Specification and Verification of a Self-Timed Token Ring Protocol

A. Semenov[a*]   A. Yakovlev[a†]   N. Anisimov[b‡]

[a] Department of Computing Science,
University of Newcastle upon Tyne NE1 7RU, England
fax: +44-91-2228232; phone: +44-91-2226053; e-mail:alex.semenov@newcastle.ac.uk
[b]Institute for Automation and Control Processes
Far East Division of the Russian Academy of Sciences
Radio Street 5, Vladivostok 690041, Russia

Technical Report No. 516

Abstract

A reliable self-timed channel, based on a pipeline ring architecture, uses a special-purpose protocol with prioritised token access. The previous design of a channel adaptor was implemented by speed-independent circuits but without a strict proof of formal correctness of the protocol. Now we are planning to redesign the interface by means of recently developed net-based methods and software tools. This paper presents a compositional technique to construct a labelled Petri net model of this protocol and describes the results of its automatic verification using Pr/T nets.

Keywords: self-timed circuits, MAC protocol, token ring, pipeline, compositionality.

1 Introduction

Asynchronous circuits are known to be inherently robust to parametric faults and self-diagnostic for stuck-at faults. A circuit would normally indicate a stuck-at fault by halting at some specified state, and this can be registered through a simple hardware-controlled time-out mechanism. Such circuits are inherently modular and signal transparent – the interacting modules produce explicit acknowledgement ("ack") or non-acknowledgement ("nack") signals. They can be good candidates for reliable implementation of communication protocols, which are often asynchronous by their nature.

A few years ago, one of us was involved in the developement of a fault-tolerant asynchronous pipeline ring interface for an airborne computing system [12] (a safety-critical application). A communication medium was designed to tolerate up to two faults in any segment of the ring, thereby demonstrating the advantages of asynchronous design approach. The choice of a ring architecture, as opposed to a shared bus, was caused by the combination of functional requirements, e.g., flexibility of addressing – "selective broadcasting", distributed arbitration, minimal channel wiring, as well as by the following inherent reliability issue.

*Supported by Newcastle University Research Studentship and ORS award from CVCP.
†Partially supported by EPSRC GR/J52327.
‡Supported by the Russian Basic Research Fund (Grant No. 93-013-17372).

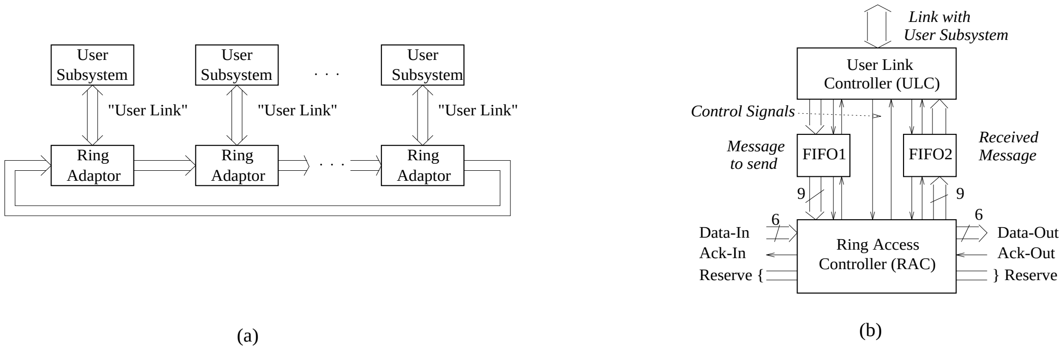

Figure 1: Ring channel structure

Asynchronous circuits are not triggered by the common clock as their synchronous counterparts; they are therefore reactive to input signal transitions. This makes them more vulnerable to intermittent effects, such as noise and glitches occurring on the input lines. Any unspecified input change may start a sequence of internal transitions leading to an erroneous state. Asynchronous interface designs based on a multiplexed bus are prone to such situations [4, 10] to a much greater extent than point-to-point interconnections used in a ring.

The design reported in [12] lacks one important feature: its protocol has not been formally proven correct. It is only believed, on the grounds of simulation results, to be deadlock-free and fair (with respect to its prioritised arbitration mechanism).

We are currently planning to redesign the entire interface on a more systematic basis, with the use of the up-to-date circuit verification and synthesis tools (e.g. [8, 9, 13]) based on labelled Petri nets. One of the tasks to be carried out on this way is to construct a formal Petri net specification of the protocol, which would be amenable for quick changes and efficient manipulations to cater for better performance and reliability. This paper is aimed at showing a way to design a labelled Petri net model for such a (low level) protocol using a compositional approach, outlined, for instance, in [2]. A closely related issue is protocol verification. In this paper, we demonstrate how the net model of the channel can be verified for deadlock-freedom and arbitration fairness. To perform such a checking for a channel of a scalable dimension without a serious descriptive complexity penalty, we employ Pr/T-nets and associated analysis tools.

In the following sections we briefly describe the ring's overall organisation and its protocol (Section 2), the major elements of our compositional approach (Section 3), the protocol model construction (Section 4), and, finally, the protocol verification technique (Section 5).

