Designing an Asynchronous Pipeline Token Ring Interface *

A. Yakovlev
Dept. of Computing Science
Univ. of Newcastle upon Tyne
NE1 7RU, England

V. Varshavsky
Hardware Dept.
University of Aizu
Japan 965-80

V. Marakhovsky
Software Dept.
University of Aizu
Japan 965-80

A. Semenov
Dept. of Computing Science
Univ. of Newcastle upon Tyne
NE1 7RU, England

Abstract

We describe the design of a speed-independent interface based on a pipeline token-ring architecture. The original goal was to build a reliable communication medium, able to tolerate up to two faults in any segment of the ring, to be used in an on-board multicomputer. We believe that the pipeline ring approach can help reduce some negative "analogue" effects inherent in asynchronous buses (including on-chip ones) by means of using only "point-to-point" interconnections. We briefly outline the major ideas of the channel's organisation, protocol and our syntax-driven implementation of the channel protocol controller. The protocol has been recently verified for deadlock-freedom and fairness.

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.

Asynchronous circuits, whose action is not triggered by the common clock as in their synchronous counterparts, are 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. One example has been reported by the designers of the Post Office chip [1, 2]. Its "spine bus", interfacing several ports and the local processing element, multiplexes the request and acknowledgement handshake links. It is emphasised in [2] that "when asynchronous handshake signals are not local ... care must be taken to assure that failures do not occur due to violations of the assumption that signals are "digital"".

Such "analogue" problems happen on a wired-OR asynchronous bus because each simultaneous input can contribute incrementally to the signal level. Glitches may be caused by multiple reflections of signals produced by bus drivers (usually implemented by open-collector devices) [3]. One example was reported for a broadcast handshake (with one sender and several receivers, using a wired-OR interconnection) in Futurebus [4] where special integrator-based bus drivers were considered as a possible way to deal with the problem. These drivers however add significant delays thus casting doubts on their applicability for high-speed data broadcasting. Additional problems of wired-OR logic concerned with its effect on self-checking properties of the interface were discussed in [5].

A few years ago three of us were involved in the development of a fault-tolerant asynchronous interface for an airborne computing system [6] (a safety-critical application). We designed a communication medium that was able to tolerate up to two faults in any segment of the ring. The techniques employed in this design for fault-detection, localisation and recovery were recently presented in [7]. Our choice of a ring architecture was caused by the combination of functional requirements, e.g., flexibility of addressing – "selective broadcasting", distributed arbitration, minimal channel wiring, as well as reliability concerns of the above-mentioned character.

This paper looks, though a bit retrospectively, at our design and attempts to draw the attention of practical designers to the advantages of pipeline ring architectures in asynchronous interfaces. We hope that today, with the recent progress made in asynchronous design methods and tools, the ideas of [6] can be viewed in a different, more pragmatic, light.

In the following sections we briefly describe the overall organisation of the channel (Section 2), its protocol (Section 3) and the design of a channel adapter (Section 4).

2 Pipeline Ring Organisation

2.1 Overall requirements

According to ISO classification of multi-layered Local Area Network architectures our pipeline ring is a medium access control (MAC) layer interface. The latter is a sublayer in a link layer, which is normally

*For A.Yakovlev this work was partly supported by EPSRC grant No. GR/J52327. A.Semenov's work is supported by the grants of Newcastle University's Research Committee and CVCP.

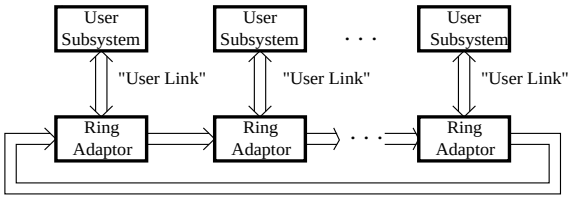

Figure 1: Ring channel structure

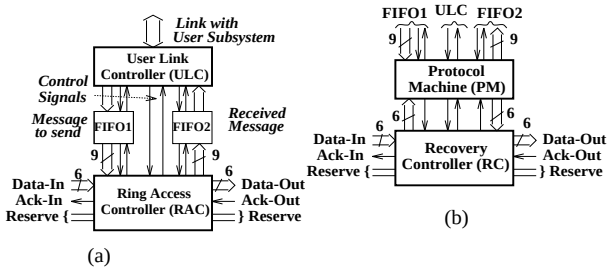

Figure 2: Ring adaptor structure (a) and ring access controller substructure (b)

set between the network layer and the physical layer. A better-known example of a MAC layer ring-based interface is the IEEE802.5 Token Ring [8]. Token Ring was however not aimed at asynchronous designs and would not be fully compatible (in its protocols and implementation assumptions) to our goal – to design a speed-independent interface. Furthermore, some of the aspects of the IEEE802.5 Token Ring are not deemed to be relevant to our application, such as "dynamic insertion" of new users. The on-board system using this channel is assumed to be a virtually imbedded system where subsystems can only be turned-off in the on-line mode but not inserted.

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

According to the design requirements, the MAC layer channel is 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 is meant to be "hidden" in the MAC layer. It includes fault-detection, localisation and self-repair procedures that are invoked with respect to the faults in the ring wires and in the channel adaptor circuitry.

