_1(s))))) = \text{Timeelapsed}(t_1(s)) +$
 $\text{GreaterOf}((\text{Timein}(p_6) + \text{Timein}(p_7)) \text{ or } (\text{Timein}(p_1) +$
 $(\text{Timein}(p_5) + \text{Timein}(p_7)))$

The theorem performance2 states formally that the time taken for the connection-disconnection phase to be completed, if the initiating entity requests for disconnection, is equal to the greater of the times (z_6+z_7) and $(z_1+z_5+z_7)$.

theoremPerformance3

for all s, $\text{Timeelapsed}(t_1(t_5//t_9(t_2//t_6(t_1(s))))) = \text{Timeelapsed}(t_1(s))$
 $+ \text{GreaterOf}((\text{Timein}(p_1)+\text{Timein}(p_4)+\text{Timein}(p_7)) \text{ or }$
 $(\text{Timein}(p_2)+\text{Timein}(p_6)+\text{Timein}(p_9)))$

The theorem performance3 states formally that the time taken for the connection-disconnection phase to be completed, if either of the two entities may request for disconnection is equal to the greater of the

times $(z1+z4+z7)$ and $(z2+z6+z9)$.

All the above theorems can be proved by Structural Induction.

5.6 GENERAL DISCUSSION

In this section we discuss the methodology in terms of its scope, limitations and possible extensions. As mentioned earlier, the methodology developed as a result of our attempt to do a formal specification of the liveness and performance property of the INPAR protocol.

5.6.1 Liveness Properties (~~something~~ eventually happens)

It can be seen that because of the definition of the firing of transitions in a TPN, each transition **has to fire** as soon as all its enabling (input) places have available tokens. This models the **eventuality** of the transitions. In other words, there are no probabilities associated with the firing of transitions and it can be said that there is always both a necessary and sufficient condition for a transition to fire.

Hence it was possible to specify liveness properties of the protocol using the Marking vector to represent the states of the protocol, as discussed in Section 2. We were able to state and prove that for every possible sequence of operation of the protocol, starting from the initial state, it was eventually possible to reach the same state again.

For the ECMA protocol phase considered in this paper, we can similarly specify its Liveness by stating the theorem **ECMAprogress** as

theorem**ECMAprogress**

for all $s1, s2$ ($s2 = t8(t3(t6(t1(s1))))$) and

Marking($s1$) = Reading(1,6)

imp Marking($s2$) = Reading(1,6));

This theorem is stated for the sequence of transitions given by I1. We can similarly state the liveness property for the sequences I2 and I3.

Temporal logic has hitherto been considered the most appropriate for specifying progress of a protocol [Schwar82]. Our approach using

Timed Petrinets are believed to be equally appropriate, with the added advantage of being simpler.

5.6.2 Timeouts

The inclusion of the selector variable representing time elapsed in the protocol (Timeelapsed) after each transition, enables us to specify any protocol in which the sender times out if a response (Ack) is not received from the receiver within a certain time interval. The actual Time Out event can be specified as an operation (transition) of the protocol TPN. Its enabling condition can be taken as the appropriate value of the variable Timeelapsed.

5.6.3 Control vs data

At present we have not explicitly considered separating control messages from data messages or identifying the actual data message since this is not possible to do with ordinary TPNs as hitherto defined. This may be necessary in order to specify more precisely the type of response (NAK/Ack) received, or the order of their transmission. Predicate/Transition nets are considered very useful for such representation [Genric80], where arcs to transitions give the variables (predicates) defining the pattern of tokens used by the transitions. Such Predicate/Transition (Pr/tr) nets can be expanded to obtain ordinary Petrinets [Berthe82].

By incorporating time delays in the places of the Pr/tr nets, resulting in Timed Pr/tr nets, it may be possible to analyze them for performance characteristics and thus specify the performance properties of protocols modelled by them.

5.6.4 Explosion of states

It is quite possible that in the TPN model of the protocol, there is an explosion of states for complex protocols. It then becomes difficult and sometimes impossible to specify each state separately. This difficulty was experienced for the INPAR protocol specification.

However, we were able to overcome this very common limitation of Petrinets, by considering some but not all the places of the net for defining its Marking vector (state). Places which did not contribute

at all to the 'conditions' for firing of transitions but were present only to model certain data structures of the protocol, were not considered in defining the state of the protocol. If necessary such a place was specified explicitly as an enabling variable.

This approach helped to reduce considerably the total number of distinct states in the protocol and hence simplify its formal specification. We feel that this is possible to do in most cases.

5.6.5 Deterministic Model

The TPN used to model the ECMA protocol can be seen to be a deterministic one because of its definition for transitions. It may be argued that such deterministic models do not effectively represent the non-deterministic character of a communication protocol as represented by the probabilistic nature of the underlying medium or of the user. For example, in the connection-disconnection phase of ECMA discussed above, it is possible that the data transfer state may be cut short by the arrival of a remote disconnection request. The model cannot represent this as the data transfer times are assumed constant (z_2 and z_7), ie., it is assumed that any remote request for disconnection can only arrive after the data transfer state is over. This assumption may not necessarily be true for the protocol. Thus our TPN model may be more deterministic than the actual protocol it models.

However, since the non-deterministic nature of the medium is manifested by the need for retransmissions, we are able to cover this fully by explicitly considering all possible transition sequences in the model.

5.6.6 Related Work

While there has been no reported work on the formal specification of the performance of a communication protocol using Timed Petrinets theory, two works recently published, merit attention here because of their similarity to our foregoing discussions on protocol performance. We compare their work with ours in the following paragraphs.

a) Unification of functional and performance analysis [Yemini82]
Yemini and Kurose were the first to argue. that the

hitherto prevailing concepts of protocol correctness (ie., safety and liveness) do not capture the notion of the time required to achieve the protocol goal. In this sense, they argue, the much verified AB protocol is not really correct, as long as performance analysis is not combined with functional analysis. It is necessary to show that the time periods for 'good things to happen' do not 'grow with probability one to infinity'.

Yemini et al have used a recently developed theory of the semantics of probabilistic program, to adequately capture the probabilistic behaviour of the protocol. This theory is concerned with the **distribution** or measure associated with a program variable rather than with its specific value. This measure can be seen as analogous to a fluid mass distributed over all possible values of the variable and being redistributed (probability transformed) by the execution of the program.

The authors argue that several aspects of protocol operation involving probabilistic and random times, can be modelled and analyzed by the semantics of probabilistic programs. This can lead to a system 'transformer' which can be expressed in terms of the conditions that enable transitions and the probability of the firing of transitions, once enabled. They show how this probability transformer has many analogies with Markov processes.

We see this probability transformer model of the protocol as very closely approaching the Stochastic Petri net model in which probabilities are associated with transition firing times [Molloy78] since both are analyzed using the theory of Markov processes.

However, we have dealt with the probabilistic nature of the protocol by explicitly considering all possible behaviour of the protocol due to this probability, as discussed earlier.

Yemini et al have not given a formal specification or verification of the protocol performance, only an intuitive approach to the solution of the protocol performance analysis problem.

b) Projection of time-dependent communication protocols [Shanka82]
Shankar and Lam have also emphasised that protocols, which belong to the class of time dependent distributed systems, cannot be adequately

modelled without measures of time. As such, they have modified an earlier basic protocol model of theirs with the use of time **variables** to measure elapsed times between systems, and time **events** to age these time variables. They develop mathematical models of the simplex stop-and-go protocol using these time measures and examine time constraints in the system.

Time variables correspond to timers that are actually implemented in the system. They may also be auxiliary variables used to state assertions about the system's behaviour and for representing time constraints of system modules.

Using the model and their theory of protocol projections [Lam82], they are able to verify the timeout condition and to calculate its proper value for preserving 'safety' of the protocol. They have not concerned themselves with performance of a protocol, only the proper use of timeouts, by considering time variables. As mentioned earlier, we have not considered timeouts explicitly, but used time variables for performance analysis.

5.7 CONCLUSION

The chapter has shown that the use of the effective representative power of Timed Petrinets, makes it possible to specify algebraically and prove certain performance properties of a communication protocol and hence increase our confidence in it. Timed Petrinets have been shown to be ideal tools to model, represent and specify protocol characteristics relating to liveness and performance.

We have specified performance in terms of the time taken to complete certain operations of the protocol. There may be other ways in which performance properties could be specified. As far as we are aware, this is the first reported work on formal specification of protocol performance properties, using Timed Petrinets, though a couple of papers have appeared recently, dealing with 'timeout' and 'recoverability' of protocols using time in Petrinets.

Liveness properties were hitherto considered to be difficult or impossible to specify using finite state machine models, but we have shown that use of TPNs also overcomes this limitation.

A real life protocol, the ECMA standard has been used to illustrate our performance specification and validation methodology.

We have shown in this chapter that, in addition to the reductions made possible in a Timed Petrinet model as given in the last chapter, we can further simplify our model for specifying its performance, once its safety and liveness have been proved.

We shall show in the next chapter how the performance values calculated for a protocol affect quantitatively the overall design and performance criteria for the distributed multiple processor system for which it has been designed.

CHAPTER 6

MESSAGE BUFFERING SUBSYSTEM AND COMPOSITE MULTIPROCESSOR MODELS

6.1 INTRODUCTION

This chapter deals with the third and final subsystem whose Timed Petrinet models have been considered in the thesis, namely, the message buffering system. It also deals with a composite model of a multiprocessor developed from this by incorporating the other two subsystems considered earlier in the thesis, ie., the bus contention subsystem and the communication protocol on the bus. Most researchers have reported performance analysis techniques for various subsystems only, ^{but} we believe this is the first reported work on the analysis of a composite multiprocessor performance using its subsystem's analysis. We show the ease with which this is accomplished using Timed Petrinet models, due to their flexibility and extensibility.

The message buffering system in the bus interface hardware is another example depicting conflict, concurrency and synchronization within a distributed multiprocessor. It is equivalent to the classical producer-consumer problem with bounded buffers and has been extensively modelled and studied using Petrinets [Peters81]. In [Sifaki77], it has been analyzed using Timed Petrinets to find the number of buffers required in the system, given producing and consuming times, both for single and multiple producer and consumer cases.

In a distributed multiprocessor system, the processor is a producer of messages and its bus interface hardware is the consumer sending the message across the bus. Messages may be buffered in a single or two sets of (outgoing and incoming) buffers. Two types of buffer access are possible in the communication hardware in distributed systems -

- a) the sequential memory access case, where either the processor or bus interface only can have access to the buffers at any instant
- b) the dual -port memory buffer case which allows simultaneous access to buffers by both processor and bus interface hardware.

We have used Timed Petrinets to model and analyze these two cases and

find the conditions under which each results in greater message throughput on a single shared bus. The usual design assumption is that a dual-port memory would result in less delays and hence greater message throughput in the system. Our analysis will show if and when this assumption is valid.

Further, the analysis will indicate quantitatively the following design and performance parameters for the bus interface hardware -

- a) Number of buffers required for each processor - BI pair
- b) Number of messages produced and consumed in a given interval of time, given message producing and transmitting times
- c) Effect of change in processing times on the above factors. This models the use of heterogeneous processors on the bus.
- d) Effect of change in consuming times. This models the change in protocol overheads due to error correction, if any.

As before, we shall use our Message Bus multiprocessor system characteristics to model a distributed system, using Timed Petrinets.

In Section 2 we consider the Timed Petri net model for sequential buffer access and find the number of buffers required in this case. Section 3 considers the Timed Petri net model for the dual-port memory case and shows how it differs from the previous case. In Section 4 we consider the composite multiprocessor model. Its performance evaluation results are given in Section 5. Section 6 discusses related work on performance analysis and also compares Timed Petri net models with such diverse objects as PERT/CPM charts and Queueing Network models. Conclusions are given in Section 7.

6.2 TPN MODEL FOR SEQUENTIAL BUFFER ACCESS

Fig. 6.1 represents the single producer-consumer system with a bounded buffer of capacity B . The left hand side represents the processor and the right hand side represents the bus interface hardware. The place P_b is the 'buffer available' condition. It is associated with a time delay Z_b representing the time between two consecutive accesses of buffer memory. Z_b may or may not have a zero value.