2 Pipeline Ring Organisation

2.1 Overall requirements

Using the ISO classification for LAN protocols, the token ring under consideration here can be viewed as a medium access control (MAC) layer interface. Our asynchronous protocol is, however, different from a a standard MAC layer ring-based interface known as the IEEE802.5 Token Ring [1]. The main difference is that our protocol was originally targeted at a self-timed (totally free from clocking) communication medium. Another important feature is that this protocol does not allows dynamic "insertion" of users. The users can only switch on- and off-line but not inserted.

The overall structure of the system with a ring channel is shown in Figure 1(a). Each user subsystem is connected to the ring through the local standard bus link (Q-bus in the original design in [12]).

According to the design requirements, the MAC layer channel was made transparent to its users. That is, the network layer protocol entities should be able to communicate knowing virtually nothing about its actual implementation. Even most of the ring's fault-tolerance service was meant to be "hidden" in the MAC layer. It included fault-detection, localisation and self-repair procedures that were invoked with respect to the faults in the ring wires and in the channel adaptor circuitry.

In this paper, for the sake of clarity, we will omit the modelling and verification of the fault tolerance mechanisms of the ring channel. We will focus on the main protocol functionality, the ring access procedures, analysing their safety and fairness properties. Thus we will assume that all fault detection and recovery procedures are abstracted away and both the medium and all adaptors are fault-free.

The higher (network) layer protocol is assumed to be implemented in the user modules, which would exchange messages via their adaptors. Each message consists of a header, message body and a tail. The header contains information on the message priority and type, the recipient addresses, the sender address, and the message length. The tail contains a checksum of the message. The required transfer modes are one-to-one and one-to-many. Message transmission has to ensure that the reception (transmission) by the adaptor at the user link and the transmission (reception) in the ring channel are "decoupled" in time. Messages must therefore be buffered in the adaptor. This is also to enable user subsystems with local clocking to access the channel.

The basic structure of the ring adaptor is shown in Figure 1(b). It consists of two controllers, the user link controller (ULC) and the ring access controller (RAC), and a pair of FIFO buffers. The FIFOs can be designed in a self-timed way (see, e.g., [13]). The design of a link controller is essentially application dependent and is also not discussed in this paper.

Data is transmitted between adaptors in half-bytes using a self-timed, "optimally balanced" code C_6^3 [11] using the four-phase, Return-to-Zero, handshake signalling. Each half-byte is acknowledged in a pipeline manner through the Ack wires.

Interaction between RAC and ULC. The RAC is reset to the initial state by its user via one of the control signals. Another control signal is responsible for setting the RAC to the mode called "System Manager" (SM) , which enables exactly one of the adaptors in the ring to generate the *initial token*. The aim of this action is to start the first channel-acquisition procedure in the ring. In addition, there are some wires between the RAC and ULC, as well as *Reserve* wires the ring, that are used in fault diagnosis, recovery and signalisation procedures – they can be ignored in this paper.

Interaction between RAC and FIFOs. The pair of internal (for the adaptor) nine-bit data buses, supported with two handshake pairs (we use bundled data here for simplicity, which seems to cause no problems if the wire delays are properly controlled at fabrication), connects the RAC to the FIFOs. Thus data is sent between the FIFOs and the RAC in bytes; the ninth bit is used for tagging the end of the messages while they are in FIFOs (either for sending or after having been received).

2.2 Basics of MAC layer protocol

As stated above, our protocol layer is a medium access control (MAC) layer (possibly a sublayer of the link layer) of a LAN with a ring baseband channel. The primary service this layer has to provide to the network layer is reliable transfer of a sequence of message bytes originating in a *sending* user subsystem, via the ULC and buffered in FIFO1, to one (or more) *receiving* user(s), via its (their) ULC(s) and buffering in FIFO2. This structuring of the link layer assumes that the ring channel will effectively consist of a chain of RACs that are capable of getting message bytes from FIFO1 (in bytes) and from their input ring channel port (in half-bytes), and of putting them to their output ring channel port [1] (in half-bytes) and to FIFO2 (in bytes).

To start transmitting data, an RAC must first bid for the channel using a token access method with priorities. When arbitration is won, the RAC becomes the ring's "master" and can transmit its half-bytes into its output channel. The transmission of data is essentially an asynchronous pipeline process, such that each half-byte advances to the next RAC in the ring as soon as a free space is available for it. This is ensured by handshake synchronisation between the transmitting and receiving RACs in each ring segment.

The transmission mode is finished by the "master" after it has determined the end of the message. The message structure is, therefore, not important for our purposes. We will distinguish only five types of data allowed on the ring: tokens M1, M2, M3 (for bidding process); token M4 indicating the end of transmission mode; and data token. The meaning of each token can be understood from the next subsection.

2.3 Channel acquisition protocol

The bidding process uses a dynamic priority increase mechanism (for "fairness"). Its verbal description follows below.

• At the start, after a "major reset" in the system, the System Manager (SM) issues token $M1$. In subsequent operation, the current master becomes a source of the initial token $M1$. It generates $M1$ immediately after the ending half-byte of the last transmitted message.

• Each RAC, starting in the "idle" mode, upon receiving $M1$ from its input channel, checks if there is a request from its FIFO1 to transmit a message. If there is no such a request, the RAC remains "idle" and simply issues $M1$ to its output channel, allowing the token to propagate further. If the request has been registered, the RAC issues, first, token $M2$ and then a half-byte with its priority value N, extracted from the first byte of the message to be sent. It also changes its mode to "bidder" in the competition.