The network layer protocol is assumed to be implemented in software 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 2(a). 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., [9]). The design of a link controller is essentially application dependent and is also not discussed in this paper. The RAC is further subdivided into a protocol machine (PM) and a recovery controller (RC), as shown in Figure 2(b). The PM circuit, being of our main concern in this paper, implements the MAC protocol, while the RC acts as a "wrapper" protecting the PM from the real, potentially faulty environment. The RC realises the main fault-diagnosis, fault-localisation and recovery procedures described in [7]. It is convenient to separate the latter concerns from the MAC protocol issues in the remainder of this section. Therefore, when talking about a ring adaptor or a RAC we will effectively mean its PM unless stated otherwise.

To ensure correct transmission of data between adjacent adaptors, regardless of the relative delays in parallel wires, a self-timed "optimally balanced" code C_6^3 [10], of a family of Sperner codes [11], has been employed. With this code, using 6 lines, it is possible to have 20 valid code words, which allow to encode one half-byte of information. The valid code words alternate with the all-zero spacer, thus creating a four-phase, or return-to-zero (RZ), handshake signalling protocol between adjacent adaptors. In this action, the detection of three ones on the code lines (*Data-In*, if we look at the left-hand side link of the RAC) indicates setting of the valid code word (*Request* signal). The code word reception is acknowledged by an explicit *Acknowledgement* signal (*Ack-In*).

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. If an error arises in delivering the SM status to one of the RACs, the system is reinitialised by a special self-recovery procedure, and the role of the SM is passed to another module (initially configured as "System Deputy"). Both FIFOs can be reset by the ULC to eliminate the remainder of an untransmitted message for subsequent re-transmission should this be effected upon by the network layer protocol. The group of control signals also includes a pair of signals from the RAC to its ULC, to inform the latter about a fault in FIFO1 or FIFO2 when data is "pumped" in or out, respectively. In response to these signals, the ULC must reset the corresponding FIFO(s), and either arrange retransmission of the previous message in the channel (for the FIFO1 case), or

transmit an overhead message to the source of the undelivered message (from FIFO2), asking the source to retransmit the latter. These issues must however be resolved at the network level. Finally, one more control signal is used by the RAC to inform its user about a fault in the channel.

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 use of the nineth bit is explained later.

Note that despite the one-way transmission (say, "clockwise") in the ring, the actual interconnections between adaptors are bidirectional. In the fault-localisation and recovery procedures these lines should handle signals sent in both directions. According to the fault-tolerance requirements, the channel has to recover from up to two faults in one ring segment, thus two spare (*Reserve*) lines can be used either for information or acknowledgement signals. The technique we use for spare line substitution is the so-called *sliding redundancy* [12], in which when the i-th line fails, all the signals whose line numbers were from i onwards are connected to one line ahead. That is signal i is connected to line $i+1$, and so on. In this case the physical line used as a logical acknowledgement line may become a logical information line. It should be noted that, in addition to the wire itself, a line includes also the amplifiers and receivers operating on the line.

3 MAC Layer Protocol

3.1 Basics

As stated above, the protocol layer under consideration 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 of this layer to the network layer is providing a 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. If 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.

<message1> ::= <header1> [<data part>] <terminator1>
<header1> ::= <byte of priority and recipient address length K>;<bit9=0>
{<recipient address byte>;<bit9=0>}^{K}
<data part> ::= {<data byte>; <bit9=0>}^{N}
<terminator1> ::= <end-word byte>; <bit9=1>

(a)

<message2> ::= <header2> [<data part>] <terminator2>
<header2> ::= <state byte>;<bit9=0>
<data part> ::= {<data byte>; <bit9=0>}^{N}
<terminator2> ::= {<end-word byte>; <bit9=1>}^{K}

(b)

Figure 3: Message formats: FIFO1 (a) and FIFO2 (b)

Each RAC must be able to extract information related to the MAC layer out of the messages coming from its FIFO1 or from the input channel. This includes the message priority and the recipients' addresses. On the other hand, each RAC must be able to supply information about the ring's status in the message put to its FIFO2. A structure is therefore put on the message format. Messages coming from FIFO1 and sent to FIFO2 have the formats shown (in a customary *bnf*) in Figure 3(a) and (b).

Note that recipient addresses are represented in a unitary code; this is to simplify the task of implementing a "selective broadcast" mode. Furthermore, this allows to implement the address decoding in a purely speed-independent way. We assume that in "embedded" applications, the number of modules using the ring would not be large [2].

It should be obvious that certain parts of the network layer message structure, set in the previous section, may be ignored at the MAC layer. They are transferred as ordinary data bytes (`<data part>`). For instance, since the end of each message is tagged by a byte in which the tag bit (`<bit9>`) is set to 1, the end of a message coming from FIFO1 can be determined by the RAC without counting its length. Therefore the MAC layer's "envelope" needs neither the message length nor its type. We also decided that the checksum is not produced on this layer since the transmission is done by means of a self-timed code, which is robust to one-way bit value distortions (zero instead of one). The checksum diagnosis is assumed to be done at the network layer.