Places P_p and P_m with time delays Z_p and Z_m respectively, represent the message processing and consuming times. The number of available

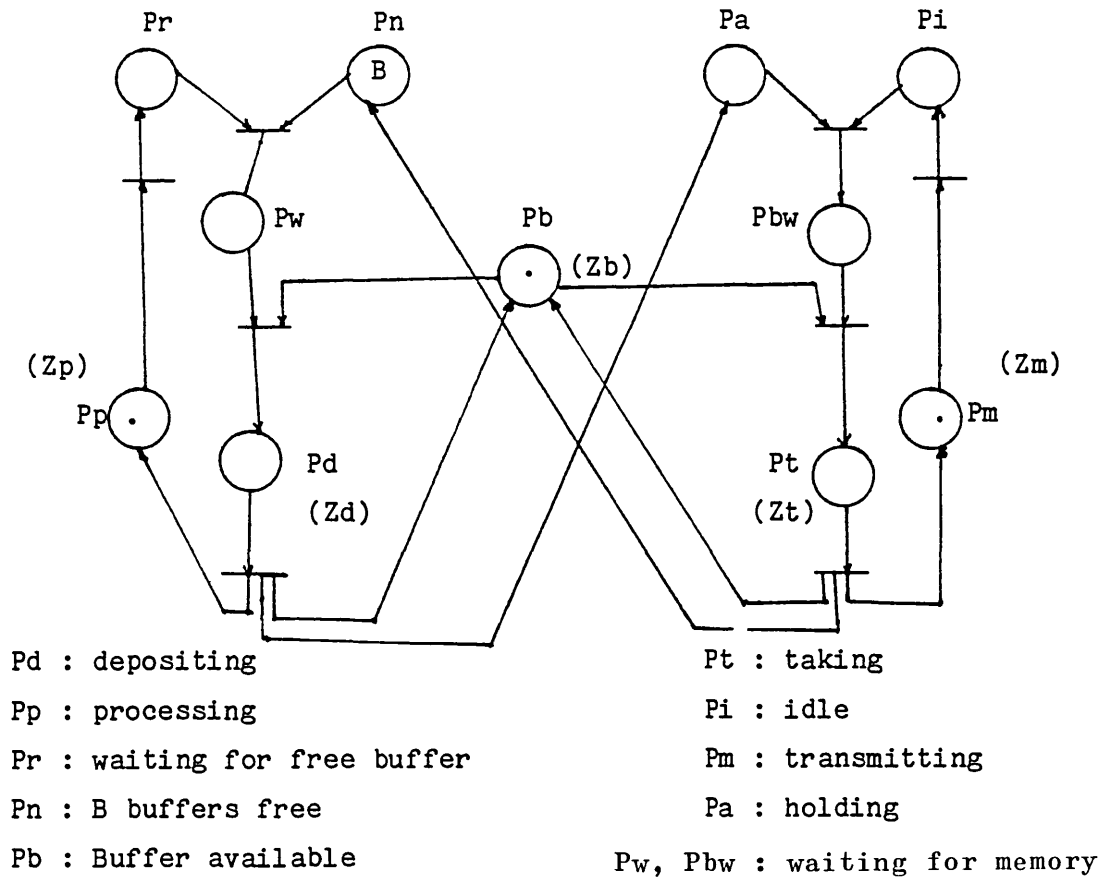


Figure 6.1 Sequential memory access buffer

buffers at any instant is given by the B tokens in place Pn. Depositing a message in the buffer (Pd) and removing it (Pt) takes time Zd and Zt respectively. The other places shown in the TPN do not have any time delay associated with them. The processor processes a message and may have to wait in the 'message ready' state (Pr) if no buffers are available. If a buffer is free, but the memory is being read by the bus interface, the processor waits in place Pw. The bus interface is normally idle (Pi). When a message has been deposited in the buffer (Pa), it waits (Pbw) for the buffer available condition to be true and/or removes it for transmission on the bus. The place Pa represents the state where a message in a buffer is being held for consumption by the bus interface.

Analysis of the TPN of Figure 6.1 leads to the following relations, corresponding to the cover by the four state graphs -

$$\begin{aligned}
 i &\leq 1/(Z_p + Z_d) & \text{where } i \text{ is the maximum rate of computation} \\
 & & \text{of the net and represents the number of} \\
 i &\leq 1/(Z_d + Z_t + Z_b) & \text{messages transferred in unit time, ie.,} \\
 & & \text{the message throughput rate for the pro-} \\
 i &\leq 1/(Z_m + Z_t) & \text{cessor-Bus Interface pair.}
 \end{aligned}$$

$$i \leq B/(Z_d+Z_t)$$

Thus the value of i is given by

$$\begin{aligned} i &= \min. (1/(Z_p+Z_d), 1/(Z_d+Z_t+Z_b), 1/(Z_m+Z_t), B/(Z_d+Z_t)) \\ &= \min. (1/(Z_p+Z_d), 1/(Z_d+Z_t+Z_b), 1/(Z_t+Z_m)) \end{aligned}$$

This shows that the cycle time of the message buffer system increases with increase in processing and/or consuming times, as long buffer read/write times are low. This is what we would normally expect to happen. Cycle time is the inverse of the message throughput rate in the buffering system, where the latter is defined as the number of messages processed and transmitted by each processor - Bus Interface pair. Note that this is different from the message throughput rate discussed for the bus contention system which is the overall rate for the multiprocessor.

If $T = 1/i$ denotes the actual cycle time of the message buffer system, then $B = (Z_d+Z_t) / T$

But T is always $\geq (Z_d+Z_t)$ from the relations for i , giving $B = 1$.

This means that the number of buffers required is always one. It also means that wait states are introduced in the buffer read/write loop and their value depends on the message processing and consuming loop times. For example,

if $T = Z_m+Z_t$, then messages deposited in the buffer have to wait for a time

$$dw = (Z_m+Z_t) - (Z_d+Z_t) = Z_m-Z_d \text{ to be consumed.}$$

But if $T = Z_p + Z_d$, then the free buffer has to wait for a time

$$dr = (Z_p + Z_d) - (Z_d + Z_t) = Z_p - Z_t \text{ for the next message to be ready.}$$

On the other hand, if $T = (Z_d+Z_b+Z_t)$, then the processor and /or the bus interface wait for the buffer to be available.

6.3 TPN MODEL FOR SIMULTANEOUS BUFFER ACCESS

If in Figure 6.1 we remove the place P_b representing the availability of buffer, we get the TPN model for the case when the message buffer memory can be simultaneously accessed by both processor and bus interface. This represents a buffer implemented by a dual-ported memory and the resulting model is shown in Figure 6.2. The places P_w and P_{bw} representing wait states by the processor and bus interface

are no longer present in the TPN as both can 'apparently' access the buffer memory simultaneously, using their separate ports. (What actually happens, of course, is that the dual-ported memory has half the cycle time of an equivalent single-ported memory and the processor and bus interface access it in alternate cycles, giving the impression that the buffer is being simultaneously accessed by both.)

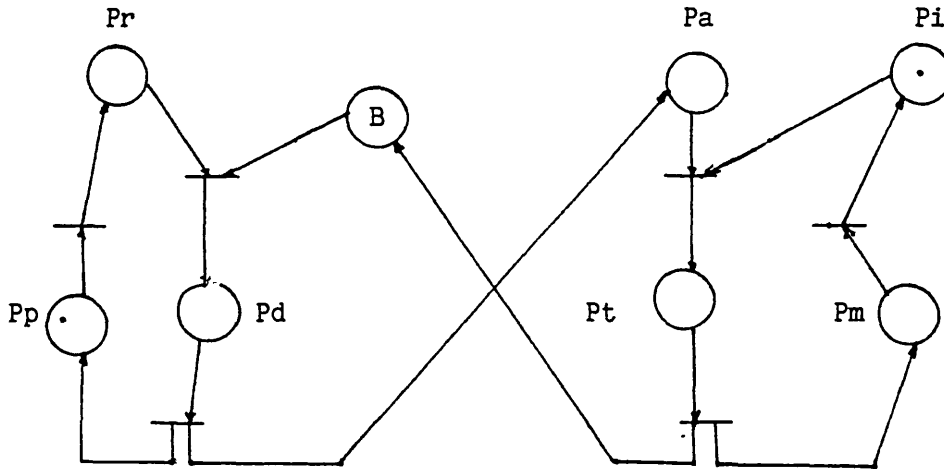


Figure 6.2 Simultaneous access memory buffer

The analysis of the TPN of Figure 6.2 results in the second inequality of the previous case for i to be removed, ie.,

$$imax. = \min. \{ 1/(Zp+Zd), 1/(Zt+Zm), B/(Zd+Zt) \}$$

implying that the computation rate and hence cycle time now depends on the value given to B , in addition to the other time values, as in the previous case.

$$\begin{aligned} \text{Let } T &= \text{cycle time} = 1/imax. \\ &= \max \{ (Zp+Zd), (Zt+Zm), (Zd+Zt)/B \} \end{aligned}$$

If $T = (Zd+Zp)$ or $(Zt+Zm) > Zt+Zd$ then $B = 1$ as before.

But if $(Zd+Zt) > (Zp+Zd) > (Zt+Zm)$

then T has a value less than $(Zd+Zt)$ and is given by

$$T = (Zd+Zt) / B \quad \text{and } B > 1.$$

We conclude from the above analysis that -

- a) If the buffer loop time, ie., the sum of the message write and read times in buffer is small as compared to processor or Bus Interface time, then the presence of a dual-ported memory does not affect the cycle time of the system and hence its message throughput rate. The number of buffers required is always 1.

- b) On the other hand, if the buffer loop time is large as compared to processor or Bus Interface times, then the dual-ported memory helps to decrease the cycle time and hence increase message throughput rate, allowing an increased number of buffers to be used in the system.
- c) A decrease in processing or consuming times decreases the cycle time, ie., increases the message throughput rate as long as the former are larger than the Buffer loop time.

6.4 COMPOSITE TPN MODEL FOR A MULTIPROCESSOR SYSTEM

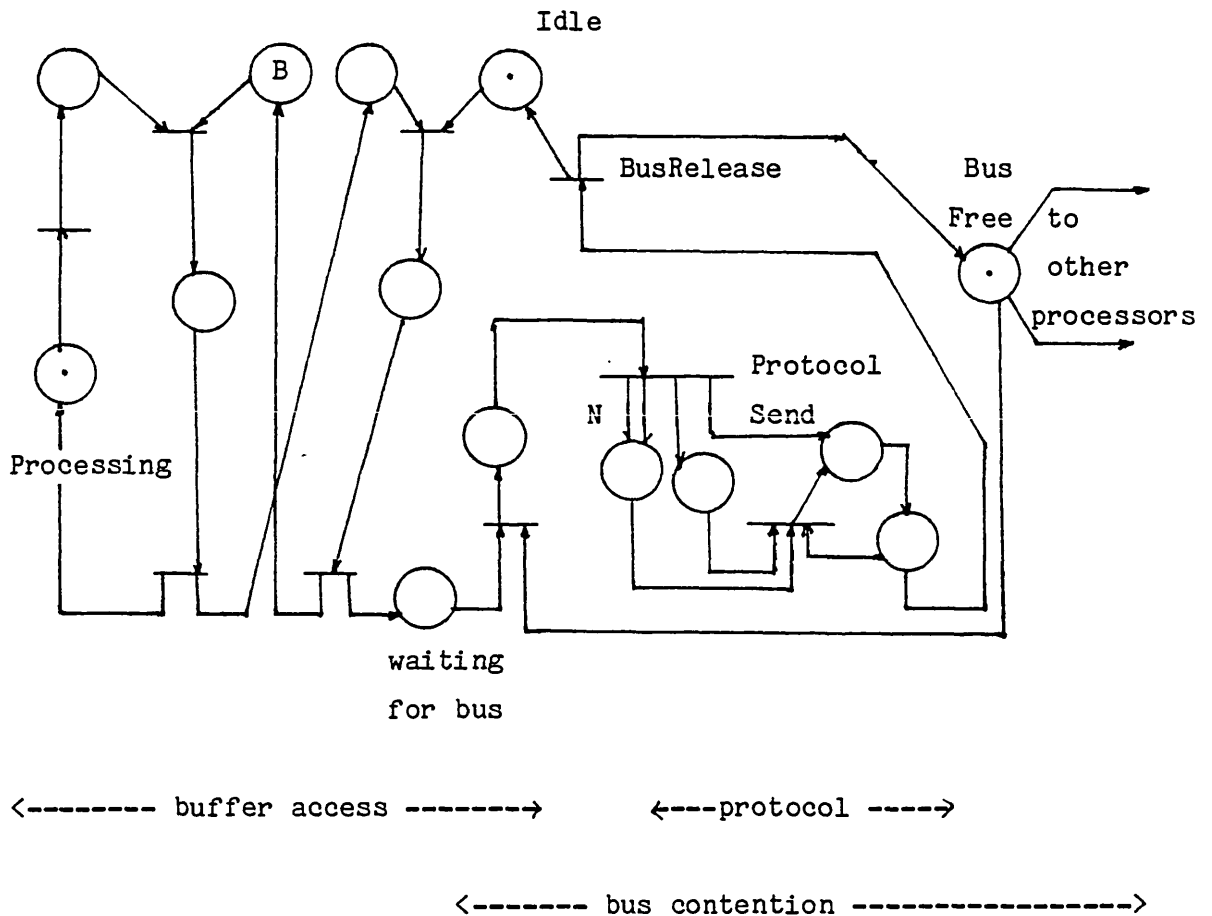


Figure 6.3 Composite Timed Petrinet for multiprocessor

The composite model for the complete multiprocessor system can be drawn and analyzed using the three TPN models considered and discussed earlier, ie., for the bus contention system, the communications protocol and the buffer access systems. This composite model is shown in Figure 6.3 and is typical for every processor and bus interface pair on the bus. The three individual subsystem models can be clearly seen in the composite model.

The composite net can be viewed as the Timed Petrinet of the buffer access system, in which the place Pm representing message consumption, has been split into two places, one for waiting and one for accessing the bus by the Interface hardware. These correspond to similar places in the bus contention model. The place representing bus access has been further replaced by the net of the communication protocol.

It is clear from the composite net that such a representation has the advantage that it need not be analyzed afresh; the analysis of the previous section for buffer access still holds. The only change to be made is in the value of the buffer loop time now applicable. This is affected by the presence of the other processor-BI pairs and the wait state Zw introduced. The buffer loop time is in fact the cycle time of the bus contention system given by T_{BC} where

$$T_{BC} = \max. [\max. ((P_i + A_i), i=1, \dots, 16), \sum A_i]$$

Here, P_i = processing time for processor i = Z_p

and A_i = accessing time

= time taken by the protocol to send the message on the bus

= $(Z_2 + Z_3) \cdot (N + 1)$ where

N = Number of retransmissions

Z_2 = message sending time

Z_3 = response receiving time

Thus, for the composite TPN of the multiprocessor,

cycle time $T = \max. \{ 1/(Z_p + Z_d), B/(Z_d + Z_t), 1/T_{BC} \}$

In other words, the message throughput rate of any processor -Bus Interface pair depends not only on the processing and transmitting times and the number of buffers used, but also on the number of processors in the multiprocessor.

