The Rotating Bus as a Basis for Interprocess Communication in Distributed Systems

Nenad Marovac
Department of Mathematical and Computer Sciences, San Diego State University, San Diego, California 92182, USA

In the last decade a large number of different interprocess connecting structures and associated protocols were implemented in systems incorporating multiple processors. Efforts in the research of interprocess communication have been intensified in latter years in connection with the research and development of distributed systems. However, most of the developed structures were oriented towards communication between complete computers with full processing power, and involved these computers too much in the communication between processes. What we propose here is more along the lines of the classical bus structure with distributed data propagation and control. Our mechanism is based on the rotating bus concept; it ensures simplicity of interconnecting structures, communication control procedures and very small time overhead in interprocess communication within a computing system using the rotating bus features. It also allows for simple expansion of the system. The mechanism, with completely distributed control, is mainly intended for use in interprocess communication in multi-processor systems, computing systems with distributed architecture and in-house networks linking intelligent terminals.

INTRODUCTION

The distributed systems encompass data processing systems comprising a multiplicity of different types of general purpose resources.[1] These resources are physically distributed and dynamically assigned as required to processes taking place within the system. The processes, also being physically distributed, mutually interact through an interprocess communication mechanism linking the system together. The processes are coordinated by an operating system. The operating system unifies and integrates, *but does not centralize* the control of the resources or the processes which operate within the system in a cooperative autonomy.

This informal definition of distributed systems brings to light two important features of the distributed systems: first, the need for an efficient interprocess communication mechanism in such systems, and second, the suggestion that beside resources, data and control of system resources being distributed, the control of the interprocess communication mechanism should also be highly decentralized or rather completely distributed.

In the last two decades a large number of different interprocess connection structures and associated mechanisms were implemented in systems incorporating multiple processors.[2–5] References 6–9 present excellent efforts in systematic classification and evaluation of such structures. Recently, however, research in interprocess communication has been intensified in connection with the research and development of distributed systems.

In the context of this paper we are not interested in distributed systems spread over a wide geographical area. Instead, we are concerned with systems housed in one or more cabinets implementing single or multiprocessor computing systems with distributed architecture. In other words, *our objective is to replace a conventional bus structure with a communication mechanism suited for the development of novel computing systems with the distributed architecture of tomorrow, with increased throughput by enchancing computing concurrency.* We are also interested in using the communication mechanism being proposed in this paper in a system of terminal stations, i.e. intelligent terminals incorporating microprocessors. Such stations can be placed in departments and executive offices within enterprises and our communication mechanism is to be used as a basis for cooperative processing and transmission of data in textual and pictorial form from one station to another.

This paper proposes a universal mechanism between processes. The mechanism, based on the 'rotating bus' concept and its associated protocol, enables 'simultaneous' one-to-one, one-to-many, and one-to-all (broadcasting) modes of communication between processes in the system, as well as simple expansion of the system. As already stressed, the mechanism is mainly intended for use in interprocess communication in multiprocessor and distributed computing systems, as well as in-house systems of linked intelligent terminals incorporating microcomputers.

In the Appendix to the paper we describe a simple and popular procedure to link together two processors. We also illustrate how different processors can be adapted to the requirements for that communication procedure. The purpose of the discussion in the Appendix is to show that processors with very different communication properties and different levels of intelligence, ranging from full computers, through processors and DMA controllers, to passive elements like memories, can be successfully adapted for interchange of data using the basic simple procedure on which we intend to establish the principles of our communication mechanism.

In the next section we state the requirements for a general interprocess communication mechanism for distributed systems. In the section on 'The rotating bus' we present the basic design and prinicples for such a mechanism based on these objectives and the observations made in the Appendix. This section also includes a discussion on the achievable data transfer rates between any two processes in a system, and a discussion on the reliability and security of the system design. The

CCC–0010–4620/82/0025–0022 $05.00

following section briefly describes the word (byte) transfer level protocol.

The rotating bus, as we present it in those two sections, is based upon a simple bus loop structure. The final section includes some generalizations to make the rotating bus a more versatile communication mechanism. Furthermore, although the rotating bus is mainly intended for physically compact distributed systems, it can be adapted to circumstances where components are spaced over wider geographical areas by inserting serial components for transmission over larger distances.