• If a RAC, being in the "idle" mode, receives $M2$ and then a half-byte with the current maximum priority value N_{in}, it becomes an "observer" if no request has been registered, and transmits $M2$ and N_{in}. If, however, the request is set on, it becomes a "bidder", issuing $M2$ and followed by the priority value N_{max}, which is equal to the greater of the following two: its own value N and the value N_{in} arrived from the channel (current maximum). If the RAC being in the "observer" mode receives $M2$ and N_{in}, it remains in the same mode and passes $M2$ and N_{in} to the channel.

• If a "bidder" receives $M2$ and N_{in} from the channel, and its priority N is less than N_{in}, it remains "bidder" and passes $M2$ and N_{in}, whereas if $N_{in} \leq N$ it enters the "master" mode and issues token $M3$, after which it immediately initiates message transfer from its FIFO1 to the channel.

• If an "observer" or a "bidder" receives $M3$ it becomes a "recipient-observer" or "recipient-bidder", respectively, passes $M3$ to the channel and activates its reception operation. The RAC which has become the master of the ring, upon receiving $M3$, also becomes a "potential recipient" of its own message (for error-checking purposes), therefore an intrinsic concurrency, between transmission and reception, is implied by the protocol inside the ring master.

The channel acquisition is completed when all the RACs in the ring are in one of the following three modes: "master-receiver", "recipient-bidder" and "recipient-observer". Two recipient modes are distinguished to indicate a recipient which has a pending request for the channel but whose priority has been insufficient to become the master in the last competition. To avoid starvation, the protocol uses a dynamic priority increase mechanism relative to the

[1] Further, we use terms "input channel" and "output channel" for brevity.

priority that is initially specified in the message[2]. The "master-receiver" option is introduced with the aim to allow the master, when transmitting a message in a broadcast (one-to-many) mode, to address itself. Further we will assume that the ring operates in broadcast mode only.

It is easy to estimate the worst case in which any module may stay a "recipient-bidder" until it becomes "master". If the total number of modules in the ring is n and the number of priority levels is m, the largest possible number of channel acquisitions before a module acquires the medium is $n+m-3$. Of course, in this estimation we assume that the arbitration element inside every module is "locally fair", i.e. it does not ignore the pending request of the module when the "polling" token arrives.

3 Compositional approach to protocol modelling

3.1 Basics of compositional approach

Communication protocols are inherently compositional and the token ring protocol is not an exception. We can clearly identify modules (adaptors) as communicating entities which perform service required from the protocol. Adaptors can then be designed hierarchically so that at the bottom level modules use only primitive actions. One can easily identify certain features in the protocol that pertain to its compositionality.

- A system consists of a number of entities which correspond to different layers in the standard OSI model [5] or may play an auxiliary role, maintaining the work of the others. The entities are interconnected, according to the reference model, through service access points.
- An entity is usually a complex module which, in turn, consists of a number of internal modules – procedures. A procedure normally performs one protocol function. Thus an entity is built from the procedures which are composed together using special rules.
- A procedure has its own internal structure which is constructed from primitives such as service primitives and data units. If a procedure is still too complex, it can be refined to the lower level (sub)procedures which are composed together using the same compositionality rules.

We can thus distinguish three levels of compositionality:

- System level, which represents the overall structure of the system consisting of entities.
- Entity level, where each entity is designed from the procedures combined together using composition rules.
- Procedural level: Each of the procedures is defined using a set of protocol primitives and data units.

3.2 Compositionality in Petri Nets

Petri Nets (PNs) initially attracted attention of protocol designers due to adequate representation of concurrency, existence of formal verification methods, ease of understanding of graphical representation, etc. Yet, they seem to have been lacking compositionality, which,

[2] Note that when the dynamic priority level in a "recipient-bidder" has reached its maximum possible value it will remain constant until the arbitration is won.

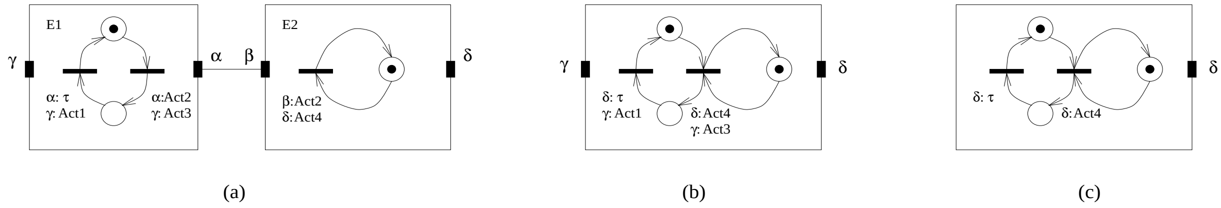

Figure 2: Illustration of the concurrent composition and abstraction operations.

until quite recently [3], has been a major impeding factors on their wider utilisation by protocol designers. Indeed, an attempt to build a PN for a relatively simple protocol may result in a very complicated and difficult to handle model.

Compositional techniques provide a natural way of hierarchical design of a protocol down to the level where each module can be easily represented by a PN. The composition operations can be applied automatically, taking the burden to build a complex representation from the designer. Also, these methods allow abstraction from the details of a particular implementation, making the verification process simpler. In addition, if needed, modules can be verified separately on lower levels, which is often much easier than verification if this module together with the other modules.