The composite model can, of course, be viewed as a modified bus contention or protocol model and the same relation for T would apply.

Next, we shall consider our Message Bus multiprocessor system with heterogeneous microprocessors connected to the bus, for illustration of the above conclusion, using typical values for the parameters used.

6.5 PERFORMANCE EVALUATION OF MULTIPROCESSOR

Processing times

We consider typical 8-bit and 16-bit microprocessors available today, with varying degrees of power and capability, ie., MC6800, MC6809, Z80, Intel 8086, Z8000, MC68000 and LSI 11/23 etc. If we assume that an average task to be performed by these is equivalent to a program of about 1000 instructions, then the processing times for the above microprocessors will range from 2 to 7 msecs., ie., $Z_p = 2$ to 7 msecs.

Buffer times

The time taken by the processors to deposit a 132 byte message after the above processing, in their private memory, by block transfer or otherwise, ranges from 500 μ secs. to 1msec. Let $Z_d = 700\mu$ secs.

Similarly, the time taken by the Bus Interface hardware to read such a message from the processor's memory by cycle-steal DMA ranges from 200 μ secs to 400 μ secs. Let $Z_t = 300 \mu$ secs. $\Rightarrow Z_t + Z_d = 1$ msec.

Transmitting times

If we assume that the data rate on the Message Bus is about 2 megabits per second, then transfer of a message of 144 bytes (including protocol overheads) takes 580 μ secs., i.e., $Z_2 + Z_3 = 580 \mu$ secs.

For an average 2 retransmissions for each message, ie., $N=2$

$$\Rightarrow Z_c = (N+1).(Z_2+Z_3) = 1740 \mu\text{secs.}$$

Again, for 16 processors on the bus, each processing and transmitting a different length of message, the average transmitting time A_i can be taken as 1000 μ secs., giving $\sum A_i = 16$ msecs. $\gg (P_i + A_i)$ for all i .

Hence $T_{BC} = \sum A_i$ (here the value of m_i as defined in Chapter 3 is 1, for all i).

Similarly for 2 processors on the bus, $T_{BC} = P_0 + A_0 = 7+1 = 8$ msecs. since $\sum A_i = 2$ msecs. and some m_i may be > 1 . In other words, the bus interface loop time can vary from 8 to 16 msecs. for a multiprocessor

system.

The above gives a rough idea of the range of values that can be expected given the range of available microprocessors.

In general, then, we can say that the buffer loop times are much smaller than the processing and consuming times, implying that the **number of buffers needed will not be larger than 1.**, unless the buffer read and write times are made comparable to processing and transmitting times either by using much faster processors and bus data rates (fiber optic Bus?) or by using program controlled I/O for buffer memory. In such cases, the number of buffers required for each processor - Bus Interface pair will increase.

The graph of Figure 6.4 shows how and when such a shift in B will be necessary for the range of processing and transmitting values considered above.

Multiprocessor Speed Up, Processing Power and Efficiency

Using the above values for the multiprocessor, it is possible to calculate its Processing Power, Speed Up and Efficiency.

Let task processing involve 1000 instructions per processor, and the time taken to do this be 3msecs., ie., $Z_p = P_i = 3 \text{ msecs.}$, for all i.

Hence, $Z_p + Z_d = 3.7 \text{ msecs.} > \text{buffer loop time} \Rightarrow B = 1$

This means that cycle time T of the multiprocessor is given by

$$T = \max. [3.7, T_{BC}] = \max [3.7, P_i + A_i, \sum A_i]$$

Let messages be of length 132 bytes, and let us assume that $N=0$, ie., no retransmissions are needed on the bus and that its data rate is 10 Mbits/sec. Then $A_i = 116 \text{ } \mu\text{secs.}$

For 16 processors on the bus, $\sum A_i = 1.85 \text{ msecs.} < P_i + A_i$

Hence, $T_{BC} = P_i + A_i = 3.116 \text{ msecs.} \Rightarrow T = 3.7 \text{ msecs.}$

The Processing Power PP_{16} for the multiprocessor is thus

$$= 16 \times 1000 / 3.7 = 4.33 \times 10^6 \text{ instructions/sec.}$$

$$= 4.33 \text{ MIPS}$$

This may be compared with the instruction rate for a single processor which is found to be (at 1000 instructions in 3 msecs.)

$$PP1 = 1000/3 = 0.33 \text{ MIPS}$$

A single processor would perform the complete 'task' in $T1=16 \times 3=48$ msecs., giving Speed Up for the multiprocessor as $= T1/T16 = T1/T = PP16/PP1 = 16 \times 3/3.7 = 13$ giving an Efficiency of $E = 13/16 = 0.81$

Table 6.1 Multiprocessor performance for various data rates on bus and number of retransmissions

Data rate (Mbits/s)	N	Ai (usecs.)	P0+A0 (msecs.)	$\leq A_i$ (msecs.)	T	P (MIPS)	Sp	E
10	0	116	3.11	1.85	3.7	4.33	13	.81
	1	232	3.23	3.7	3.7	4.33	13	.81
	2	348	3.35	5.57	5.57	2.8	8.6	.53
	3	468	3.47	7.42	7.42	2.14	6.4	.4
5	0	232	3.23	3.7	3.7	4.33	13.1	.81
	1	464	3.47	7.42	7.42	2.14	6.4	.4
	2	696	3.7	11	11	1.45	4.36	.27
	3	928	3.92	14.8	14.8	1.2	3.24	.2
2	0	580	3.58	9.28	9.28	1.72	5.17	.32
	1	1160	4.16	18.5	18.5	0.84	2.6	.16
	2	1740	4.74	28	28	0.57	1.7	.106
	3	2320	5.32	37	37	0.43	1.3	.08
40	0	30	3.03	.48	3.7	4.33	13	.81
	1	60	3.06	.96	3.7	4.33	13	.81
	2	90	3.09	1.44	3.7	4.33	13	.81
	3	120	3.12	1.92	3.7	4.33	13	.81

Table 6.1 gives the values of P, Sp and E for different values of retransmission times and data rate on the bus, for message lengths of 132 bytes, message processing time in each processor of 3msecs. and for 16 processors on the bus. Note that the maximum possible value of Processing power for 16 processors would be the hypothetical value (if communication time is zero) of $16 \times 0.33 = 5.28$ MIPS. Data rates of 40 Mbits/sec. correspond to an optical fibre bus. These results are plotted in the graph of Figure 6.5 (on page 135).

6.6 COMPARISON WITH RELATED WORK

This section is divided into two parts. In the first part we discuss related work on performance evaluation of computer systems in general. Here the authors have used other Timed Petrinet models and techniques. In the second part, we compare and contrast our Timed Petrinet models with two very different but nonetheless well known and conventional techniques, namely, PERT/CPM charts and Queueing Network models. We discuss when and where these different techniques can be used.

6.6.1 TPNs with fixed transition firing times

In [Zubere80] Timed Petrinets with fixed firing times associated to transitions have been used (as originated by Ramchandani) in order to give the preliminary performance evaluation for different uniprocessor computer architectures.

For a certain class of TPNs as described above, Zuberek defines its instantaneous description ID which is nothing but the state of the net at that instant. He derives the relation of reachability of that ID and based on this, a finite labelled directed graph of reachable ID is constructed. This represents the behaviour of the timed net. Labels represent probabilities and times of transition between vertices of the graph. Subsequent analysis is done using finite state Markov Chains because the labelled graph forms a probabilistic graph similar to Markov chains.

The method is applied to evaluation of the performance indices for two simple uniprocessor architectures by constructing their equivalent TPN models. One is a simple sequential execution architecture and the other is a pipelined one. The performance index chosen is the number of instructions executed in a second (Mega Instructions/sec. or MIPS).

The resulting models are very complex indeed, even for the simple architectures discussed. More realistic examples could well be unmanageable. We have not understood the advantage of converting the initial fixed firing times, defined as some time functions assigning positive rational numbers to each transition, into probabilities, giving rise to nondeterministic models.

6.6.2 Asynchronous concurrent real time systems [Ramamo80]

Ramamoorthy and Ho have used Timed Petrinets with fixed transition firing times to model real time control systems for their performance evaluation. Based on the model, a consistent system is classified as being decision-free, or safe-persistent or general. The performance measure considered is minimum cycle time for processing a task of the system. Procedures for predicting and verifying this are presented for all three types of systems.

It has been shown that for the first two types, performance can be verified quite efficiently and a technique is given to do this. The technique seems very elaborate and enumerative, and, according to the authors themselves, is useful mainly for verifying a given system performance rather than predicting it. We feel that Sifakis's theory for Timed Petrinets is neater and more compact, even though in both methods, computations involved are of $O(n^2)$ where n is the number of places and transitions in the Petrinet.

For the case of a general system, the authors have proved that the verification of system performance is an NP-complete problem. A heuristic method for computing the upper and lower bounds for such a system is proposed. The TPN models of Sifakis can be classified as general systems but it is still possible to predict an upper bound on the cyclic rate i (reciprocal of cycle time) of the transitions.

Another difference is that we have shown that in our bus contention system model, which is another example of a general system, the problem of throughput maximization is solvable in pseudo-polynomial time since it is not strongly NP-complete.

6.6.3 Time driven systems [Coolah83]

In this recently reported work, the authors have used the same class of Timed Petrinets as ours, without explicitly referring to Sifakis's work, to model what they have called time driven systems. The model incorporates a master timing mechanism which controls the repetitive operation of similar activities at regular intervals. Several subclasses of time driven systems with increasing levels of complexity are defined using their models. For each class they derive the conditions under which the Petrinet model can be proven to be 'safe',

thus giving timing requirements for the individual processing 'modules' in the system. No formal analysis, like reachability or liveness techniques, have been used, but several new conditions have been given to prove safeness in the presence of time.

The approach taken is thus exactly opposite to ours. Whereas we have used Timed Petrinet analysis to 'predict' performance of a real time system in terms of cycle time and throughput, given processing and execution times, Coolahan et al have determined the relations needed between execution times for modules in order to maintain a given cycle time or response time in the system.

6.6.4 Comparison with PERT/CPM scheduling procedures

It can be seen that there are some striking resemblances between Sifakis's fixed delay TPN models for system representation and the arrow (network) diagram representation of the planning phase of any project by PERT/CPM scheduling techniques of Operations Research. These resemblances have been made more apparent by the addition of the notion of time in Petrinets.

Both ordinary PNs and PERT/CPM charts represent the interdependencies and precedence relationships among their various activities (places) and events (transitions) [Taha81]. With TPNs as with PERT/CPM charts, time values are associated with each activity and delays are observed if two activities are concurrent. Both can be used to determine the minimum or maximum times of the completion of the project or system.

On the other hand, there are many differences. Obviously, the PERT/CPM chart represents a project which is only a one time effort, ie., where each activity or event executes only once. In the TPN model, however, a periodic cycle of computation is usually followed. Secondly, the TPN model shows clearly, the 'conflict' between two contending activities and the disabling of one event if the other fires; the PERT/CPM chart is unable to do so. This means that whereas the latter can always be represented by an equivalent TPN model [Peters81], the converse is not true. Our bus contention model is an excellent example of this shortcoming of the PERT/CPM chart.

Thirdly, PERT/CPM charts are usually analyzed using probability theory when time estimates for different activities are probabilistic. Not

so our TPN models, which as mentioned earlier, are deterministic models which do not require any such probability considerations.

Finally, it is seen that the PERT/CPM diagram has normally to be updated and analyzed to determine new schedules if necessary, but no such updating is required for the TPN diagram which models the system's whole working life.

Chrettiene [Chrett83] has modelled and studied the above classical scheduling problem with resource and succession constraints, using Timed petrinets, which he has called Timed Event Graphs due to this very analogy. He has solved the scheduling problem for periodic (recycling) firings, and also analyzed its earliest controlled execution, both of which, as mentioned above, were not possible to do in the classical case. He has shown that the twin domains of Timed Event Graphs and Operations Research actually enrich each other.

6.6.5 Comparison with Queueing Network models

Queueing network models have played a very important role in the performance modelling of computer systems, since their first use in the late 1960s. That was the time when Interactive User Terminals first appeared and it was required to predict their performance. The reason for the popularity of Queueing network models is, of course, the efficient way in which Queueing Theory can handle random system parameters like job arrival times and their distributions.

Such models are now being used for the Reliability Analysis of multiple processor systems, since this is a major point of interest in the latter's design. Reliability of a multiprocessor system is dependent on the breakdown pattern of the processors and also of their repair pattern. This in turn dictates the efficiency of the system in terms of the processing capacity available at any instant.

Mitrani and King [Mitran83] have studied the tradeoffs between efficiency and reliability in M/M/N Queueing systems subject to breakdowns. They have assumed two different patterns of breakdown and repairs and found their effect on the optimal number of servers (processors) in the system. They have obtained the maximum possible rate of arrival of jobs in the system, given the number of servers, their service rate and the pattern of breakdown.