3.2 Channel acquisition protocol

The bidding process uses a dynamic priority increase mechanism (for "fairness"). Its verbal description follows below. The part referring to each RAC is formally described by an interpreted Petri net shown in Figure 4.

At the start, after a "major reset" in the system, the System Manager (SM) issues token $M1$.

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

[2] In fact, in our original circuit implementation we limited it to eight, thus allowing for one address byte.

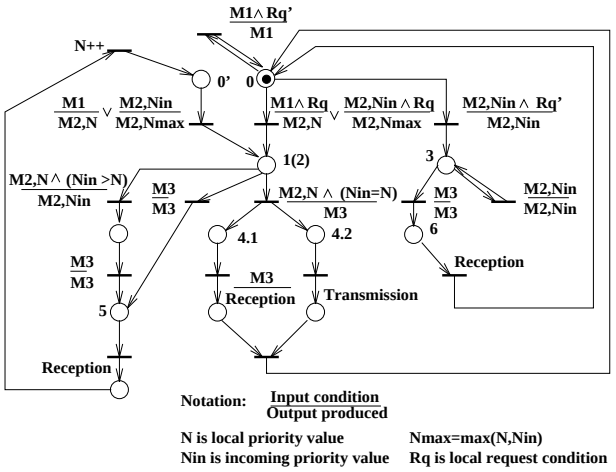

Figure 4: Ring channel acquisition procedure defined by interpreted Petri net

Note. All the tokens are also encoded as half-byte(s), which is made possible because the six-bit self-synchronising code C_6^3 allows to have 16 data words and 4 spare words that are used to encode control tokens, three of which are employed in the channel acquisition process, and the forth, $M4$, is used as a terminator in the sequence of half-bytes for each message transmitted by the current master of the ring.

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 state 0 ("idle"), upon receiving $M1$ from its input channel, checks if there is a request from its FIFO1 to transmit a message (this is done by means of a two-way arbitration as the choice is essentially non-deterministic since the arrival of the request is totally independent of token propagation). If there is no such a request, the RAC remains in state 0 and simply issues $M1$ to its output channel, allowing the token to propagate further. If the request has been registered (through a local mutual exclusion mechanism), 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 state (from 0 to 1), becoming a "bidder" in the competition.

If a RAC, being in state 0, receives $M2$ and then a half-byte with the current maximum priority value N_{in}, it becomes an "observer" (state 3) if no request has been registered, and transmits $M2$ and N_{in}. If, however, the request is set on, it becomes a "bidder" (state 1), 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" state receives $M2$ and N_{in}, it remains in the same state 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" state (represented by a pair of places 4.1 and 4.2) 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 (the data transfer protocol allows this option 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 states: "master with recipient", "recipient-bidder" and "recipient-observer". Two recipient states 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 (note a transition labelled with $N{+}{+}$) relative to the priority that is initially specified in the message [3].

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 $n+m-3$. Indeed, assume that the given module needs to send a message with the lowest priority level (say, 0). Then let, for every acquisition session, be always a module whose priority is higher. The largest number of sessions before the given module's priority reaches the max priority is therefore $m-1$. Then, if we assume that, when the last such session completes, the "master" of the ring is the forward neighbour of the given module. It is obvious that there can be at most $n-2$ more sessions won successively by other modules, before the "winning turn" reaches the given module. 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.

The "master with recipient" option is introduced with the aim to allow the master, when transmitting a message in a broadcast mode, to address itself, e.g. to facilitate a check of the message back in the sender and by whom and how it has been received. A special mechanism of invalidating the valid address bits is used (see the next section).

The net shown in Figure 4 plays the key role in verifying the overall protocol. An outline of the verification technique is presented in Appendix.

[3] 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.

3.3 Addressing Method

As was pointed out in [12], within a bus architecture, there seems to be no purely speed-independent solution to a problem of the implementation of a "collective acknowledgement" in *one-to-many* data broadcasts. The use of wired-OR logic can be a way but only with certain delay assumptions. In a ring-based organisation the principle of speed-independence can be implemented fully. Here a message sent by the master into the ring and received back can be such a "collective acknowledgement".

We used the following *relative* addressing method in the ring. Since the address is given by unitary code, the length of the address field of the message is equal to the number of modules in the system. Each module, whose number in the ring corresponds to the number of the bits set to 1 in the address field is *selected* as a recipient of the message. Thus, the presence of several (all) 1's in the address specifies a one-to-many (one-to-all) broadcast mode of interaction. In addition, to ensure that the address-decoding circuit is self-checking, each module must ensure a one-to-one mapping between the "geographical" position of the module relative to the master and the number of the corresponding bits in the address field. The latter is achieved as follows.

Consider a module that is at position i from the master in the "forward" direction of the ring (say, clockwise). The "geographical" position of the module relative to the master stipulates that 1 is set in the i-th bit of the address counting from the most-significant bit (MSB). When a message travels around the ring, each module records the MSB, and then passes the address field to the ring with a one-bit shift towards the MSB. Thus, the i-th module receives the address field whose MSB *always* contains the "selected/unselected" (SEL) flag for the i-th module. This flag is stored for the period until the next decoding event. If SEL is equal to 1, then RAC copies the message (byte-by-byte) to FIFO2 and to its output channel port. If SEL is at 0, only the latter action is performed. Using this organisation of addressing we virtually avoid building a decoder circuit and resolving its self-checking problem. This imposes, however, additional functionality on the higher network layer, which should provide relative addresses in the transmitted messages. This is practically the only factor negatively affecting the transparency of the MAC layer. As a possible way to implement the relative addressing, each user subsystem can store a table to map between relative and absolute addresses of other subsystems in the network.