OBJECTIVES IN INTER-PROCESS COMMUNICATION

The purpose of the examples given in the Appendix was to illustrate that in systems of interconnected asymmetrical processors (systems containing diverse computers, processors and other types of data processing devices which have different communication properties and degrees of intelligence) it is possible to adapt these devices in such a way as to have equal properties with respect to a communication procedure applied within these systems. In the remainder of the main part of this paper we will examine how to fit the simple handshaking procedure into an efficient and flexible interconnecting structure for intercommunication between computing processes in systems with distributed architectures containing processors of different natures. We will also consider the basic data transfer protocol for such a network.

Initially we must consider what our goals are in interprocess communication. The situation to which we wish to address ourselves is a system of N, possibly interacting, computing processes. These processes may perform independently or they may cooperate on a set of tasks. In general, we assume that the individual processes divide their time between computation and communication with other processes.

With this situation in mind, we outline in the following list what we feel are the most important requirements for an interprocess communication mechanism:

(i) Fully decentralized control. The control of the communication mechanism should not be localized in one process, nor should there be a device dedicated solely for controlling the communication mechanism. This is in order to avoid the bottleneck usually common to such systems. Also, a system in which the control is decentralized and in which separate control functions are located where needed offers higher reliability and larger flexibility of operation. This applies particularly to a system in which all processes are on an equal functional level of importance.

(ii) Asynchronous operation. The individual processes should not have to concern themselves with global timing considerations in data transfer within the system.

(iii) Simplicity of operation. A process, and the processor on which the process is running, should not be required to deal with any aspect of communication operations except to place messages into the system, and remove them from the system when they are addressed to the process in question. In particular, a process should not have to be engaged in routing or re-transmitting messages between two different processes.

(iv) Versatile operation. A process should be able to address as many other processes at once as desired (i.e. a single message could be addressed to one other process, several other processes, or all the other processes in the system).

(v) Reliable operation. Processes should be able to detect and recover from most errors in message transmission.

(vi) Design economy. We would like a communication mechanism, the implementation of which is not more expensive than the processors within the system.

(vii) Flexibility to expansion. Our mechanism should be capable of easy expansion in the event that we wish to include more processors in the system.

(viii) Universality. The mechanism should be sufficiently general to accommodate any kind of a computing device with given word size, from diverse computers, processors, DMA and device controllers to memory units.

It should be noted that most of these considerations apply equally well to the design of a general-purpose, long-distance computer network communication mechanism. This, however, is not the purpose of our study. We are interested in a mechanism which is suitable for a closely spaced network such as an in-house text and picture distribution system or, more importantly, network-type distributed computer architecture with intensive interaction between the processors.

THE ROTATING BUS

Communication between processes in a system may be performed either by means of shared memory or through message transfers. Formally it would appear that the shared variable method is not as suitable as the message transfer method.[10] Certainly there are difficulties to be overcome in the way of possible race conditions. In the following we will consider only the message transfer method.

In order to transport a simple handshaking procedure and utilize the observations made in the Appendix for our problem of interprocess communication in a system of N processes, we will approach the problem by considering the configuration given in Fig. 1. In this

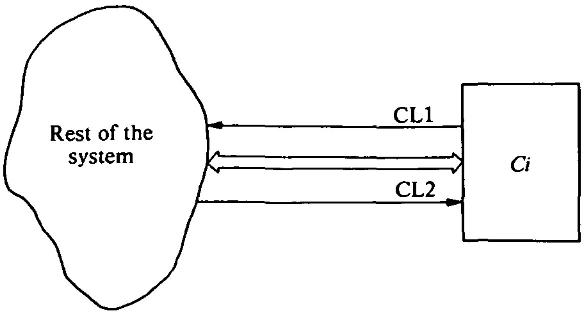

Figure 1. Handshaking between *Ci* and the rest of the system.

figure, a process *Ci* is extracted from the rest of the system and is interplaying with the *entire remaining part* of the system via the handshaking procedure. All the *Ci* process sees is an *n*-bit bi-directional data path and the two control and sense lines. CL1 and CL2. The CL2 line is used to indicate to *Ci* the presence of a message from the system or acknowledge reception of a message from *Ci*. The CL1 line indicates to the rest of the system the intention of *Ci* to send a message or acknowledge reception of a message from the rest of the system.