It is interesting to see that it is possible to arrive at the same value of 1 for the system when it is modelled using Timed Petrinets and analyzed as discussed in the thesis. Obviously, we would use fixed values for all time parameters in the TPN model instead of the random ones used in the Queueing model which deals with Poisson and exponentially distributed service and interarrival times. The fixed values used in the TPN model can be considered as the means of the random variables used in the Queueing models. Further, the former can be thought of as modelling, for example, the MTTR (Mean Time To Repair) and MTBF (Mean Time Between Failures) in the system.

Figs. 6.6 & 6.7 show, respectively, the Queueing Network models for an n processor system for two types of breakdowns [Mitran83], ie. systems A and B. For both it is assumed that jobs arrive at rate 1 in a Poisson stream and the service rate at each server is m/N , where m is the service rate for a single server system and N is the number of servers in the multi-server system. Breakdowns in system A are modelled such that each server goes through alternating periods of being operative and inoperative independently of others and the mean lengths of these periods are $1/e$ and $1/n$. For the system B, breakdowns are modelled such that the system as a whole is subjected to a stream of breakdowns arriving at rate s . These are treated as special 'jobs' with mean service time $1/n$ and have preemptive priority over normal jobs.

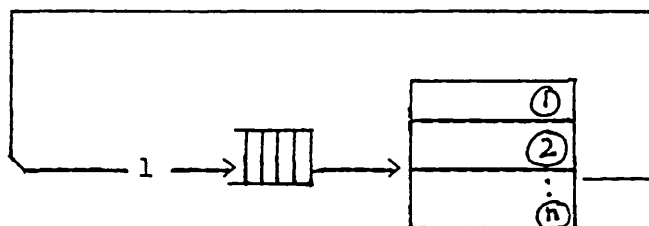


Figure 6.6 Queueing network model for system A

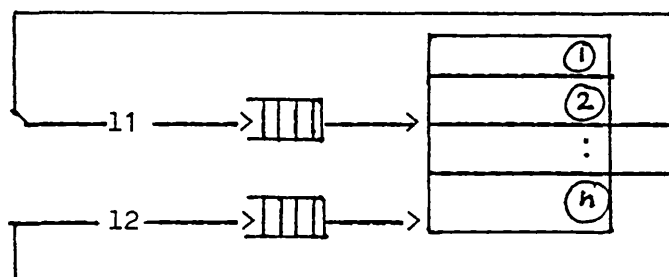


Figure 6.7 Queueing network model for system B

It has been proved by Mitrani and King that the nonsaturation conditions for the systems are given by

$$\begin{aligned} 1 &\leq mn / (e+n) && \text{for system A} \\ \text{and } 1 &\leq m [1 - s/Nn] && \text{for system B} \end{aligned}$$

The Timed Petrinet models for systems A and B are given in Figures 6.8 and 6.9 respectively. The place p1 shows the N ready servers in both. In Figure 6.9, p2 models the inoperative condition of a server whose time delay is 1/n, p3 shows the operative state with time 1/e which is the same as the reciprocal of the service rate of each server. Place p4 models the 'job ready' condition of the system. Analysis of the TPN of Figure 6.8 gives $i1 = i2 = i3$.

But i1 is the rate of arrival of jobs in the system and is given by 1. and $i2 \leq N / (1/e + 1/n)$. Using the fact that $1/e = N/m$, we can show $i2 \leq mn / (e+n)$ Hence $1 \leq mn / (e+n)$ for system A.

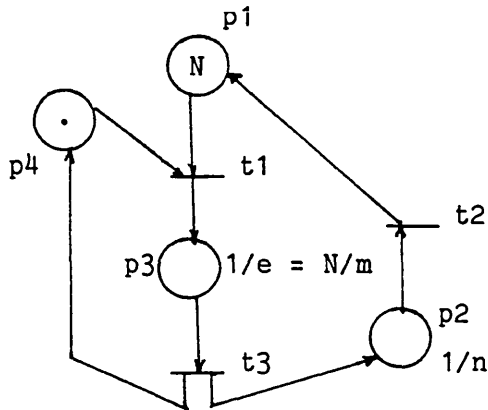


Figure 6.8 Timed Petrinet model for system A

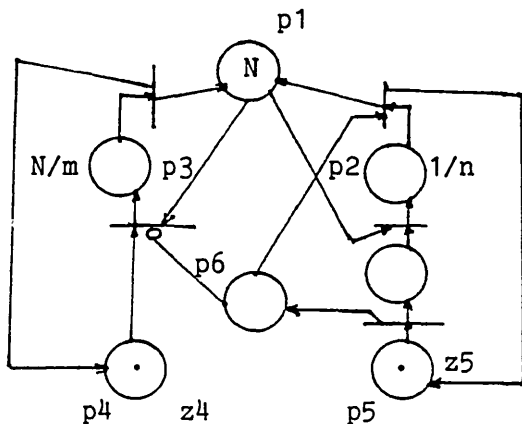


Figure 6.9 Timed petrinet model for system B

For the TPN model of Figure 6.9, the two streams of 'jobs' can be clearly seen. The service times for the regular and breakdown jobs are modelled at p3 and p2 by N/m and $1/n$ respectively. Places p4 and p5 model the 'job ready' states for the two types of jobs. Place p6 shows that the breakdown jobs have priority over the regular jobs.

Analysis of the TPN of figure 6.9 gives

$$i1 = 1 \leq 1/(z4 + N/m)$$

$$i2 = s \leq 1/(z5 + 1/n)$$

$$N \geq i1.N/m + i2.1/n$$

On substitution $N \geq 1.N/m + s/n$

hence $1 \leq m. [1 - s/Nn]$ for system B

Thus it can be seen that the same results can be obtained by both the TPN and the Queueuing network models of the systems. Expressions for the Response Times of the systems can be obtained to study how it varies with varying values of N , and the type of breakdowns, as shown by Mitrani and King.

6.7 CONCLUSION

We have considered in this chapter the classical producer-consumer problem in a new light, ie., as applicable to multiprocessor system analysis. The processor is seen as the producer of messages, depositing each into a bounded buffer. The bus interface is the consumer, transmitting the message in the buffer over the bus.

Two cases have been considered for this subsystem. Firstly, the buffer is seen to be implemented by a sequential access memory, with either the processor or the bus interface gaining access to it at any time. In the second case, the buffer is seen as being implemented by a simultaneous access memory, where both the processor and the bus interface access it simultaneously (dual-port memory).

Contrary to expectations, it has been shown that while a dual-port memory has certain advantages over single port memory when used as a buffer between producer (processor) and consumer (shared bus interface), it does not necessarily lead to lesser delays in the system. This has been demonstrated by quantitative results, using Timed Petrinet models for the system.

Next, we have added the TPN models of the other two subsystems considered in the thesis, i.e., the protocol model and the bus contention model and drawn and analyzed a composite TPN model for the multiprocessor. We have shown the ease with which Timed Petrinet models allow this to be done.

Further, it has been shown how certain important design and performance criteria can be quantitatively evaluated in great detail, to give realistic values for multiprocessor performance measures like Speed Up and Efficiency. It thus helps to fill in the 'gaps' produced by an informal design procedure.

We believe this is the first time a composite design and performance validation has been done for a multiprocessor system using the individual design and performance criteria of its subsystems. In this respect we have compared our approach to that taken by other researchers for performance evaluation of multiprocessor systems. We have also compared TPN models with Queueing Network models and PERT/CPM charts.

Our analysis for a composite system can be easily automated using recently reported techniques for Petrinet analysis [AFCET 83]. The work in this chapter can thus be viewed as providing the first steps in a unique approach to the Computer Aided Design of Large Scale Systems, using net formalisms.

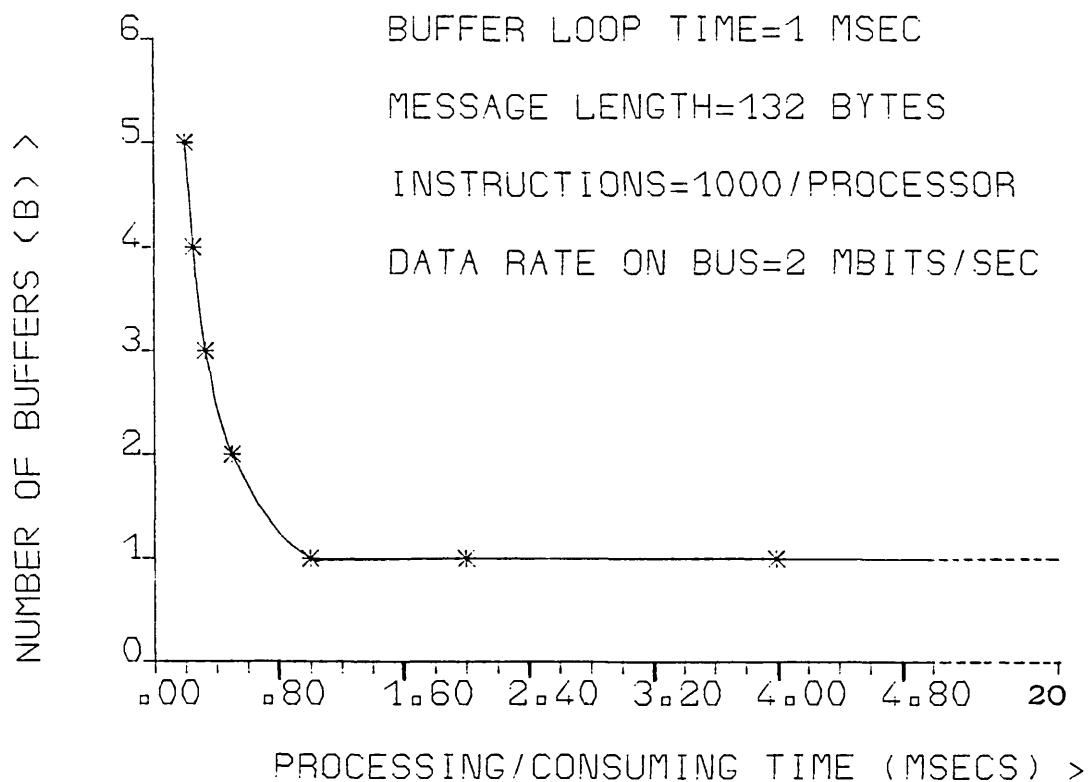


Fig. 6.4 Buffer Requirement in Multiprocessor

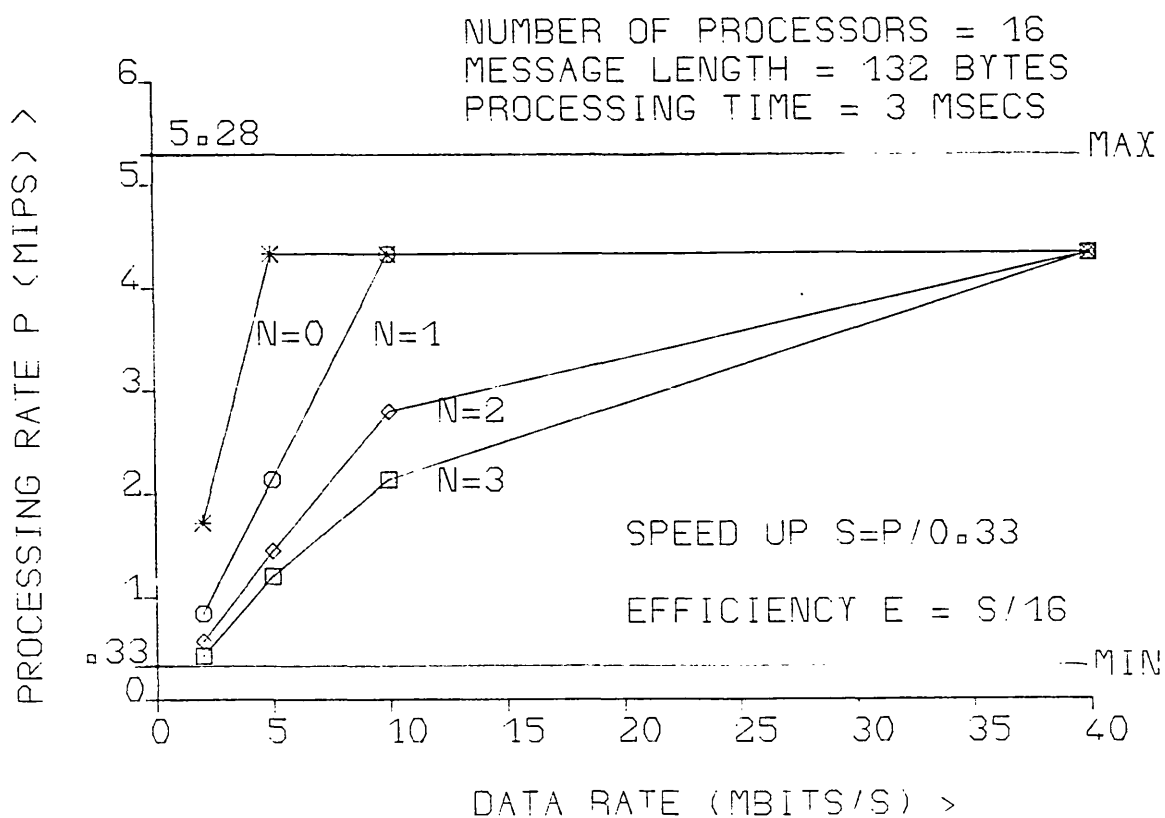


Fig. 6.5 Processing Power of Multiprocessor

CHAPTER 7

SUMMARY AND CONCLUSIONS

7.1 SUMMARY AND CONTRIBUTION

The thesis has addressed the technical issues involved in the application of Timed Petrinets theory and modelling to the efficient design and performance validation of multiple processor systems. It thus touches upon and contributes to two distinct areas of Computer Science, viz., Distributed, multiple processor systems and Mathematical Modelling and Analysis of Computer systems.