The following approach may be used to facilitate the update of this mapping in each subsystem should such a need arise after a potential system reconfiguration. Assume that a module i that has been in the r-th geographical position relative to a given subsystem fails. The formerly $r+1$-th module would thus become the r-th in the ring and so forth. To update the mapping, the network layer protocol should execute a procedure in which the subsystem becoming the System Manager (SM) sends a message of a "Who are you?" type to each module *individually* (successively generating a 1 – in a "one-hot" way – in each of the address bits), thus learning the absolute addresses of its newly configured relative neighbours. Upon receiving a 1 in the r-th position, the r-th module issues a "This is who I am" message in a *one-to-all* broadcast mode (since it cannot know who sent him the "Who are you?" request). This message is received by the SM, who records the new absolute address of the module in the r-position of the table. After the table is updated, the SM can transmit its own table to the remaining modules. The latter adjust their own tables allowing for a shift of the table positions made relative to the SM's table.

The remaining parts of the protocol and its verification are described in Appendix.

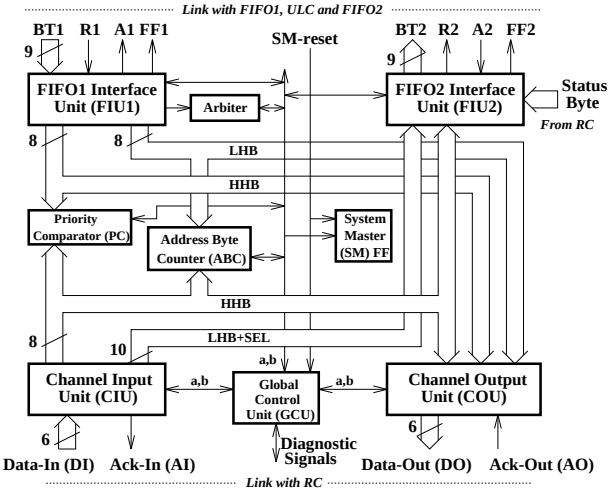

Figure 5: Protocol Machine Structure

4 Protocol Controller Design

4.1 General Structure

Recall the ring access controller (RAC) structure from Figure 2(b). By separating the MAC protocol functions from the fault-tolerance mechanism employed at the ring level we try to insulate the PM from all problems concerned with fault-diagnosis and recovery. The RC therefore acts as a structural "wrapper" for the PM. At the same time, when faults occur in the interface between the RCA and FIFOs they are detected by the PM and lead to exceptions raised to the ULC (see Appendix) through special control signals.

In this section we present the major ideas lying behind the PM design. The two main aspects of concern here are the operational structure (i.e., data path) and the control mechanism of the PM. Other issues include the types of data path operations, the notation for control flow specification (obviously related to the Petri net language we used for protocol description), and the way of converting the control description into the circuit. All these are briefly outlined in the following subsections.

4.2 Operational structure

The overall structure of PM is shown in Figure 5. It consists of channel input and output units (CIU and

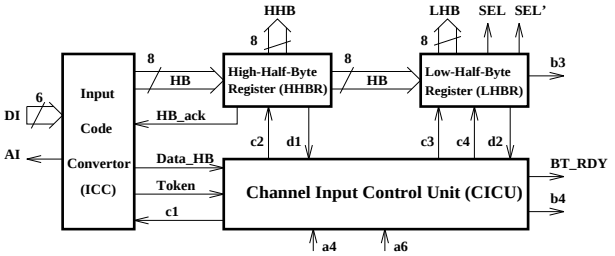

Figure 6: Structure of the channel input unit

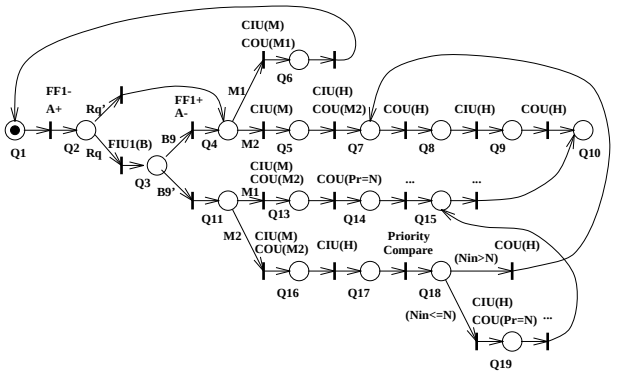

Figure 7: Fragment of Petri net specifying the control circuit

COU), FIFO1 and FIFO2 interface units (FIU1 and FIU2), a priority comparator (PC), an address byte counter (ABC) [4], an arbiter, a "System Manager" flag flip-flop (SM-FF), and a global control unit (GCU).