It now remains to determine the organization of the interconnection structure to establish such interplay and the contents of the vector data lines.

If we think of messages as data, and our system of processes as a computer, then we immediately think of a bus structure as a means for transferring messages from one process to another, just as a conventional computer uses a bus to transfer data, say, from memory to the CPU, and so forth. A bus structure also conforms to the principles discussed in relation to Fig. 1. Unfortunately, the conventional bus has major disadvantages for a system of multi-processors. In a conventional computer when one device places information on the bus no other device may use the bus at the same time. Control of the bus is passed between different units, i.e. CPU, DMA controller and so forth, but at any instant of time, only one unit is firmly in control of the bus. The concept of a unit being in complete control of the bus at any instant of time is not in keeping with the principle of giving each processor equal power at the same time in a system of processes of equal importance. This comment applies also to the case of a centralized bus controller which governs time multiplexing of the bus, granting use of the bus to different units either at their explicit request or in a round robin manner.

On the other hand, there are significant advantages to a bus in interprocessor communication. Bus structure enables parallel transfer of the entire data unit, as a word or byte. The addressing of the receiving device is simple, no need for complex routing procedures. Also, the bus communication mechanism is just as efficient for data transfers involving devices incorporating intelligence, like CPUs, DMA controllers and so forth, as it is for passive devices like memories, I/O ports, etc.

Bearing in mind the aforementioned comments on the use of bus structures in interprocessor communication, and the objectives outlined in the section on objectives, we decided to adopt the following design, which is a modification of the conventional bus structure. Instead of a 'static' bus we decided to use a 'rotating' bus, namely a circular vector data path in which data being transferred between processors rotates around the bus from one process to another in a manner similar to the rotating magnetic field in polyphase AC asynchronous motors. Basically, what we are proposing is a circular structure resembling a conventional bus closed into a loop and then partitioned into a large number of segments, as shown in Fig. 2(a). The length of these segments, referred to as *slots*, is one bit and their width is equal to the bit-width of the bus. Each slot contains, at any one time, a single information unit called a *frame*. A frame contains an actual bus data message (word or byte) plus the control information, i.e. addressing and sequencing information in certain implementations. Each bit of information in a frame, data and control, is on a separate parallel line, so

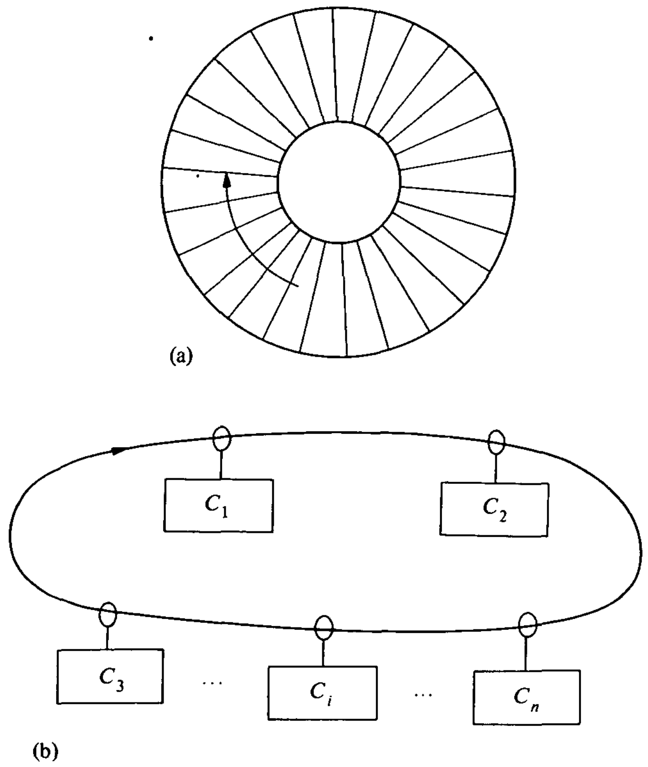

Figure 2. Illustration of a rotating bus: (a) bus segments; (b) computers on a rotating bus.

that the entire data frame rotates together as a single unit from one slot to another. There are many data frames rotating around the bus *at the same time*. The rate of rotation of data frames around the bus may be considerably faster than the processing rate of the processes connected to the bus. In our experimental rotating bus, the rotating rate of data is five million data frames per second. We may then, for all practical purposes, consider the data transfer between two processes to be instantaneous, and that all processes on the bus are able to communicate simultaneously with each other in a bus-like manner.