Traditionally, a DCCS station has been implemented as a uniprocessor structure, with the single processor performing all application and communication functions. In chapter 2 we justified the need for a multiprocessor structure in order to achieve desirable properties like modularity, extensibility and reliability in the station. Such a design necessitates emphasis on three aspects, viz., choice of a suitable interconnection medium, identification of its functional requirements and design of an error-free communication protocol for reliable message transfer between processors connected to it.

We have examined in depth the functional requirements of such an interconnection medium for reliable operation in a real time environment. These have been influenced largely by the CONIC Operating System designed and developed at Imperial College for DCCS and the structure of the station is intended to reflect and complement the latter's modular structure.

The result is the **Message Bus**, a low cost, single, serial, shared Bus which gives rise to a loosely coupled architecture for the station. Transactions on the Bus are in the form of messages rather than words, which suits the message-based CONIC architecture.

The Message Bus differs from other typical multiprocessor buses and Bus standards since there is no shared memory on it. It is similar to Local Area Networks in this respect, but has been designed for distances within a crate.

The communication protocol designed for the Message Bus is called INPAR (for Immediate, Negative, Positive Acknowledgements and Retransmissions) protocol and a major design decision for it has been to implement it fully in the hardware of the Bus Interface of every processor. This is done in order to achieve speedy message transfer commensurate with real time requirements.

We have shown how, using the WIRED-AND implementation of the Message Bus and certain features of the INPAR protocol, it is possible to achieve reliable transfer not only of single destination but also multidestination or Broadcast messages. In other words, a Reliable Broadcast Protocol has been designed for the Message Bus.

It was felt that a conventional design such as the above fell short on two major accounts. On one hand, certain important design choices could not be validated. For, example, it could not be ascertained that the priority based bus arbitration mechanism chosen would result in small delays and hence greater message throughput. Secondly, the number of buffers required in the Bus Interface could not be agreed upon in an ad-hoc manner, nor could the maximum number of retransmissions that were to be allowed. Thirdly, it was not possible to say whether a dual-port memory buffer would indeed be better than the conventional single port one. Finally, the INPAR protocol needed to be formally verified for correctness of operation in order to be fully confident of its design. On the other hand, only preliminary performance characteristics regarding throughput etc. on the bus could be predicted; it was not possible to predict performance in depth, since the exact effect of various factors could not be gauged quantitatively.

Consequently, it was necessary to use some formal methods for validating the design and performance of the proposed multiprocessor system. We chose Petrinets for this purpose since they allow a clear description of the concurrency, conflicts and synchronization of parallel processes. They also present a simple, yet elegant, formalism for the latter's description. Further, in order to be able to validate performance of the system, we chose a recent extension of Petrinets, ie., Timed Petrinets to model the system.

In this respect, we have shown that a certain class of Timed Petrinets (TPNs), with delays associated ^{with} places of the net, are ideally

suited for modelling the deterministic nature of real time process control systems like the Message Bus multiprocessor and that TPN theory provides the right tools for performance analysis.

The approach taken was to consider certain subsystems of the multiprocessor and to model and analyse each using Timed Petrinets, first separately and then together as a composite system. Three subsystems have been considered, namely, the Bus Contention and Arbitration system, the INPAR protocol and the Message Buffering system. Previous approaches to multiprocessor performance evaluation concentrated only on one subsystem. We believe that our approach will thus enable substantial progress to be made in the efficient design of Large Scale Systems.

The Bus Contention system TPN model has been analysed for maximum allowable Message Throughput and Bus Utilization, using processor priority as the basis for arbitration. The analysis has not only justified the choice of a priority based bus arbitration, but also has important implications for the manner in which priorities should be allocated to the processors in order to maximize throughput on the Bus. TPN analysis has been used in a novel application showing that by considering both the time taken by each processor to individually process a message and the time taken to send it on the Bus, its periodic Bus access interval can be minimized, provided a certain priority is given to it.

Further, using the fact that the throughput maximization problem is very closely related to problems of Dynamic Programming, we have written a simple program for the case of 16 processors on the Bus and presented a technique whereby priorities can be allocated for fair Bus Arbitration based on the above. This decentralized Bus Arbitration mechanism is seen to be easily implemented in the Bus Interface hardware. We believe that this results in an improved design for the Bus Interface hardware and is a significant step towards the goal of hardware realization of communication protocols using VLSI techniques.

The numerical values obtained from the above analysis have also been used to predict the performance of the Message Bus multiprocessor in terms of well-known measures such as Speed Up and Efficiency. The analysis is easily extended to multiple-bus multiprocessors.

The functional design characteristics of the INPAR protocol have been verified by doing a Formal Specification and Verification of its correctness of operation. This has been done using the TPN model of the protocol. The use of TPNs has led to the possibility of considering not only its property of Safety (correct delivery), but also of Liveness (progress without deadlock) and Performance (minimum delays). The language chosen for specification has been borrowed from the recently reported AFFIRM system.

We have presented a unique approach for doing a performance analysis of communication protocols by using their TPN models and by defining performance in terms of the time taken to complete certain operations of the protocol. Further, we have shown that even for complex protocols, it is possible to overcome the limitation regarding explosion of states in the Petrinet, by considering only a subset of these states, ie., those which actually affect the firing of its transitions and hence its performance analysis. Another important aspect that has been brought to light is that Eventuality of protocol operations and hence its Liveness properties can be easily specified using Timed Petrinets because of the restriction that ^{all} enabled transitions _(at least one in the case of conflict) must fire as soon as all input places have available tokens. This was not possible to do using State Machines and ordinary Petrinets.

The above mentioned performance analysis technique can be used on any typical protocol. This fact has been illustrated using the ECMA Standard Protocol (connection/disconnection phase).

The Message Buffering system between each Bus Interface and its host processor has also been modelled and analyzed to give a quantitative measure for Buffer requirements and holding times. Cases for sequential and simultaneous (Dual Port) memory access have been considered to see how they compare with respect to performance.

Finally, a composite TPN model of the multiprocessor has been studied to see how each subsystem affects the other's and also the overall multiprocessor performance. This has enabled us to find out quantitatively the effect of heterogeneous processors in the system, the number of maximum allowable retransmissions and the number of buffers, on the Throughput in the system. It has also strengthened our belief that TPN models can be easily extended to incorporate large

scale characteristics.

Thus we have illustrated how composite multiprocessor characteristics can be evaluated using techniques used for its subsystems. Since these techniques can be easily automated, they can be viewed as basic tools in the Computer Aided Design of Large Scale Systems.

7.2 DIRECTIONS FOR FURTHER RESEARCH WORK

7.2.1 Use of Merlin's Time Petrinets for Performance Analysis

We have mentioned in Chapter 1 that Sifakis's Timed Petrinets are one of several interpretations of associating 'time' with ordinary Petrinets. Merlin's Time Petrinets associate two time values a and b , where $0 \leq a \leq b$, with every transition of a petrinet. This is interpreted to mean that once the transition has been enabled, it may not fire before a time a and must fire before or at time b after being enabled. Such Time Petrinets have been the subject of much recent research. For example, in [Bertho83] they have been used to prove the correctness of time dependent protocols.

We have also mentioned in Chapter 3 that TPNs with constant firing times (single values) for transitions have been shown to be transformable and hence equivalent to TPNs with constant times associated to places [Sifaki78]. In a similar manner, it should be possible to transform Merlin's Time petrinets to timed petrinets in which two time values are associated to the places instead of to the transitions. Further it would be worthwhile to develop an enumerative state analysis theory for such Timed petrinets on the same lines as that for Sifakis's or Merlin's Petrinets.

Thus, given a Merlin TPN modelling some system, one could first convert it to an equivalent TPN in which two time values a and b are associated to each place, with a representing the minimum allowed delay at that place and b representing the maximum. From this, it should be possible to specify the minimum and maximum time taken to complete a cycle of operation using a modified Sifakis's theory.

Such an extension would be useful for systems in which the time delay at a place may vary between two limits. For example, in our bus contention model, the accessing times A of the processors could be

given as (Ae, Al) to represent the fact that the bus access time is Ae if no retransmissions are necessary for message transfer and Al if one or more (upto a maximum) retransmissions are required.

7.2.2 Inclusion of time in Predicate/Transition nets

To overcome the limitations of ordinary Petrinets, many interesting extensions, besides Timed petrinets, have been suggested. We have, for eg., Place-coloured (pl-cl) nets, Predicate/ Transition (Pr-Tr) nets, Predicate/Action (Pr-Ac) nets, etc.

Of these, the most promising are the Predicate/Transition nets, as they can be viewed as a concise abbreviation for large ordinary Petrinets [Genric79]. Places here stand for sets of ordinary Petrinet places and can contain tokens having n-tuples of parameters specified by constants or variables; similarly, transitions stand for sets of ordinary Petrinet transitions, arcs are labeled by n-tuples of constants or variables defining the pattern of tokens produced or consumed by a transition in a place, and finally for every transition, a predicate specifies relations between values of different tokens which are needed or produced by its firing.

Such Predicate/Transition nets can be expanded to obtain ordinary Petrinets so that an analysis may be done for them. Also, they have the advantage that tokens in a place carry both an identity as well as a relationship with those of other places. Thus they can be used very efficiently in, for example, aiding recovery from error by identification of a token and providing an order of emission or reception.

Of course such Pr/Tr nets do not yet have any timing associated to places or transitions, hence they cannot be used for performance evaluation of the systems they model, unlike Timed Petrinets.

We feel that if the time factor is incorporated into Predicate / Transition nets, they would provide ideal tools for the complete functional as well as performance analysis of systems like communication protocols. This is an interesting area for further research. At the same time it has to be kept in mind that this may not be very simple to do, as it is well known that extensions to Petrinets, while improving their modelling power, may actually adversely affect their decision power [Peters81].

7.2.3 Automated analysis of Timed Petrinets

As with most analytical theories, Sifakis's TPN analysis tends to get complicated and time consuming for large models. It becomes increasingly complex, for example, to write down the Incidence matrix for the net and to enumerate all its Current vectors.

To enable analysis of large ordinary petrinets, researchers have now developed software packages on computers. Notable among these are the OVIDE, Petripote and Petrinet Package [Aft83]. These packages handle the description and the validation of the nets they help to build and design. They provide for most of the functions listed below -

- net editing (creation/deletion/modification) through either a graphic net editor or a textual description of nets;
- enumerative analysis methods, by which a marking graph can be drawn and reachability analysis done for the net;
- property preserving reduction routines which reduce the complexity of the net while still preserving important properties such as Boundedness, Liveness or the Cyclic property;
- Structural analysis methods, by which functions are provided for extracting place- or transition- invariants from the nets.

We believe that development of a similar facility for the automated analysis of Timed Petrinets will go a long way in doing an efficient design and performance validation of Large Scale Systems (like distributed and multiprocessor systems) based on the techniques presented in the thesis.

For automated Timed Petrinet validation packages, the following functions will need to be provided in addition to those listed above -

- assignment/modification of time delays to places;
- creation/modification of subnets for J vectors;
- creation/modification of subnets for I vectors;
- computation of maximum firing rates in the net.

REFERENCES

- [AFCET83] Proc. 4th AFCET European Wksp. on Appln. and theory of Petrinets, Toulouse, France, Sept., 1983.
- [Agarwa83] A. K. Agarwal, ed., Special issue on multiprocessor performance evaluation, IEEE-Trans. Computers, vol.C-32, no. 1, Jan. 1983.
- [Agerwa79] T. Agerwala, 'Putting Petrinets to work', Computer, Dec. 1979, pp, 85-94.
- [Andler81] S. Andler et al, 'On enhancing local network communication devices', in Local Networks for Computer Communication, eds. A.West & P.Janson, NH Publ. Co., 1981, pp. 191-206.
- [AP28 79] Intel Multibus Interfacing, Application Note AP28A, 1979.
- [Bass80] C. Bass et al, 'Local network gives new flexibility to distributed Processing', Electronics/September 25,1980, pp 114-122.
- [Belang80] P. Belanger et al, 'Performance measurements of a local multicomputer network', in Local Networks for Computer Communication, A.West & P.Janson, NH Publishing Co, 1981, pp 181-190.
- [Berthe82] G. Berthelot & R. Terrat, 'Petrinet theory for correctness of protocols', IEEE Trans. Communications, vol.COM-30, Dec '82, no. 12, pp 2497-2505.
- [Bertho83] B.Berthomieu, 'An enumerative approach for analyzing Time Petrinets,' Proc. IFIP'83, Paris, Sept.83, pp 41-46.
- [Biscoe81] C.Tyne-dale Biscoe & G. R. Hobbs, 'EUROBUS, an interface standard for microcomputer systems', EUROMICRO 81, pp 181-190.
- [Bochma80] G.Bochmann & C. Sunshine, 'Formal methods in communication protocol design,' IEEE Trans. Communications, vol. COM-28, no.4, April 80, pp 624-631.
- [Borill82] P.Borill, Ed., 'Special issue on Backplane Bus standards,

- [Brinkm80] B. Brinkman, 'The DEMOS-86 Multimicrocomputer', Report, Scicon Consultancy Int. Ltd., London, 1980.
- [CD82] Computer Design / December 1982, Report, 'Factory environment fiber optic provides emi immunity, increases security' pp 32.
- [Chrett83] P. Chrettienne, 'Les Reseaux de Petri Temporises,' These d'etat, Universite Paris VI, June 1983.
- [Christ79] N. Christofides, 'COMPUTATIONAL OPTIMIZATION', John Wiley, 1979.
- [Chu80] W. W. Chu et al, 'Task allocation in distributed data processing' Computer, vol. 13, pp 57-69, Nov. 80.
- [Coolah83] J.E. Coolahan & N.Roussopoulos, 'Timing requirements for time driven systems using Augmented Petrinets', IEEE Trans Software Engg. vol SE-9, Sep 83, pp 603-616.
- [Crane81] R.C.Crane, 'Software pack and controller link DEC computers in an Ethernet', Eletronics/December 15,1981, pp 133-136.
- [Davies81] 'Distributed Systems: Architecture and Implementation - an advanced course', Lecture Notes in Computer Science,no.105 edited by Davies & Lampson, Springer Verlag, 1981.
- [Deiss81] S.R. Deiss et al, 'Applicability of FASTBUS standard to distributed control', IEEE Trans. Nucl. Sci., vol. NS-28, no.3, pp 2222-4, June 1981.
- [Diaz82] M. Diaz, 'Modelling and Analysis of communication and Co-operation protocols using Petrinet based models,' in 'Protocol Specification, Testing and Verification', ed. C. Sunshine, NH Pubs, 1982.
- [ECMA80] ECMA, 3rd draft of Transport Protocol, ECMA/TC24/80/16 Feb. 1980.