All these units, excluding the GCU, comprise the data path. They interact with the GCU via handshake pairs, requests a_i and acknowledgements b_j. Their internal implementation is based on aperiodic circuit design solutions published elsewhere. For example, the "Sperner code – dual-rail code" convertor is described in [10]. All internal data paths in the PM are dual-rail encoded. This is to facilitate the purely speed-independent implementation of the PM. Due to the lack of space we cannot describe the logic of each unit. Only their major functionality is outlined below.

The CIU receives half-bytes (C_6^3-encoded) from the input channel, converts them into dual-rail code for 16 data words and one-hot signals for each token $M1, ..., M4$. It also builds a byte out of two adjacent half-bytes, the byte to be sent to the FIFO2 via FIU2, registers an incoming priority half-byte for comparison (in PC) with the local priority value, places the address length value into ABC, and forwards a half-byte, either intact or with an MSB shift (for address or end word), to the output channel via COU.

The FIU1 receives and temporarily stores a byte from FIFO1, produces two half-bytes for the output channel via COU, maintains the local priority value (incrementing it in the "recipient-bidder" state), passes the "Message is Ready" request to the arbiter and signals to the ULC if a fault occurs in reading from FIFO2.

Both CIU and FIU1 are units with a storing capacity. They contribute as a memory stage to the data path pipeline of the ring. Other data path units, COU and FIU2, are combinatorial with respect to data path, their main function being multiplexing several paths into one. A specific feature of COU is that it is also a convertor from the internal dual-rail (one-hot for tokens $M1, ..., M4$) to the external C_6^3 code.

The meaning of the remaining units, arbiter, CP, ABC and SM-FF should be obvious from their names. Note also that the CIU provides a flip-flop to store the value of the SEL flag. The mechanism of signalling faults (FF1 and FF2) by FIU1 and FIU2 is based on the idea of applying a critical time-out (a scalable delay element) when waiting for a request $R1$ from FIFO1 and an acknowledgement $A2$ from FIFO2.

[4] This counter actually consists of two independent counters, for the address and end word length counting.

4.3 Two-level control structure design

When organising the control structure, a number of design factors should be born in mind to a much greater extent than for the data path units. The most important are performance, control circuit complexity and overall wiring complexity. We use a two-level control approach. The top level is centralised to conform the nature of the MAC protocol, whose main part was shown in Figure 4. At the same time, some operations involved in the protocol actions (see the transition labels in Figure 4), such as receiving tokens or half-bytes with or without a shift, address recognition, producing a byte from two adjacent half-bytes and others can be controlled at a lower level. As an example let's consider the internal structure of the CIU shown in Figure 6.

This unit is a typical case for the second control level. The overall organisation of this unit is a two-stage pipeline register with a code convertor at the front. Its interface with the central control (GCU) is realised via the (a, b) handshake pairs, where the CIU also produces a number of "condition" signals, such as one-hot token values, and the "byte is ready" and SEL flags, which help to steer the global control flow in GCU. The local control is based on the (c, d) handshakes, where the sequences of lower level commands c_i are produced from higher level requests a_j.

Note that the signals involved in different pairs are not necessarily disjoint – e.g., request a_4 may be responded by either b_3 or b_4, depending on the value in the data path. Appropriate elements are used in the control circuit to synchronise the pairs.

To determine the nomenclature of the handshakes operated by the GCU, we first refined the operation labels in the protocol description. Then, to minimise the number of individual handshake pairs we partitioned a total set of operations into the sets of generic operations as per data path units. For example, for the CIU: CIU(M) means "receive a token from the input channel", its controls are (a_4, b_4); CIU(H) is "receive a half-byte from the input channel" (a_4, b_6); CIU(S) is "receive a half-byte from the input channel

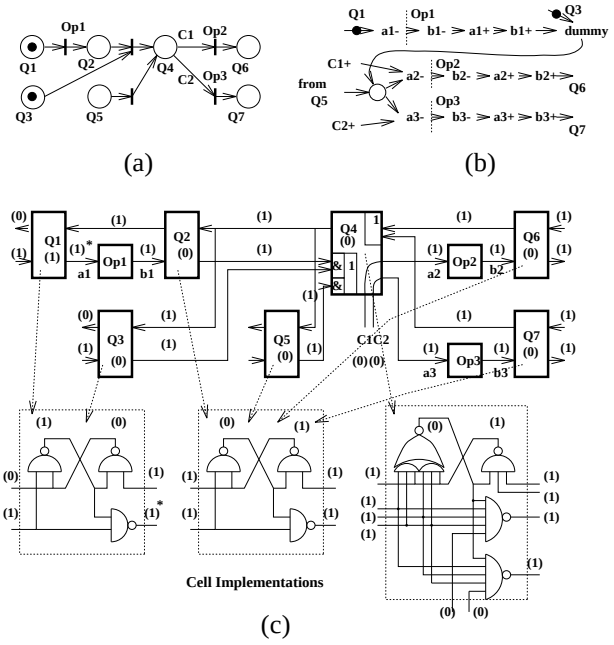

Figure 8: "Circuit compilation" of Petri net specification: fragment of net specification (a); graph describing the operation synchronisation at the signal level (b); circuit implementation based on "distributor cells" (c)

with shift towards MSB" (a_6, b_3); CIU(H,M) is "carry out either CIU(H) or CIU(M) depending on the input half-byte's code value" $(a_4, (b_4, b_6))$; CIU(S,M) is also either CIU(H)or CIU(S) $(a_6, (b_4, b_6))$;

On the basis of such a refinement we constructed a labelled Petri net model of the control executed by the GCU. A fragment of this net corresponding mainly to the transitions from the "idle" (0) state in the net of Figure 4 is shown in Figure 7.