3.2.1 Petri net entities

We will assume that the reader is familiar with the basics of the Petri Net theory. We will define a *labelled Petri Net* (LPN) as a PN N in which every transition is labelled with some label from the set Δ. Note that one label may correspond to several transitions. We also define a set of primitives $Act = \Delta \cup \{\tau\}$, where τ is an invisible action.

A *PN entity* (or entity) is a pair $E = \langle L, \Gamma \rangle$, where L is an LPN and Γ is a finite set of labellings called *access points*. Each access point $\gamma : T \to \mathcal{M}(Act)$, where T is a set of transitions on L and $\mathcal{M}(Act)$ is a multiset of communication primitives together with an invisible action τ.

Each access point defines which of the transitions are observed from this access point and how the entity can communicate. A transition may be visible from one access point and invisible from another (if it is labelled with τ).

Graphically, entities are represented as boxes with access points and lines denoting connections between entities. Each of the entities is later refined into an LPN, where each transition is multi-labelled by elementary names preceded by the access point name.

There are two main operations defined on the entities composing them into one LPN for future verification:

- *Concurrent composition*, is a parallel composition of two PNs L_1 and L_2 with synchronisation performed on transitions provided that they are visible through the access points α and β. The visibility of the transitions of the resulting entity through the remaining access points, denoted as $\hat{\gamma}$, is defined as a multiset of the sum of their labellings.

- *Abstraction.* A constructed entity may contain redundant access points. The abstraction operation removes the redundant access points from an entity and yields a new entity with only remaining access points.

We illustrate operations of concurrent composition and abstraction using simple entities given in Figure 2(a). Figure 2(b) demonstrates the concurrent composition of entities E_1 and E_2 and Figure 2(c) illustrates the abstraction operation E removing access point γ.

3.2.2 Petri net procedures

It is not always convenient to represent an entity by an LPN. The LPN representation may be too big and complicated which makes its understanding difficult. In the real life protocol descriptions, a complex entity is usually represented as a composition of procedures. This approach can be naturally applied for describing entities within our framework.

A procedure has same access points as the enclosing entity. In addition, in order to connect procedures themselves, each procedure should have some information about its states. The information about the state of procedure is formalised in the notion of *macrostate*. A macro state is defined as a set of reachable markings $\mathcal{S} = \{M_1, M_2 \ldots M_n\}$ such that each marking M_i contains only one marked place.

We say that a PN is in a macro state if its current marking M belongs to $\mathcal{S}$. Each procedure is defined as a tuple $D = \langle N, \Gamma, \Pi \rangle$, where N is a PN, Γ is a set of access points and $\Pi = \{h, l, r\}$ is a set of *head*, *tail* and *reachable* macrostates respectively. Thus a procedure is defined as an entity equipped with a set of states instead of a single initial marking. These states carry information for further composition of procedures into an entity.

We require new rules to compose procedures. To obtain an entity from a set of protocol procedures we need the following composition operations on the procedural level:

- *Sequential composition*, which composes two PNs together by merging the tail macrostate of the first PN with the head macrostate of the second one.

- *Parallel composition*, which simply puts two PN together without synchronising them.

- *Iteration*, which merges tail macrostate of a PN with its own head macrostate.

- *Disabling*, which merges each reachable macrostate of the first PN with the head macrostate of the second one. Thus D_2 can start from any reachable state of D_1.

- *Transforming to entity*, which prepares a composed entity for the system level composition. It is possible when the head state of the procedure has only one marking.

Each procedure can be refined as a LPN. If the procedure is still too complex to handle, there may be additional level(s) of subprocedures which are composed following the same rules. At the bottom level (sub)procedures are refined into simple LPNs.

We will apply the above framework (its more formal description can be found elsewhere [2]) to the design of the asynchronous token ring protocol.

4 Protocol Definition

Let us assume for simplicity that there are only three adaptors connected into the ring and that there exist only three levels of priority: high, medium and low. It is easy to see that such a model can be extended to an arbitrary number of adaptors and priorities without loosing any important properties.

4.1 Architecture

The general architecture of the token ring was given in figure 1. Each adaptor is connected to its left- and right-hand side neighbour and to the user. The connections with the user are organised through two FIFO modules (FIFO1 for sending information and FIFO2 for receiving information). Both of the FIFO modules are able to process one byte at a time. In our analysis we will assume that there exists an additional buffer (possibly as a part of the FIFO module) which converts the incoming stream of half-byte data into the byte stream