- [Estrin81] J. Estrin & B. Cavico, 'Local network enlists Z80s for Distributed Processing', Electronics / February 10, 1981, pp 149-153.
- [Garey79] M.R. Garey & D.H. Johnson, 'COMPUTERS AND INTRACTABILITY- A GUIDE TO THE THEORY OF NP-COMPLETENESS' Freeman SF, 1979.
- [Garg82] K. Garg, 'Hardware design of a station for DCCS', Internal Report, DOC Imperial College, Feb. 1982.
- [Garg83a] K. Garg, 'Formal specification and verification of the Message Bus Protocol', Internal Report, DOC, Imperial College, Feb. 1983.
- [Garg83b] K. Garg and M.S. Sloman, 'The Message Bus, an interconnection mechanism for a multicomputer, ' Research Report, Imperial College, DOC 83/8, March 1983, London.
- [Garg84a] K. Garg, 'An approach to performance specification of communication protocols using Timed Petrinets', accepted at the ^{IEEE} 4th Intl. Conf. on Distributed Computing Systems, San Francisco, May 1984.
- [Garg84b] K. Garg, 'An approach to multiprocessor performance analysis using Timed Petrinet models', accepted for the INRIA Intl. Conf. on Modelling Techniques & Tools for performance analysis, Paris, May 1984.
- [Genric78] H.J. Genrich & K. Lautenbach, 'The analysis of distributed systems by means of predicate/Transition nets,' in Lecture Notes in Computer Sc., vol. 70, ed. G. Kahn, NY Springer Verlag 1978.
- [Ghosh77] S. Ghosh, 'Some comments on timed Petrinets', Journee AFCET sur les Reseaux de Petri, Paris, May 1977.
- [Goldbe82] A. Goldberger et al, 'Small area networks fit jobs too small for local nets', Electronics / November 3, 1982, pp 119-122.
- [Guttag75] J.V. Guttag, 'The specification and application to program-

ming of abstract data types', Ph.d.thesis, Dept. Comp.Sc., Univ. of Toronto, Oct. 75.

- [IEEE81] IEEE 802 Local Network Standard, draft B, October 19, 1981.
- [ISO81] ISO/TC97/SC16/DP 7498, 'Basic Reference Model of OSI', Computer Networks, vol.5, no.2, April 81, pp 81-118.
- [Jensen78] E. D. Jensen, 'The Honeywell Experimental Distributed Processor - an overview', Computer, January 78, pp 28-38.
- [Kirman81] H.Kirmann, 'A serial interprocessor link for multiprocessor management in the P896 Backplane Bus', EUROMICRO 81, pp 171-180.
- [Knippe81] P.Knippel and K. Progran, 'Local networks and ISO's OSI Architecture', Local Networks for Computer Communication, eds. A.West & P.Janson, NH Publishing Co., 1981, pp 447.
- [Kramer83] J.Kramer et al, 'CONIC: an integrated approach to DCCS', IEE Proc., Pt. E, vol. 130, no. 1, Jan 1983, pp 1-10.
- [Lam82] S. Lam & A. U. Shankar, 'An illustration of Protocol projections in Protocol specification, Testing and Verification, ed., C.Sunshine, NH-Pubs., 1982, pp 343-360.
- [Lavenb83] S.S. Lavenberg, Ed., 'Computer Performance Modelling handbook, Academic Press, London 1983.
- [Ma82] R. P. Ma et al, 'A task allocation model for distributed computer systems', IEEE Trans. Computers, vol.C-31, no. 1, Jan. 1982, pp 41-47.
- [Marken82] P. Markenscoff, 'Analysis of a multiple processor system for real time applications', IEEE-82, Proc. of A symp. on Real Time Systems, Dec. '82, LA, Calif., pp 45-56.
- [Marsan83] M.A. Marsan et al, 'Modelling Bus contention and memory interference in a Multiprocessor system', IEEE Trans. Computers vol. C-32, no. 1, Jan '83, pp 60-71.

- [McQuill78] J.M.McQuillan, 'Enhanced message addressing capabilities for Computer Networks', Proc. IEEE, vol. 6, no. 11, November '78, pp 1517-1527.
- [Menasce83] M. Menasche & B.Berthomieu, 'Time Petrinets for analyzing and verifying and analyzing Time-dependent Communication Protocols', 3rd IFIP WG 6.1 Wksp.on Protocol Specification Testing & Verification, Ruschlikon, Switzerland, May-June, 1983.
- [Merlin74] P. M. Merlin, 'A study of the recoverability of Computer systems' Ph.d. Thesis, Comp. Sc., Dept. U.C. Irvine.
- [Merlin79] P.M. Merlin, 'Specification and Validation of protocols', IEEE Trans. Communications, COM-27, Nov.'79, pp 1671-1680.
- [Metcal76] R.M.Metcalfe & D.R.Boggs, 'Ethernet: Distributed packet switching for local computer networks', Comms. ACM, July 76, pp 395 - 404.
- [Mimica82] O.Mimica & B.Marsden, 'A datagram based network for micro-computers', Computer Communications, vol.5, no.3, June 82, pp 128-139.
- [Mitran83] I.Mitrani & P.J.B. King, 'Multiserver systems subject to breakdown', Corres. IEEE Trans. Computers, vol.C-32, no.1, Jan.83, pp 96-98.
- [Mockap76] P.Mockapetris et al, 'Design of network access arrangement' ICS, University of California at Irvine, TR no.95, Nov. 1976.
- [Molloy81] M.K. Molloy, 'On the integration of delay and throughput measures in Distributed Processing models', Ph.D. thesis, UCLA, 1981.
- [Motoro82] VME Bus Specification manual, Motorola.
- [Nader79] A.Nader, 'Petrinets for real time control algorithm decomposition', IFAC Wksp. on DCCS, Florida, 1979.
- [Osborn79] 'The IEEE GPIB 488', in Introduction To Microcomputers,

- [Ozsu82] M.T. Ozsu & B.V. Weide, 'Modelling of Distributed Data-base Concurrency control mechanisms using an Extended Petrinet formalism', Proc. 3rd Int. Conf. on DCS, Miami, Fla., Oct. 1982, pp 660-665.
- [Patzel82] R. Patzelt, 'Microprocessor systems; Multiprocessor/ Multi-task Bus system requirements', in EDISG minutes of 17th meeting, Brussels, June 2-4, 1982.
- [Peters81] J. L. Peterson, 'PETRINET THEORY AND THE MODELLING OF SYSTEMS', 1981, Prentice-Hall.
- [Piment81] J.R. Pimental, 'Performance of a multiple microcomputer system for distributed processing in control application', pp 179-185.
- [Price81] C.C. Price, 'The assignment of computational tasks among processors in a distributed system', in Proc. Nat. Comput. Conf. May 1981, pp 291-296.
- [Prince81] S. Prince & M.S. Sloman, 'Communication Requirements of a DCCS', Proc. IEE vol. 128, Pt. E, No. 1, Jan. 81, pp 21-34.
- [Ramamo80] C.V. Ramamoorthy & G.S. Ho, 'Performance evaluation of asynchronous concurrent systems using Petrinets', IEEE Trans. Soft. engg., Vol SE-6, no. 5, Sept. 80, pp 440-449.
- [Ramcha73] C. Ramchandani, 'Analysis of asynchronous concurrent systems by Timed PetriNets', Ph.D. thesis, MIT, Sept. 1973.
- [Reisig83] W. Reisig, 'System design using Petrinets', Proc. 4th Europ. Wksp. on Application and Theory of Petrinets, Toulouse, France, Sept. '83, pp 309-321.
- [Sauer81] C.H. Sauer and K.M. Chandy, 'COMPUTER SYSTEMS PERFORMANCE MODELLING', Prentice Hall, NJ. 1981.
- [Schwar82] R. Schwartz & P.M. Melliar-Smith, 'From state machines to temporal logic : specification methods for protocol

- standards, in Proc. 2nd Intl. Wksp. on Protocol Specification, Testing and Verification, ed. C.Sunshine, NH-1982, pp 3-20.
- [Sifaki77] J. Sifakis, 'Petrinets for performance evaluation', Intl. Symp.Measuring, Modelling and Evaluating Computer Systems, 1977, pp 75-93.
- [Sifaki80] J.Sifakis, 'Performance Evaluation of systems using nets', Lecture Notes in Comp. Sc., no. 84, ed. W. Brauer, 'Net theory & its Applications', Springer Verlag NY-1980, pp 307-320.
- [Shanka82] A.U. Shankar & S. Lam, 'On time-dependent communication Protocols and their Projections', in Proc. 2nd Intl. Wksp. Protocol Specification, testing and verification, ed. C. Sunshine, NH-Pub.1982, pp 215-236.
- [Sloman82] MS.Sloman, 'The CONIC communication system for distributed process control, ' Research Report DOC 82/8, Imperial College, June 1982.
- [Sommer76] R.Sommer, 'COBUS, a firmware controlled data transmission system' EUROMICRO 1976.
- [Spang76] H.A.Spang, 'Distributed Computer Systems for control,' 1st IFAC - IFIP Symposium on Software for Computer Control, SOCOCO 1976, pp 47-65.
- [Sunshi82a] C.A.Sunshine, 'Formal modeling of communication protocols' in Computer networks and simulation II, ed. S. Schoemaker, NH- Pubs., 1982, pp 53-78.
- [Sunshi82b] C. A. Sunshine et al, 'Specification and Verification of communication protocols in AFFIRM using State Transition Models', IEEE Trans. Soft. Engg. vol.SE-8, no.5,Sep.82, pp 460-489.
- [Taha81] H.A.Taha, 'OPERATIONS RESEARCH - AN INTRODUCTION', 3rd Ed. 1981, Macmillan Pub. co., pp 361-62.

- [Weitzm82] C.Weitzman, 'Performance measures for Distributed Computer systems', Proc. 3rd. Intl. conf. on DCS, Miami, Fla., Oct. 1982, pp 705-709.
- [Wensel78] Wensely et al, 'SIFT: Design and analysis of a Fault-tolerant computer for aircraft control', Proc.IEEE, vol.66, no. 10, Oct. 1978, pp 1240-1268.
- [Yemini82] Y. Yemini & J. Kurose, 'Towards the Unification of the functional and performance analysis of protocols, or, Is the Alternating Bit protocol really correct?' in Protocol Specification, Testing & Verification, ed., C.Sunshine, NH Pubs., 1982, pp 189-196.
- [Zubere80] W.M.Zuberek, 'Timed Petrinets and preliminary performance evaluation', in Proc.7th Annual Symp.Computer Arch., 1980, pp 88-96.