The next design step was to translate the net-based behavioural description into the structural implementation of the GCU. In this work we resorted to the so-called syntax-driven approach to translation of nets into circuits. This was because we wanted to avoid problems with race-free encoding and hazard elimination, which would be quite complex to resolve for the control of such a size and under the accepted speed-independence requirements. At least, none of the automated tools available at the time were capable to synthesize our circuit.

The principles of the "circuit compilation" of a Petri net have been presented in a number of publications. Our approach followed the idea of a phase-distributed circuit implementation of parallel algorithms described in [10]. Here each (or almost each, since some "code optimisation" is performed at the level of control circuit macro-cells, which is an intermediate representation in our "compilation") place of the net is associated with a state-holding macro-cell. The gating logic, both AND and OR, normally realised by the Petri net transitions and places, is implemented within the macro-cells. Figure 7 illustrates the major idea of our "circuit compilation".

As a result of such a two-step compilation, "Petri net model → macro-cell network → gate-level control circuit", we obtained an implementation consisting of 220 NAND and AND-OR-NOT gates with the maximum fan-in limit of 4. In general, any syntax-driven method, including ours, would produce a circuit whose size is proportional to the size of the behavioural description (unlike STG-based synthesis methods, e.g. [9]).

5 Summary and Conclusions

In this paper we have tried to share our experience in designing an asynchronous interface based on a pipeline ring. The implementation of the protocol machine is totally speed-independent: the ring operates according to its protocol regardless of the delays in the interconnections between adaptors and all gate delays in the adaptor circuits. Speed-independence implies self-checking with respect to stuck-at faults on gate outputs. We were therefore able to detect on-line faults in the adaptors and transmission lines of the ring, and construct a special fault-tolerance mechanism, based on fault-localisation and self-recovery. The implementation of the latter is left outside the scope of the present paper (see [7]). We believe that the use of a ring instead of a bus, irrespective of its further enhancement with fault-tolerance facilities, can itself be a way to improve the realiability of interfaces, even of on-chip ones, in asynchronous systems. The performance of systems can also be improved in some cases, due to the effect of point-to-point interconnections and pipelining.

Acknowledgements

Two of our former colleagues, Vladimir Volodarsky and Yuri Tatarinov, took an active part in the design of the fault-tolerant ring channel. We thank Ken Stevens for his useful comments about our work.

References

[1] 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.

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

[3] J. Theus and D.B. Gustavson. Wired-OR on transmission lines. *IEEE Micro*, Vol.3, No.3, June 1983, pp. 51 – 55.

[4] D.M. Taub. Arbitration and control acquisition in the proposed IEEE 896 Futurebus. *IEEE Micro*, Vol.4, No.4, August 1984, pp. 28 – 41.

[5] V.I.Varshavksy,V.B. Marakhovsky, L.Ya.Rosenblyum and A.V.Yakovlev. Implementation and analysis of the TRIMOSBUS self-clocking interface. *Automatic Control and Computer Science*, Vol. 19, No. 4, pp. 80 – 87, 1985 (translated from Russian).

[6] 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).

[7] V.I.Varshavsky and V.B.Marakhovsky. Fault-tolerant self-timed system mono-channel. The Institute of Electronics, Information and Communication Engineers, Tokyo, Japan, Technical Reports CPSY94-24, FTS94-24, ICD94-24, April 1994, pp.79-85.

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

[9] 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.

[10] 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).

[11] E.Sperner. Ein Satz uber Untermengen einer endlichen Menge. *Math. Z.*, 27, 1928, pp. 544-548.

[12] V.I.Varshavksy,V.B.Marakhovsky,L.Ya.Rosenblyum, Yu.S.Tatarinov and A.V.Yakovlev. Towards fault-tolerant hardware implementation of physical layer network protocols. *Automatic Control and Computer Science*, Vol. 20, No. 6, pp. 71 – 76, 1986.

[13] H.J.Genrich. Predicate/transition nets. *Advances in Petri Nets, LNCS 256*, Springer-Verlag, 1987.

[14] 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.

Appendix

Other Protocol Functions

Here we briefly outline the message transmission and reception functions. The labelled Petri net (LPN) for the transmission procedure executed by the ring master is shown in Figure 9(a) while the global description of the message reception procedure carried out by any recipient is shown in Figure 9(b).