Another difference between the conventional bus and the rotating bus, besides rotation of data, is the addition of another address field to the bus. In the conventional bus, either the sender or receiver of data is implicit, being the unit controlling the data bus at that time. The other participant, the receiver or sender, is specified explicitly by having its address on the address part of the bus (address bus). In the rotating bus there are many data frames on the bus at any one time. As these data frames have different senders and receivers, and as all processes are potential participants in data transfers due to the fact that no single process is in control of the bus (in accordance with objective (i)), each data frame on the rotating bus must contain the address of both the sender and receiver of that data unit.

The rotating bus as an interprocess communication mechanism incorporates both the simplicity of a loop network topology,[11–14] and the simplicity of a bus mechanism. It is well known that networks based on loop topology satisfy objectives (iii), (vii), and the bus mechanism together with the handshaking procedure can be adapted to satisfy objectives (ii), (iii), (iv), (vi), (vii) and (viii). This will be further elaborated upon in the remainder of this section. Furthermore, as will be apparent from the next subsection, control of the rotating bus is completely distributed. This satisfies objective (i), which is of paramount importance in distributed processing systems.

Devices, like computers, processors, memory units and so forth, are connected to the rotating bus via Rotating Bus Interfaces (RBI), as shown in Fig. 2(b). The function of an RBI is to monitor the bus for messages for its associated process, or for an empty slot (frame) if the process wishes to send a message. When an RBI detects a message designated to its associated process it: (i) informs its process that a message has arrived; (ii) releases the message to the process if it is not read automatically; (iii) allows the process to place a message on the bus if desired; and (iv) continues to monitor the bus for the next message directed to its process.

When its process wishes to place a message on the bus, the RBI waits for an empty frame to appear, and then proceeds in similar fashion.

The rotating bus interface

We present here a simplified version of the RBI in such a manner as to bring out its main functional characteristics.

The structure of the RBI is depicted in Fig. 3. The central part of an RBI is an m-bit parallel-in-parallel-out (master/slave flip flop) register, known as an RBI register, slot register, or simply a slot. The register becomes a part of the bus. The bus is a uni-directional m-bit data line. The bus enters the RBI on one side coming from the previous station on the bus, and exists on the other side of the register towards the next station. Besides being incorporated into the bus the RBI register is connected to its associated process via m bi-directional lines. Connection to the process is also made from logic circuits in the RBI to interrupt the process via the CL2 line if the RBI register contains information directed to the process.

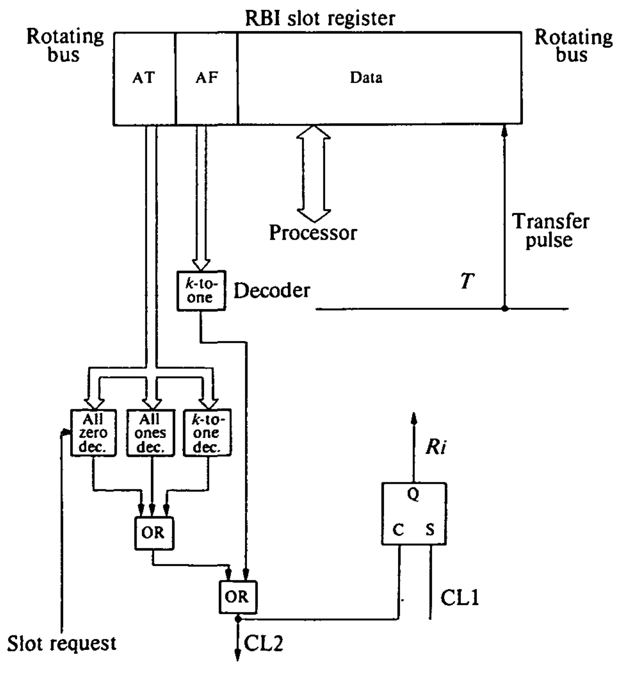

Figure 3. Structure of a RBI.