APPENDIX I KNAPSACK PROGRAM FOR A 16-PROCESSOR SYSTEM

```

        DIMENSION IF(15,45),IM(15,45)
        DIMENSION IA(15), IP(15)
C      VALUES OF ACCESSING AND PROCESSING TIMES
        READ (5,10) (IA(K), K=1,15)
        READ (5,10) (IP(K), K=1,15)
10     FORMAT (15I3)
        READ (5,11) IAZERO, IPZERO
11     FORMAT (2I3)
C      12 GIVES LIMIT FOR M
        MSUM = IAZERO
        DO 12 K=1,15
            IF(K,1)=0
12     MSUM=MSUM+IA(K)
            ICZERO=IPZERO+IAZERO
            IF(MSUM-ICZERO)30,31,31
30     LIMIT=ICZERO-MSUM
31     LIMIT=20
        DO 13 N=1, LIMIT
C      13 GIVES VALUES OF M1 AND F1(M)
            IYONE = N/IA(1)
            IYTWO = (N+MSUM)/(IP(1)+IA(1)) - 1
            M=N+1
            IM (1,M) = MINO (IYONE, IYTWO)
13     IF(1,M)=IM(1,M)
C      MKMAX IS MAXIMUM VALUE FOR MK
            DO 15 K=2,15
                DO 15 N=1,LIMIT
                    LONE = N/IA(K)
                    LTWO = (N+MSUM)/(IP(K)+IA(K)) - 1
                    MKMAX = MINO(LONE, LTWO)
                    MK = 0
                    IG = 0
16     IX = N - MK*IA(K)
                    IH = MK + IF((K-1),(IX+1))
                    IF(IH.LE.IG) GO TO 20
                    IG = IH
                    MKK = MK
C      MKK IS VALUE OF MK FOR WHICH FK IS MAX
20     MK = MK + 1
                    IF (MK.LE.MKMAX) GO TO 16
                    M=N+1
                    IF(K,M) = IG
C      IF(K,M) IS MAXIMUM VALUE OF FK(M)
15     IM(K,M) = MKK
                    WRITE(6,100)(K,(IF(K,M),IM(K,M),M=2,LIMIT+1), K=1,15)
100    FORMAT ((5HOK = ,I2/8(2I3,2X)/8(2I3,2X)/8(2I3,2X)/
X8(2I3,2X)/8(2I3,2X)/))
        STOP
        END

```

APPENDIX IIa

Timed Petrinetmachine MBProtocol;

declare s:MBProtocol;

declare m:Message;

constructors

InitializeProtocol, ProtocolSend(s,m), ReceiveRecognize(s), Deliver(s),
ReceiveByte(s), Retransmit(s), LosePacket(s), LoseByte(s), GiveUp(s),
ReleaseBus(s);

selectors ACK, NAK1, NAK2, : Byte;

selectors Sent(s), Received(s): QueueOfMessage;

selectors Pending(s), ReceiverBuffer(s): QueueOfPacket;

selector SenderToReceiver(s): SMedium;

selector ReceiverToSender(s): RMedium;

selectors RPI(s), SPI(s), Retry(s), RBA(s): Counter;

selector Timeelapsed(s) : integer;

selector Marking(s) : Register;

axioms {InitializeProtocol}

Pending(InitializeProtocol) == NewQueueOfPacket,	...i
Received(InitializeProtocol) == NewQueueOfMessage,	...ii
Sent(InitializeProtocol) == NewQueueOfMessage,	...iii
ReceiverBuffer(InitializeProtocol) == NewQueueOfPacket,	...iv
SenderToReceiver(InitializeProtocol) == InitializeSMedium,	...v
ReceiverToSender(InitializeProtocol) == InitializeRMedium,	...vi
RPI(InitializeProtocol) == NullCounter,	...vii
SPI(InitializeProtocol) == NullCounter,	...viii
RBA(InitializeProtocol) == Initialize(RBA),	...ix
Retry(InitializeProtocol) == Initialize(Retry),	...x
Timeelapsed(InitializeProtocol) == 0,	...xi
Marking(InitializeProtocol) == Reading(1);	...xii

axioms {ProtocolSend}

Marking(ProtocolSend(s,m)) == if Marking(s)=Reading(1)	
then Reading(2,5,6)	
else Marking(s),	xiii
Pending(ProtocolSend(s,m)) == if Marking(s)=Reading(1)	
then NewQueueOfPacket Add MakePacket	
(m,SPI(s),ComputeCRC(m))	

```

else Pending(s), ...xiv
SenderToReceiver(ProtocolSend(s,m)) == if Marking(s)=Reading(1)
then Transmit(SenderToReceiver(s),
MakePacket(m,SPI(s),ComputeCRC
(m)))
else SenderToReceiver(s), ...xv
Sent(ProtocolSend(s,m)) == if Marking(s)=Reading(1)
then Sent(s) Add m
else Sent(s), ...xvi
Retry(ProtocolSend(s,m)) == if Marking(s)=Reading(1)
then Initialize(Retry)
else Retry(s), ...xvii
Timeelapsed(ProtocolSend(s,m)) == if Marking(s)=Reading(1)
then Timeelapsed(s)+Timein(User)
else Timeelapsed(s); ...xviii

```

axioms {ReceiveRecognize}

```

Marking(ReceiveRecognize(s)) == if Marking(s)=Reading(2,5)
then Reading(3,4,5)
else if Marking(s)=Reading(2,5,6)
then Reading(3,4,5,6)
else Marking(s), ...xix
SenderToReceiver(ReceiveRecognize(s)) == if Marking(s)=Reading(2,5) or
Reading(2,5,6)
then Receive(SenderToReceiver(s))
else SenderToReceiver(s) ...xx
ReceiverBuffer(ReceiveRecognize(s)) == if Marking(s)=Reading(2,5) or
Reading(2,5,6)
then NewQueueOfPacket Add Front(
SenderToReceiver(s))
else ReceiverBuffer(s), ...xxi
ReceiverToSender(ReceiveRecognize(s)) == if Marking(s)=Reading(2,5)
or Reading(2,5,6)
then if RBA(s)=NullCounter
then Transmit(ReceiverToSender(s),NAK1)
else if CRC(Front(SenderToReceiver(s)))
= ComputeCRC(Text(Front(
SenderToReceiver(s))))
then Transmit(ReceiverToSender(s),NAK2)
else Transmit(ReceiverToSender(s),ACK)
else ReceiverToSender(s), ...xxii

```

```

Timeelapsed(ReceiveRecognize(s))==if Marking(s)=Reading(2,5) or
                                Reading(2,5,6)
                                then Timeelapsed(s) +
                                Timein(SenderToRececeiver)
                                else Timeelapsed(s);    ..xxiii

```

axioms {Deliver}

```

Marking(Deliver(s)) == if Marking(s)=Reading(3,4,5)
                        then Reading(4,5)
                        else if Marking(s)=Reading(3,4,5,6)
                        then Reading(4,5,6)
                        else Marking(s),                ...xxiv
RBA(Deliver(s)) == if Marking(s)=Reading(3,4,5) or Reading(3,4,5,6)
                    and ReceiverToSender(s)=Transmit(ReceiverTo-
                                                Sender(s),Ack)
                    and RPI(s)=NullCounter
                    then Decrement(RBA(s))
                    else RBA(s),                        ...xxv
Received(Deliver(s))==if Marking(s)=Reading(3,4,5) or Reading(3,4,5,6)
                        and ReceiverToSender(s)=Transmit(Receiver-
                                                ToSender(s),Ack)
                        and RPI(s)=NullCounter
                        then Received(s) Add Text(Front(ReceiverBuffer(s)))
                        else Received(s),                ...xxvi
RPI(Deliver(s)) == if RPI(s)=NullCounter
                    then Set(RPI(s),CurrentCount(SPI(s)))
                    else RPI(s),                        ...xxvii
ReceiverBuffer(Deliver(s)) == NewQueueOfPacket;        ...xxviii

```

axioms {ReceiveByte}

```

Marking(ReceiveByte(s))==if ReceiverToSender(s)=Transmit(Receiver-
                                                ToSender(s),(NAK1 or NAK2      ))
                        then if Marking(s)=Reading(3,4,5,6)
                        then Reading(5,7,6)
                        else if Marking(s)=Reading(3,4,5)
                        then Reading(5,7)
                        else if ReceiverToSender(s)=Transmit(
                                                ReceiverToSender(s),Ack)
                        then if Marking(s)=Reading(4,5,6)
                        then Reading(5,8,6)

```

```

else if Marking(s)=Reading(4,5)
    then Reading(5,8)
else Retry(s), ...xxix
ReceiverToSender(ReceiveByte(s)) == Receive(ReceiverToSender(s),
Timeelapsed(ReceiveByte(s))==if ReceiverToSender(s) =InitializeRMedium
    then Timeelapsed(s)+Timein(ReceiverTosender)
    else Timeelapsed(s); ...xxxi

axioms {Retransmit}
Marking(Retransmit(s))==if Marking(s)=Reading(5,6,7)
    then if Decrement(Retry(s)) =NullCounter
        then Reading(2,5,6)
        else Reading(2,5)
    else Marking(s), ...xxxii
SenderToReceiver(Retransmit(s))== if Marking(s)=Reading(5,6,7)
    then Transmit(SenderToReceiver(s),
        Front(Pending(s)))
    else SenderToReceiver(s), ...xxxiii

axioms {LosePacket}
Marking(LosePacket(s))==if Marking(s)=Reading(2,5,6)
    then Reading(4,5,6)
    else if Marking(s)=Reading(2,5)
        then Reading(4,5)
        else Marking(s), ...xxxiv
SenderToReceiver(LosePacket(s)) == if Marking(s)=Reading(2,5,6) or
    Reading(2,5)
    then Receive(SenderToReceiver(s))
    else SenderToReceiver(s),
ReceiverTosender(LosePacket(s))==if Marking(s)=Reading(2,5,6) or
    Reading(2,5)
    then Transmit(ReceiverToSender(s),NAK3)
    else ReceiverToSender(s), ...xxxvi
Timeelapsed(LosePacket(s))==if Marking(s)=Reading(2,5,6)orReading(2,5)
    then Timeelapsed(s)+Timein(SenderToReceiver)
    else Timeelapsed(s); ...xxxvii

axioms {LoseByte}
Marking(LoseByte(s))==if Marking(s)=Reading(3,4,5,6)
    then Reading(3,5,6,7)
    else if Marking(s)=Reading(3,4,5)
        then Reading(3,5,7)

```

```

        else Marking(s),          ...xxxviii
ReceiverToSender(LoseByte(s)) == if Marking(s) = Reading(3,4,5) or
                                Reading(3,4,5,6)
                                then Receive(ReceiverToSender(s))
                                else ReceiverToSender(s), ...xxxvix
Timeelapsed(LoseByte(s)) == if Marking(s) = Reading(3,4,5) or
                                Reading(3,4,5,6)
                                then Timeelapsed(s) + Timein(ReceiverToSender)
                                else Timeelapsed(s); ...xl

axioms {GiveUp}
    Marking(GiveUp(s)) == if Marking(s) = Reading(5,7) or Reading(3,5,7)
                        then Reading(5,8)
                        else Marking(s),          ...xli
    Sent(GiveUp(s)) == if Marking(s) = Reading(3,5,7) or Reading(5,7)
                        then Delete(Sent(s))
                        else Sent(s);              ...xlii

axioms {ReleaseBus}
    Marking(ReleaseBus(s)) == if Marking(s) = Reading(5,8) or Reading(5,8,6)
                        then Reading(1)
                        else Marking(s),          ...xliii
    RPI(RelaeaseBus(s)) == if Marking(s) = Reading(5,8) or Reading(5,6,8)
                        then Clear(RPI(s))
                        else RPI(s),              ...xliv
    SPI(RelaeaseBus(s)) == if Marking(s) = Reading(5,8) or Reading(5,6,8)
                        then Clear(SPI(s))
                        else SPI(s),              ....xlv
    Pending(ReleaseBus(s)) == if Marking(s) = Reading(5,8) or Reading(5,6,8)
                        then NewQueueOfPacket
                        else Pending(s),          ...xlvi
    Retry(ReleaseBus(s)) == if Marking(s) = Reading(5,8) or Reading(5,6,8)
                        then Clear(Retry(s))
                        else Retry(s);            ...xlvii

end {MBProtocol};

type Message;
declare m:Message;
axiom m = m == TRUE;
end {Message};

type CRCofMessage;
needs type Message;

```

```

declare m:Message;
interface ComputeCRC(m):CRCofMessage;
end {CRCofMessage};

type Counter;
needs type Integer;
declare c,c1,c2:Counter;
declare v,v1,v2:Integer;
interfaces Set(c,v),NullCounter,Increment(c),Decrement(c),Clear(c),
            Initialize(c): Counter;
interfaces InitialCount(c),CurrentCount(c),MaxCount(c):Integer;
axioms Increment(NullCounter)==Set(c,1),
        Decrement(NullCounter)==Set(c,MaxCount(c)),
        CurrentCount(NullCounter)==0,
        MaxCount(NullCounter) == 0;
axioms Increment(Set(c,v1)) == Set(c,v1+1),
        Increment(Set(c,MaxCount(c)) == NullCounter,
        Decrement(Set(c,v1)) == Set(c,v1-1),
        Clear(Set(c,v1)) == NullCounter,
        CurrentCount(Set(c,v1)) == v1;
axioms Clear(c) == NullCounter,
        InitialCount(c) ==  $\leq$  MaxCount(c),
        CurrentCount(c) ==  $\leq$  MaxCount(c);
end {Counter};

type SMedium;
needs type Packet;
declare s,s1,s2:SMedium;
declare p,p1,p2:Packet;
interfaces InitializeSMedium,Transmit(s,p),Receive(s):SMedium;
interface Front(s):Packet;
interfaces Empty(s),p in s:Boolean;
infix in;
axioms Receive(InitializeSMedium) == InitializeSMedium,
        Receive(Transmit(s,p) == if s=InitializeSMedium
                                then InitializeSMedium
                                else Transmit(Receive(s),p);
axiom Front(Transmit(s,p)) == if s=InitializeSMedium
                                then p
                                else Front(s);
axiom Empty(s) == (s=InitializeSMedium);