The module that has won the arbitration and become "master with reception" begins to transmit the message immediately after sending token $M3$. Then the master takes each byte (BT) of the message from its FIFO1, splits

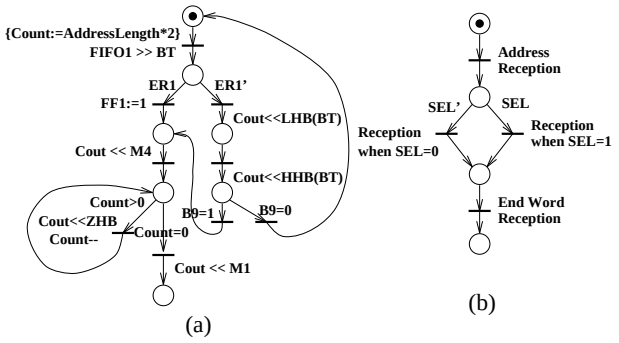

Figure 9: LPNs for message transmission (a) and a global view of reception (b)

it into two half-bytes (LHB for "low half-byte", and HHB for "high half-byte") and places them to the output channel ($Cout$) a pipeline fashion. This process continues until the last byte (marked with Bit 9 equal to 1) is encountered. Then the master produces the end token $M4$, followed by a sequence of zero half-bytes (ZHB) which are needed to "assemble" the end word of the ring channel. The length of the end word (in half-bytes) is twice the address length, measured in bytes. It is obvious that the length should be available with the master after it has been extracted from the header of the message from FIFO1 (the byte consisting of the priority and the address size). Finally, the master issues $M1$ to start the new bidding "campaign" in the ring.

The main parts of the reception procedure are refined in Figure 10. Note that the procedure of the reception of the end word is almost identical to that of the address reception (see explanation below).

The address recognition procedure in any receiver (except for the receiver section of the master) is begun with the reception (from input channel Cin) of a half-byte (HB) containing the address length, which follows token $M3$. This information (converted into the address size in half-bytes) is stored (variable $Count$) in the module. The value of the SEL variable is initialised at 0. Then, with each new address half-byte received (it is assumed that the address bits are transmitted in the ascending order of significance - the least significant one goes first), the current value of SEL is first copied in SEL_cp, and then this half-byte is put to the channel being shifted towards the higher-significance end. The MSB of the half-byte is thus stored in the SEL flag, while the previous contents of the SEL becomes the LSB of the forwarded half-byte. This continues until the most significant half-byte of the address is received. Here the last value shifted into SEL becomes the actual value of the Selected flag. If this value is 1, the module will start to copy the remainder of the message into its FIFO2 (see Figure 10(b)) beginning with the header which consists of the status byte. Alternatively, the module will play the role of a repeater (simple pipeline cell) in the ring (see Figure 10(c)).

The reception and forwarding of the end word is much

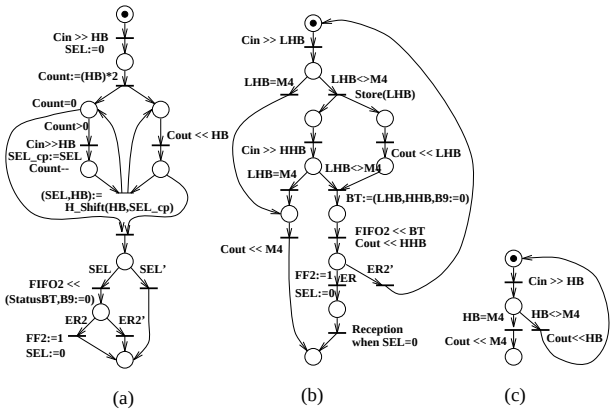

Figure 10: LPNs for address reception and recognition (a) and message reception when SEL=1 (b) and when SEL=0 (c)

similar to that of the address. The only difference is that the value of the SEL variable is initialised with the SEL flag. Using the same method of half-byte shifting-copying, we achieve the effect that each module contributes to the end word with the last value of its SEL flag (which may not be the same as the one taken from the address; the flag could have been reset due to an error in the module). Thus each bit of the end word effectively tells the message originator (recall that the message taken from FIFO1 ends in FIFO2 of the master) about the success in delivering the message to its recipient at the MAC layer. This information can be used by the network layer functions.

Note that the receiving section of the master executes the general reception protocol except that it does not copy the message half-bytes to its output channel [5].

The protocol descriptions involve explicit checking for errors in reading a byte from FIFO1 and writing a byte into FIFO2. The actual implementation of these conditions ($ER1$ and $ER2$) is based on timeouts associated with the handshakes between the FIFOs and RAC. Should an error arise, e.g., in the master reading a byte from FIFO1, the message ceases to be copied to the channel, and token $M4$ is produced instead. At the same time a special exception (signal $FF1$) is raised to the ULC. Similarly, if an error occurs in the FIFO2 link of the receiving module, its SEL flag is immediately reset to 0 and an exception signal ($FF2$) is produced to the ULC. The module thus becomes a repeater (i.e., unselected recipient).

Protocol verification using Pr/T nets

The main part of the protocol specification was shown in Figure 4. This Petri net describes the behaviour of a single protocol entity, one of the ring adaptors. To verify the protocol we need a model of the entire protocol

[5] Although in our present version the master must *always* be a selected receiver, with the message thus always being copied into its FIFO2, the latter can be easily reorganised as an option by using an explicit self-addressing mechanism.