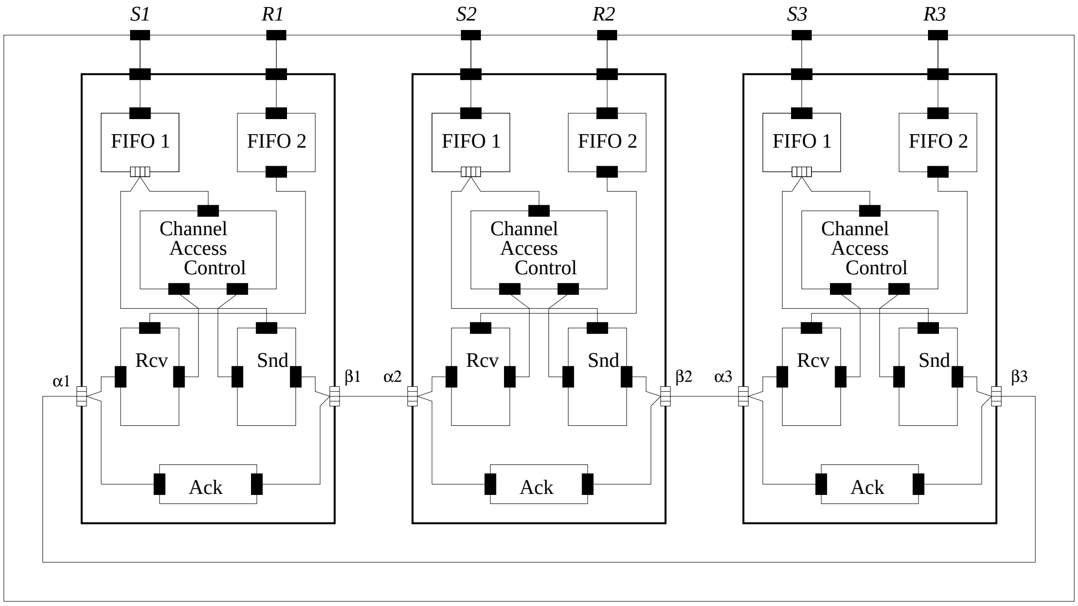

Figure 3: System level description of the token ring protocol.

for the user and performing the reverse operation for sending the data. For our purposes we therefore consider FIFOs as being able to receive a half-byte data stream.

Following the above description of the protocol architecture, each adaptor has access points α and β for connection with its neighbours. S_i and R_i denote access point to the protocol's service. We assume that the service provided by each entity is always consumed by the user and the user always produces another request for service. Hence, access points S_i and R_i can be eliminated by the abstraction operation for the purposes of verifying this protocol.

The entity level description of each adaptor is given in Figure 3. There are six entities in the protocol: *FIFO1, FIFO2, Channel Access Control (CAC), Receiver (Rcv), Sender (Snd) and ACK*. For simplicity we will denote several access points with close semantics as one access point depicted as 'barred' box. The *CAC* entity is responsible for the bidding process. Therefore, in order to verify correctness and fairness of the protocol we are primarily interested in the verification of the *CAC* entity. It is connected to *Sender* and *Receiver* entities which are responsible for communication with the neighbours of an adaptor. *Sender* and *Receiver* are connected to the *FIFO1* and *FIFO2*, respectively, to facilitate independent data transmission and reception. When a user issues a request to *CAC* it starts bidding for channel access. *CAC* issues necessary primitives to *Sender* to send appropriate tokens and "listens" to primitives from *Receiver*. When the bidding process is completed, *CAC* initiates data transmission and reception (when bidding was won) or data "passing", i.e. reception and transmission further along the ring (when bidding was lost). *Sender* and *Receiver* issue necessary primitives to *CAC* to signal the end of each process. The *Ack* entity is responsible for maintaining the self-timed communication between the adjacent adaptors (in a pipeline fashion).

The *CAC* entity has access points connecting it with with *FIFO 1* and *Receiver* and *Sender*. In addition to the access points mentioned above, the *Receiver* entity has an access point connecting it to the access point to the left-hand side neighbour and the *Sender* entity has an access point ensuring the transmission of the data to the right-hand side neighbour.

4.2 Entity level

Entities *FIFO1, FIFO2* and *Ack* are straightforward to refine into LPNs. In the real protocol *Receiver* is also responsible for decoding the address in the beginning of the message. We

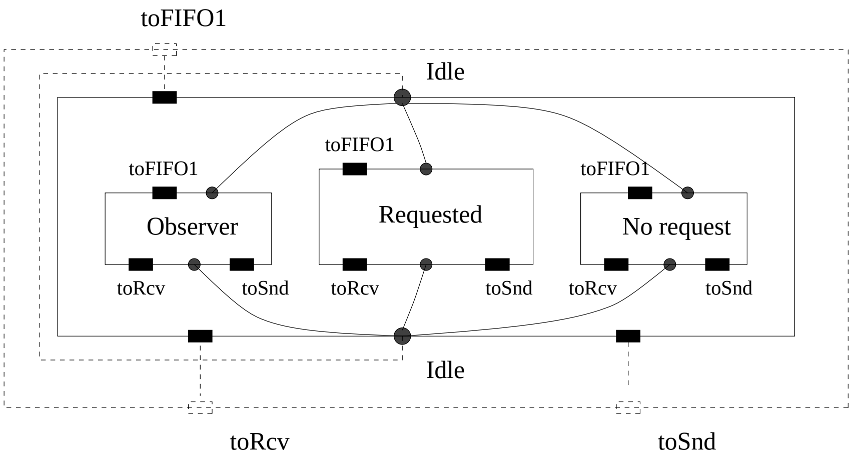

Figure 4: Entity level representation of the CAC entity.