```

```

axiom Timein(s) == z2;
axioms p in InitializeSMedium == FALSE,
           p in Transmit(s,p) == ((p=p1) or p in s);
distinct InitializeSMedium, Transmit;
end {SMedium};

type RMedium; {Similar to SMedium except that Packet is to be replaced by
                by Byte and the following change in Timein}
axiom Timein(s) == z3;
end {RMedium};

type Packet;
needs types Message, Counter, CRCofMessage;
declare p:Packet;
declare c,: Counter;
declare m,: Message;
declare cr: CRCofMessage;
interfaces MakePacket(m,c,cr), NullPacket, Corrupt(p): Packet;
interface Text(p): Message;
interface Id(p): Counter;
interface CRC(p): CRCofMessage;
axioms Text(MakePacket(m,c,cr) == m,
            Text(Corrupt(MakePacket(m,c,cr))) == ~m;
axiom Id(MakePacket(m,c,cr) == c;
axiom CRC(MakePacket(m,c,cr) == cr;
distinct MakePacket, NullPacket;
end {Packet};

type Byte; {an enumerated type with 3distinct values}
interfaces NAK1, NAK2, ACK: Byte;
distinct NAK1,NAK2, ACK;
end {Byte};

type QueueOfPacket;
needs type Packet;
declare i,i1:Packet;
declare q,: QueueOfPacket:
interfaces NewQueueOfPacket, q Add i, Remove(q), Delete(q):QueueOfPacket;
infix Add;
interfaces Front(q), Back(q): Packet;
interface i in q: Boolean:

```

```

infix in;
interface Text(q): QueueOfMessage;
axioms Remove(NewQueueOfPacket) == NewQueueOfPacket,
        Remove(q Add i) == if q=NewQueueOfPacket
                            then q
                            else Remove(q) Add i;
axioms Delete(NewQueueOfPacket) == NewQueueOfPacket,
        Delete(q Add i) == q;
axiom Front(q Add i) == if q=NewQueueOfPacket
                        then i
                        else Front(q);
axiom Back(q Add i) == i;
axioms i in NewQueueOfPacket == FALSE,
        i in (q Add i1) == (i in q or (i=i1));
axioms Text(NewQueueOfPacket) == NewQueueOfMessage,
        Text(q Add i) == Text(q) Add Text(i);
distinct Add, NewQueueOfPacket,
        Remove(q), Delete(q);
end {QueueOfPacket};

type QueueOfMessage;
needs type Message;
declare q,q1: QueueOfMessage;
declare m,m1:Message;
interfaces NewQueueOfMessage,q Add m,Remove(q),Delete(q):QueueOfMessage;
infix Add;
interfaces Front(q),Back(q):Message;
interface m in q:Boolean;
infix in;
axioms Remove(NewQueueOfMessage) == NewQueueOfMessage,
        Remove(q Add m) == if q= NewQueueOfMessage
                            then q
                            else Remove(q) Add m;
axioms Delete(NewQueueOfMessage) == NewQueueOfMessage,
        Delete(q Add m) == q;
axiom Front(q Add m) == if q = NewQueueOfMessage
                        then m
                        else Front(q);
axiom Back(q Add m) == m;
axioms m in NewQueueOfMessage == FALSE,
        m in (q Add m1) == (m in q or (m=m1));

```

```

distinct Add, NewQueueOfMessage,
           Remove(q),Delete(q);
end {QueueOfMessage};

```

```

type Bit;
declare b,b1,b2:bit;
interfaces ON, OFF, b1,:bit;
axiom b==b;
axioms b1= $\sim$ b2==b1 $\sim$ =b2,
         $\sim$ b1=b2==b1 $\sim$ =b2;
distinct ON,OFF;
end {Bit};

```

```

type Register;
needs type bit;
needs type integer;
declare i1,i2,i3,i:integer;
declare r,r(i1),r(i1,i2):register;
interface length(r):integer;
interface NullRegister,Clear(r),Reading(1,2,..length(r)):register;
interface Test(r(i)): bit;
axiom Test(Reading(i1,i2)(i1))==ON;
Axiom Test(Reading(i1,i2)(i3))==OFF;
axiom Clear(Reading(2,3))==NullRegister;
axiom Reading(3,4)==Reading(4,3);
distinct Reading(1), Reading(1,6);
end {Register};

```

APPENDIX IIb.

1. The formal correspondence between the Service and Protocol

```
declare s:INPAR Protocol;
declare m:Message;

interface rep(s):INPARProtService;
1. InitializeService == rep(InitializeProtocol)
2. UserSend(rep(s),m) == rep(ProtocolSend(s,m))
3. AddressRecognize(s) == rep(ReceiveRecognize(s))
4. Userreceive(s) == rep(Deliver(s))
5. UserRelease(s) == rep(ReleaseBus(s))
6. Failed(s) == rep(GiveUp(s))
7. Sentservice(rep(s)) == Sentprotocol(s)

8. Receivedservice(rep(s)) == Receivedprotocol(s)

9. Buffer(rep(s)) == Text(Front(Pending(s)))
10. State(rep(s)) == if InS1(s)
                        then Idle
                        else if InS2(s)
                            then Arbitrating
                            else if InReading(1)
                                then ReadyToSend
                                else if InReading(2,5,6)
                                    then Sending
                                    else if InReading(3,4,5,6)
                                        then ReadyToReceive
                                        else TransactionOver
11. DelayRecognize(rep(s))==rep({LosePacket,Retransmit}*(s))
12. DelayReceive(rep(s)) == rep({ReceiveByte,LoseByte,Retransmit}*(s))
```

(Note: the * denotes repetition upto n times)

2. Correspondence of states between service and Protocol

```
InReading(1) {ReadyToSend}
== ( Pending(s)=NewQueueOfPacket
    and ReceiverBuffer(s)=NewQueueOfPacket
    and Sent(s)=Received(s)
```

```

and InitializeSMedium
and InitializeRMedium
and Initialize(Retry(s))
and Clear(RPI(s))
and Count(RBA(s))=CurrentCount(RBA(s))
and Count(SPI(s))=CurrentCount(SPI(s))

```

InReading(2,5,6) {Sending}

```

== ( Pending(s) ~ =NewQueueOfPacket
and ReceiverBuffer(s)=NewQueueOfPacket
and Sent(s)=Received(s) Add Text(Front(Pending(s)))
and PktCurrent(s,SenderToReceiver(s))
and InitializeRMedium
and Count(Retry(s))=CurrentCount(Retry(s))
and RPI(s)=NullCounter
and Count(RBA(s))=CurrentCount(RBA(s))
and Count(SPI(s))=CurrentCount(SPI(s))

```

InReading(3,4,5,6) {ReadyToReceive}

```

== ( Pending(s) ~ =NewQueueOfPacket
and ReceiverBuffer=Pending(s)
and Sent(s)=Received(s) Add Text(Front(Pending(s)))
and PktCurrent(s,ReceiverToSender(s))
and InitializeSMedium
and Count(Retry(s))=CurrentCount(Retry(s))
and RPI(s)=NullCounter
and Count(RBA(s))=CurrentCount(RBA(s))
and Count(SPI(s))=CurrentCount(SPI(s))

```

InReading(5,8) {TransationOver}

```

== ( Pending(s) ~ =NewQueueOfPacket
and ReceiverBuffer(s)=NewQueueOfPacket
and Sent(s)=Received(s)
and InitializeSMedium
and InitializeRMedium
and Count(Retry(s))=CurrentCount(Retry(s))
and RPI(s)=SPI(s)
and Increment(SPI(s))
and Decrement(RBA(s))

```

APPENDIX IIc.

The Theorems

1. Effects on state Variables by user operations

{for the InitializeProtocol operation}

theorem 1, InS1(InitializeProtocol);

theorem 2, Text(Front(Pending(InitializeProtocol)))=NewQueueOfMessage;

theorem 3, Sent(InitializeProtocol)=NewQueueOfMessage;

theorem 4, Received(InitializeProtocol)=NewQueueOfMessage;

{for the Send operation}

theorem 5, Text(Front(Pending(ProtocolSend(s,m))))=if InReading(1)
then Text(Front
(Pending(s))) Add m
else Text(Front(Pending(s)));

theorem 6, Sent(ProtocolSend(s,m))=if InReading(1)
then Sent(s) Add m
else Sent(s);

{for the Receive operation}

theorem 7, Received(Deliver(s)) = if InReading(3,4,5,6)
then Received(s) Add Front(Text
(Pending(s)))
else Received(s);

{for the GiveUp operation}

theorem 8, Sent(GiveUp(s)) = if InReading(2,5,6) or InReading(3,4,5,6)
then Delete(Sent(s))
else Sent(s);

{for the Release operation}

theorem 9, Text(Front(Pending(ReleaseBus(s))))=if InReading(5,8)
then Remove(Text(Front
(Pending(s))))
else Text(Front(
Pending(s)));

2. Effects on state variables by spontaneous operations

{for the Sent state variable}

theorem 10, Sent(LosePacket(s))=Sent(s)

and Sent(Retransmit(s))=Sent(s)
 and Sent(ReceiveByte(s))=Sent(s)
 and Sent(LoseByte(s))=Sent(s);

{for the Received state variable}

theorem 11, Received(LosePacket(s))=Received(s)
 and Received(Retransmit(s))=Received(s)
 and Received(ReceiveByte(s))=Received(s)
 and Received(LoseByte(s))=Received(s);

{for the Buffer state variable}

theorem 12, Pending(LosePacket(s))=Pending(s)
 and Pending(Retransmit(s))=Pending(s)
 and Pending(ReceiveByte(s))=Pending(s)
 and Pending(LoseByte(s))=Pending(s);

3. The next state transitions for all operations

{for the ProtocolSend operation}

theorem13, {move from Reading(1) to Reading(2,5,6)}
 InReading(1) imp InReading(2,5,6)(ProtocolSend(s,m);
theorem14, {no change}
 InReading(2,5,6) imp InReading(2,5,6)(ProtocolSend(s,m);
theorem15, {no change}
 InReading(3,4,5,6) imp InReading(3,4,5,6)(ProtocolSend(s,m);
theorem16, {no Change}
 InReading(5,8) imp InReading(5,8)(ProtocolSend(s,m);

{for the ReceiveRecognize operation}

theorem17, {no change}
 InReading(1) imp InReading(1)(ReceiveRecognize(s);
theorem18, {move from Reading(2,5,6) to Reading(3,4,5,6)}
 InReading(2,5,6) imp InReading(3,4,5,6)(ReceiveRecognize(s);
theorem19, {no change}
 InReading(3,4,5,6) imp InReading(3,4,5,6)(ReceiveRecognize(s);
theorem20, {no change}
 InReading(5,8) imp InReading(5,8)(ReceiveRecognize(s);

{for the Deliver Operation}

theorem21, {no change} InReading(1) imp InReading(1)(Deliver(s);
theorem22, {no change} InReading(1) imp InReading(1)(Deliver(s);

theorem23, {move from Reading(3,4,5,6) to Reading(5,8)}
 InReading(3,4,5,6) imp InReading(5,8)Deliver(s);
theorem24, {no change} InReading(5,8) imp InReading(5,8)(Deliver(s);

 {for the GiveUp operation}
theorem25, {no change} InReading(1) imp InReading(1)(GiveUp(s);
theorem26, {move from Reading(3,4,5,6) to Reading(5,8)}
 InReading(2,5,6) imp InReading(5,8)(GiveUp(s);
theorem27, {move from Reading(3,4,5,6) to Reading(5,8)}
 InReading(3,4,5,6) imp InReading(5,8)(GiveUp(s);
theorem28, {no change} InReading(5,8) imp InReading(5,8)(GiveUp(s);

 {for the ReleaseBus operation}
theorem29, {no change} InReading(1) imp InReading(1)(ReleaseBus(s);
theorem30, {no change}
 InReading(2,5,6) imp InReading(2,5,6)(ReleaseBus(s);
theorem31, {no change}
 InReading(3,4,5,6) imp InReading(3,4,5,6)(ReleaseBus(s);
theorem32, {move from Reading(5,8) to S1}
 InReading(5,8) imp InS1(ReleaseBus(s);

 {for the LosePacket operation}
theorem33, {no change} InReading(1) imp InReading(1)(LosePacket(s);
theorem34, {no change}
 InReading(2,5,6) imp InReading(2,5,6)(LosePacket(s);
theorem35, {no change}
 InReading(3,4,5,6) imp InReading(3,4,5,6)(LosePacket(s);
theorem36, {no change} InReading(5,8) imp InReading(5,8)(Losepacket(s);

 {for the ReceiveByte opertion}
theorem37, {no change} InReading(1) imp InReading(1)(ReceiveByte(s);
theorem38, {no change}
 InReading(2,5,6) imp InReading(2,5,6)(ReceiveByte(s);
theorem39, {no change or move from Reading(3,4,5,6) to Reading(5,8)}
 InReading(3,4,5,6) imp
 InReading(3,4,5,6)(ReceiveByte(s) or InReading(5,8)(ReceiveByte(s);
theorem40, {no change} InReading(5,8) imp InReading(5,8)(ReceiveByte(s);

 {for the Retransmit operation}
theorem41, {no change} InReading(1) imp InReading(1)(Retransmit(s);
theorem42, {no change} InReading(2,5,6)

```

        imp InReading(2,5,6)(Retransmit(s);
theorem43, {no change}
        InReading(3,4,5,6) imp InReading(3,4,5,6)(Retransmit(s);
theorem44, {no change} InReading(5,8) imp InReading(5,8)(Retransmit(s);

{for the LoseByte operation}
theorem45, {no change} InReading(1) imp InReading(1)(LoseByte(s);
theorem46, {no change} InReading(2,5,6) imp InReading(2,5,6)(LoseByte(s);
theorem47, {no change}
        InReading(3,4,5,6) imp InReading(3,4,5,6)(LoseByte(s);
theorem48, {no change} InReading(5,8) imp InReading(5,8)(LoseByte(s);

```