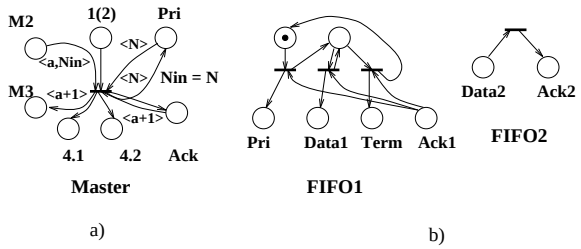

Figure 11: Pr/T net description of Master transition and FIFO modules.

layer. Note that the ring is a regular structure consisting of an "array" of identical elements. Predicate/Transition nets (Pr/T nets) [13] offer a convenient way to represent it. To obtain an Pr/T model we introduce explicit places for inputs and outputs of an adaptor. There are six places ($M1, ..., M4$, $Data$, Ack) representing inputs/outputs of an adaptor from/to its left- and right-hand side neighbour, 4 places (Pri, $Data1$, $Term$, $Ack1$) representing its interface to the $FIFO1$ and 2 places ($Data2$, $Ack2$) representing its interface to $FIFO2$. We model each FIFO as simple Pt/T nets shown in Figure 11(b). (Here and below we assume that all uninscribed arrows carry a token with the adaptor's number $\langle a \rangle$.) Each token is assigned with an adaptor number representing the fact that the token is enabling actions of that particular adaptor. Tokens in place $M2$ are assigned with a tuple consisting of the adaptor number and the priority value. Each of the net transitions in Figure 4 which consume tokens from the places representing inputs connected to the left-hand side neighbour of an adaptor are transformed to transitions of the Pr/T net which take a token from the adaptor's interface places labelled with the number of the adaptor concerned. Similarly, transitions that produce tokens into the output places of an adaptor (and hence into places of its right-hand side neighbour) are transformed into transitions producing tokens labelled with the "next in the row" adaptor. Transitions that are "local" to the adapter are transformed so that they consume and produce tokens labelled with the adapter number without changing it. An example of the Pr/t net description of the Master transition (changing an adapter's state to "master-receiver") is shown in Figure 11(a). Here the transition consumes a token labelled with the number $\langle a \rangle$ from place $M2$ which corresponds to accepting the token from the left-hand side neighbour and produces tokens labelled with $\langle a+1 \rangle$ into place $M3$ which corresponds to it producing an output for the righ-hand side adaptor. Similar, it consumes token $\langle a \rangle$ from Ack place and produces token $\langle a+1 \rangle$ in it denoting the receipt of the acknowledgement from the righ-hand side neighbour.

The behaviour of $FIFO1$ is described as follows: it issues a token labelled with its adaptor's number into the place Pri first and then generates an arbitrary finite number of tokens (to model the length of the message) into the $Data1$ place followed by a token in $Term$. Each token can

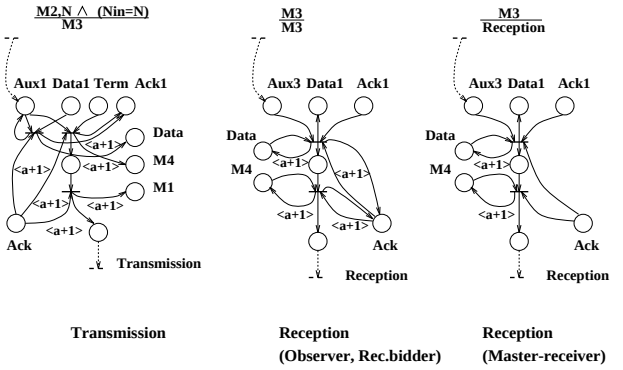

Figure 12: Pr/T description of transmission and reception.

only be generated if there exists a token labelled with the appropriate adaptor number in the *Ack1* place. *FIFO2* models a simple behaviour being able to consume a token labelled with the adaptor number arrived into *Data2* place and to generate a token labelled with the same into *Ack2* place.

We also refine the *Transmission* and *Reception* transitions from the net in Figure 4. Note that there are two possible refinements of *Reception* for the "master-receiver" state and the "observer-receiver"/"receiver-bidder" states. The corresponding Pr/T nets are shown in Figure 12. Each of the processes is connected to the adaptor's Pr/T net through auxiliary places, tokens in which are produced/consumed by appropriate transitions.

The Pr/T-net model of the ring has been verified using one of the existing tools called PROD [14]. For deadlock detection we used the "stubborn set" method implemented in PROD. This method builds a reduced reachability state space (RRSS). 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 adapter 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.

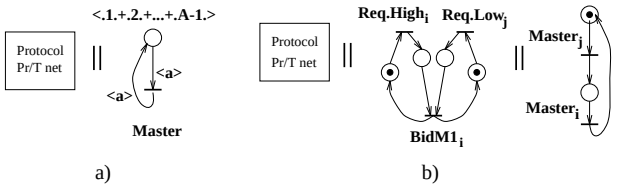

Figure 13: Verification of protocol properties.

Number adaptors	Number priorities	Reduced State Space 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.

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 13(b) (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 labeled 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 adapter 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 prior to the arrival of token $M1$ into the leftmost adaptor, the one with the higher priority will gain access to the ring first. This problem can be reduced to deadlock detection by performing concurrent composition of nets as shown in Figure 13(b). Verification shows absence of deadlocks if the proper access ordering is used and reports a deadlock otherwise. Since there can be no two adapters simultaneously accessing the ring (safety property) we conclude that the required order is maintained in our protocol.