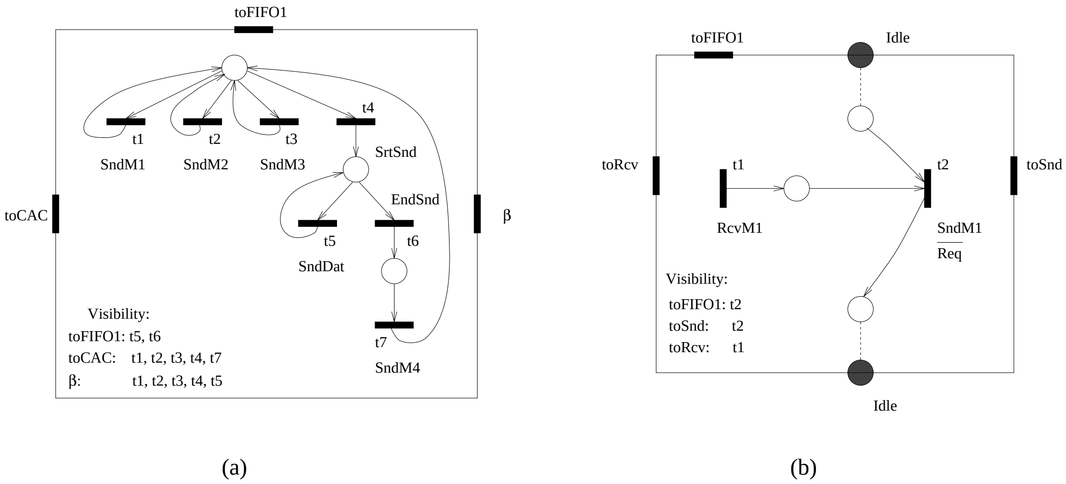

Figure 5: Refinement of the *Sender* entity (a) and `No Request` procedure (b).

are considering only the broadcasting one-to-all mode and therefore can abstract from the realisation of address decoding. Since we are also not concerned with fault detection and recovery procedures, the *Receiver* and *Sender* entities are refined into simple LPNs. An example of the refinement of *Sender* entity is given in Figure 5(a). All transitions are labelled with some communication primitives. Transitions are visible from different access points. For example, transition t_5 labelled with the primitive *SndDat* (send data to the right-hand side neighbour) is visible through access point *toFIFO1* and β but not visible through *toCAC* access point.

However, *CAC* entity is still too complex and requires further decomposition on procedural level. The decomposition of *CAC* entity is given in Figure 4. Two procedures correspond to the adaptor being in "idle" (`No request`)) and "recipient-observer" (`Observer`) modes. The third procedure (`Requested`) corresponds to all modes and operations of an adaptor when it has registered a request from its user.

4.3 Procedural level

We need to refine procedures comprising the *CAC* entity. The procedures are simple and we do not need to build additional levels of subprocedures. Refinement of procedure `No request` is given in Figure 5(b). This procedure is connected to the *Sender*, *Receiver* and *FIFO1*.

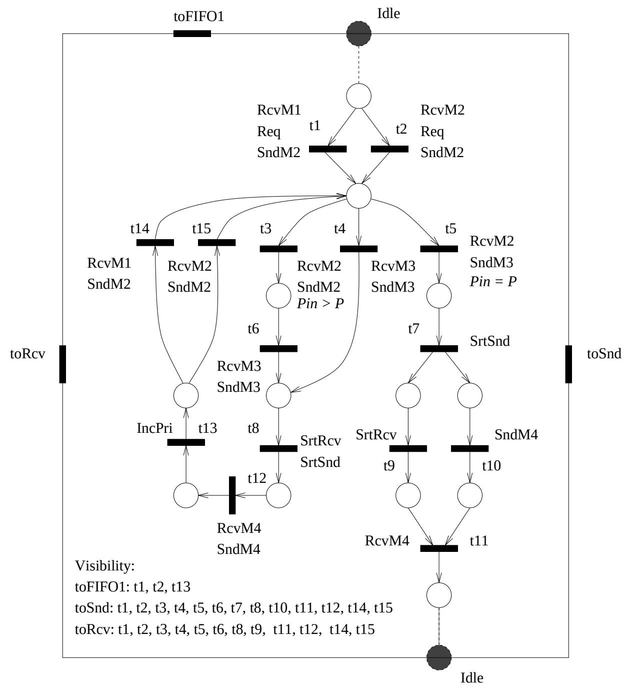

Figure 6: Refinement of procedure `Requested`.

From *FIFO1* it receives $\overline{Req}$ primitive. Note that one transition is labelled with a multiset of primitives. The refinement of procedure `Requested` is given in Figure 6. Note that in `Requested` procedure we can clearly identify two branches corresponding to an adaptor being in "master-receiver" and "receiver-bidder" modes. For simplicity we have drawn additional conditions on the priority as conditions on transitions ($Pin > P$ and $Pin = P$). Readers are invited to refine the third procedure.

5 Protocol verification using Pr/T nets

The compositional approach allowed us to design the protocol in hierarchical manner. After refining procedures on the lowest level, we can perform their composition. The resulting LPN represents the entire protocol layer which we need to verify.

The ring is a regular structure consisting of an "array" of identical elements. To avoid complexity of the protocol representation we can use Predicate/Transition (Pr/T) nets [6] for its representation. This will also allow us to use existing tools, such as PROD [7], for its verification.

In order to obtain the Pr/T net description of the protocol we, first, perform composition of one adaptor. Obtaining a Pr/T net description from the composition is straightforward. The resulting LPN will be the basic structure of our Pr/T net. We need to introduce additional places to represent the token flow. We add five places which correspond to the types of allowed tokens: M1..M4 and data token. Each token is assigned with an adaptor number $\langle a \rangle$ representing the fact that the token is enabling actions of that particular adaptor. In addition, those tokens that are in M2 place are labelled with tuples $\langle a, p \rangle$ where a is an adaptor number and p is the priority number.

Each place is then connected to the corresponding transition in the LPN which was formed by the synchronisation of transitions from *Receiver* entity. Each of these transitions consume tokens labelled with the single tuple $\langle a \rangle$ which represents the fact that adaptor a receives an input (token $\langle a, p \rangle$ in place M2 indicates that adapter a received token M2 with priority p). Similarly, these places are connected to the transitions formed by synchronisation with the

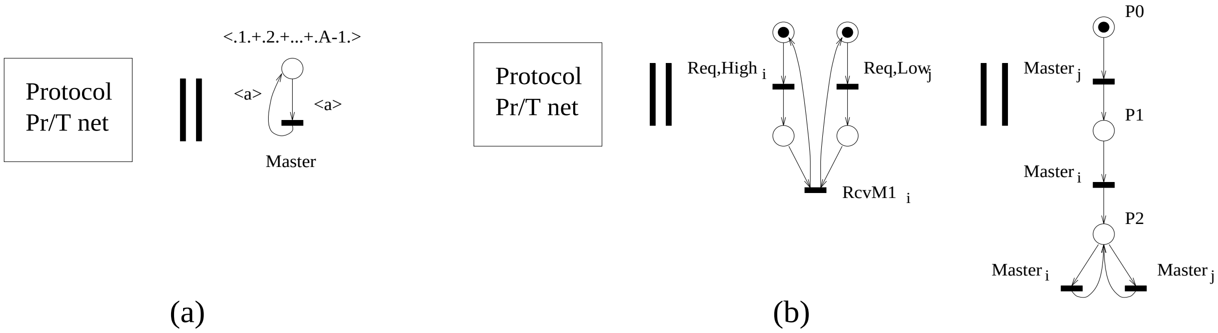

Figure 7: Verification of protocol properties.

transitions of *Sender* entity. In this case these transitions produce tokens labelled with $\langle a+1\rangle$ – the number of the "next in row" adaptor.

Transitions which are internal to each adaptor consume and produce tokens without changing the adaptor number but possibly changing other values (e.g. priority level).

Initially there are A tokens in the place "Idle" and a token with the value $\langle 1\rangle$ into place M1. This corresponds to the initial injection of token M1 into the ring by the System Manager.

The Pr/T-net model of the ring has been verified using PROD [7]. For deadlock detection we used the "stubborn set" method implemented in PROD. This method builds a reduced reachability state space (RRSS). Using Pr/T nets gives a convenient way of verification of arbitrary number of adaptors and priorities in the protocol. For example, adding a new adaptor is done by adding a token assigned with $\langle A+1\rangle$ into place "Idle" in the initial state of the protocol. Analysis showed no deadlocks in the model. The reachability analysis (see Table 1) reveals exponential growth of the RRSS in the number of adaptors and polynomial growth in the number of priorities in the ring. As the number of adaptors we take the number of active adaptors in the ring of three, i.e. those adaptors trying to gain access to the channel. An inactive adaptor is assumed not to have received requests from its user for data transmission.

We are also interested in some safety and fairness properties of our protocol. We analyse them by reducing their check to deadlock detection.

Safety property of arbitration. It ensures that the access to the ring will not be given to two or more adaptors simultaneously. To reduce the problem to deadlock detection we add a *stop-transition* to the Pr/T net description of the protocol with the input place corresponding to a place of the adaptor being in the master state and outputting into a deadlock. This transition is allowed to fire when there are two tokens in the master state. Analysis of the ring with three adaptors and three levels of priorities shows that there are no deadlocks in the net with the new transition, checking 77661 states in its RRSS.

Fairness of arbitration. It shows that a user issuing a request for data transmission via its adaptor will eventually gain access to the ring. We can check it by composing the net description of the protocol with the net shown in Figure 7(a) (where transition *Master* denotes an adaptor's state transition to master). In the Pr/T nets we need to add a place "guarding" the change into the master state. This place is marked with $A-1$ tokens labelled with the adaptor numbers. Thus we will prevent the A-th adaptor from entering the master state, i.e. acquiring the channel, and reaching any further state. If arbitration was unfair, our composed Pr/T net would not have deadlocks as in this case the inability of some adaptor to reach the master will be ignored. Analysis shows that the net deadlocks, after exploring 57639 states in its RRSS.

Priority order. It ensures that, if two users have issued requests with two different priorities before the adaptor having the request with the higher priority receives token M1 or M2, the one with the higher priority will gain access to the ring first. This problem can

No. adaptors	No. priorities	RRSS Size
1	1	104
1	2	130
1	3	156
2	1	1050
2	2	2301
2	3	4135
2	5	9204
2	10	30049
3	1	8307
3	2	31404
3	3	84432

Table 1: Experimental results.

be reduced to deadlock detection by performing concurrent composition of nets as shown in Figure 7(b). Such a composition allows to bring the Pr/T net of the protocol to the initial marking at which two requests of i (with the *High* priority level) and j (of the *Low* priority) have been issued before the i-th adaptor registers its request, i.e. before tokens M1 or M2 arrive to the i-th adaptor. The right-hand side net orders only the first firing of transitions labelled with $Master_i$ and $Master_j$ (representing adaptors entering the "master" mode) and allows their subsequent firing to occur in any order (thus totally controlled by the protocol net conditions).

Under the priority conditions set above, $Master_i$ is supposed to fire first. This should force the composition net into a deadlock. Indeed, if, in the protocol Pr/T net, the j-th adaptor becomes *Master* before the i-th adaptor (i.e. the protocol Pr/T net violates the given priority order), then, in the composed system, place p_2 will be marked. This means that there exists a marking in the system at which both $Master_j$ and $Master_i$ can fire in either order. Since there are no deadlocks in the protocol Pr/T net, then there must be no deadlocks in the composed system. On the other hand, if the protocol net does not allow the j-th adaptor to enter the *Master* state before the i-th adaptor (i.e. the protocol maintains the given priority ordering), then the system will (by the property of arbitration fairness) reach some marking at which only $Master_i$ is allowed to fire. But, if $Master_j$ has not fired yet, the place p_1 is not marked and hence the system must reach a deadlock.

Verification shows absence of deadlocks if the proper access ordering is applied ($\langle i \rangle$ goes before $\langle j \rangle$) and reports a deadlock otherwise. Since there can be no two adaptors simultaneously accessing the ring (safety property), we conclude that the required order is maintained in our protocol.

Conclusions

We have demonstrated a compositional approach to designing an asynchronous token ring protocol. We have verified the labelled (Pr/T) net description of the protocol. The protocol has been shown to be deadlock-free and having certain properties of fairness of arbitration. This has not been done in the previous design of the protocol and its asynchronous circuit implementation.

Verification of the "selective" broadcast addressing method and of the fault tolerance mechanisms exploited in the ring has been left outside the scope of this paper. We believe that the use of the compositional approach adopted here would allow us to easily build a complete model for the verification of those mechanisms. In particular, we would intend to show its tolerance to maximum two static faults (it had originally been designed for). We plan to address these problems in our future work along with providing an "on-line" verification of circuits implementing the new version of a channel adaptor.

References

[1] ANSI/IEEE Standard 802.5 Working Group. Token Ring Access Method and Physical Layer Specifications. IEEE, N.Y., 1985.

[2] B.A. Anisimov and M. Koutny. Compositionality and Petri nets in protocol engineering *Manuscript*, December 1994.

[3] N.A. Anisimov. An Algebra of Regular Macronets for Formal Specification of Communication Protocols. *Computers and Artificial Intelligence,*, Vol. 10, 1991, pp. 541–560.

[4] B. Coates, A. Davies and K. Stevens. The Post Office experience: designing a large asynchronous chip. *Integration: the VLSI journal*, Vol. 15, No. 3, Oct. 1993, pp. 341 – 266.

[5] J. D. Day, H. Zimmermann. The OSI Reference Model. *Proceedings IEEE* **71** (1983) 1334–1340.

[6] H.J. Genrich. Predicate/transition nets. *Advances in Petri Nets, LNCS 256*, Springer-Verlag, 1987, pp. 207 – 247.

[7] P. Grönberg, M. Tiusanen and K. Varpaaniemi. PROD - A Pr/T-net reachability analysis tool. Series B: Technical Reports, No. 11, Helsinki University of Technology, June 1993.

[8] M. Kishinevsky, A. Kondratyev, A. Taubin, V. Varshavsky. Concurrent Hardware: The Theory and Practice of Self-Timed Design. John Wiley and Sons, London, 1993.

[9] L. Lavagno and A. Sangiovanni-Vincentelli. Algorithms for synthesis and testing of asynchronous circuits. Kluwer Academic Publishers, 1993.

[10] K.S. Stevens. Practical Verification and Synthesis of Low Latency Asynchronous Systems. PhD Thesis, The University of Calgary, Calgary, Alberta, Sept. 1994.

[11] V.I. Varshavsky, M.K. Kishinevsky, V.B. Marakhovsky, V.A. Peschansky, L.Ya. Rosenblum, A.R. Taubin and B.S. Tsirlin. *Self-Timed Control of Concurrent Processes*, Ed. by V.I. Varshavsky. Kluwer AP, Dordrecht, 1990 (Translated from Russian; Russian Edition – Nauka, 1986).

[12] V.I. Varshavksy, V.Ya. Volodarsky, V.B. Marakhovsky, L.Ya. Rosenblyum, Yu.S. Tatarinov and A.V. Yakovlev. Structural organisation and information interchange protocols for a fault-tolerant self-synchronous ring baseband channel (pt.1). Hardware implementation of protocols for a fault-tolerant self-synchronous ring channel (pt.2). Algorithmic and structural organisation of test and recovery facilities in a self-synchronous ring (pt.3). *Automatic Control and Computer Science*, Vol. 22, No. 4, pp. 44 – 51 (pt.1), No. 5, pp. 59 – 67 (pt.2), Vol. 23, No. 1, pp. 53 – 58 (pt.3), 1988, 1989 (translated from Russian).

[13] A. Yakovlev, A.M. Koelmans and L. Lavagno. High level modelling and design of asynchronous interface logic. *IEEE Design and Test of Computers*, Spring 1995, pp. 32–40.