In its basic design, the m bits of an RBI are composed of d bits of data, an address-to field AT and an address-from field AF. The number of bits in the address fields is determined by the number of stations to be placed on the loop. Since two address values are reserved for special conditions, n address bits can address 2^{n-2} stations. We use an address-to value of zero to denote an empty frame, and a value where each bit is set to '1' to indicate that the data item (frame) is being broadcast to all stations on the bus.

There are three decoders in an RBI attached to the address-to field, and one attached to the address-from field. The decoders on the address-to field detect whether the address is that of the process, or whether it is all ones or zeros. The first two cases indicate that a message has arrived for the process, which is then interrupted to receive the message. The last case signals the arrival of an empty data frame and this generates an interrupt if the process has activated its slot request line. The decoder on the address-from field generates an interrupt to the process if the address is that of the process. It can signal either the return of a broadcast which has traversed the bus and must now be removed, or it can signal the fact that a message sent to another station on the bus was not received. If this occurs three times with the same message the sending station signals a warning that the station being addressed is not responding.

In addition to the RBIs the rotating bus includes a bus control line T. This line is controlled by ready flags from all processors in the bus, and a bus clock. The transmit control line T carries clocking pulses governing the rotation of messages within the system, i.e. transmission of message data from one RBI to another around the rotating bus.

Comparison of the rotating bus with other loop structures

Loop topologies have been used by others in computer networking. However, previous work was on the development of serial ring network structures.[11-16] There are a number of advantages, as we see them, in using the rotating bus in the development of distributed architectures:

(1) Parallel data access; parallel structures allow for wider bandwidth and higher data transfer rates through the system.

(2) Simpler addressing; the rotating bus allows for simple bus addressing and address processing strategies. There is no need to accept part of a message in order to decide whether to accept or forward the message, which generally adds to communication delays.

(3) The inherent problem of loop structures, viz. lost token detection and token regeneration[17,18] is avoided in the use of the rotating bus.

(4) Two register RBIs (Fig. 4) provide for better reliability of the rotating bus than CRC block error or parity checking techniques used in serial communication rings.

(5) Superior security and privacy arrangements.

(6) Complexity of interfaces; RBIs are simpler than interfaces in conventional serial loop networks, which, in general, require a much higher degree of intelligence. The latter usually incorporates microprocessors, while

RBIs include only passive devices. Furthermore, the complexity of interfaces in serial loop structures implies further delays in their different stages which means further decreases in bandwidth as compared to the rotating bus.

(7) Protocol simplicity; The bus type protocol of the rotating bus is simpler than message protocols in serial loop structures; as the bus becomes narrower, the control function becomes more complex.[6]

The main disadvantage of the rotating bus is the multiplicity of lines. However, we feel that the expense of the additional lines is offset by the greater cost of interfaces in serial loop structures and the other advantages of the rotating bus as listed above.

Functionally, the rotating bus is a system of time slots. However, if frames were grouped into variable length packages (variable number of frames per packet) and RBIs were adequately modified, the rotating bus could be built to resemble either pass control[4] or DDCLN rings;[14] in fact for simulation purposes our first rotating bus was built as an 8-frame-packet system.

However, for application in distributed architectures it is more efficient to use a rotating bus as a system of time slots. In distributed architecture computing systems single frame messages (i.e. one word or one byte) are more frequent and more natural than multiple frame packets. Moreover, even in the case of multi-frame messages, occurring for example in transferring and processing of arrays, transfer of single frame packets is more efficient. The rotating bus clock is considerably faster than the clocks of individual processors 'hanging' on the bus. Therefore, after receiving a frame from or placing a frame onto the rotating bus, a relatively long time interval will expire before the processor is ready to accept or deliver another frame (word or byte). This means that a number of frames could pass by the processor in this 'idle' time. These idle or in-between frames can just as well be used by other processors on the bus.

Furthermore, the length of buffer queues (see section on rotating bus in uniprocessor environment) is then dependent only on the local environment, i.e. disparity in speeds of the rotating bus and the local processor (and to a lesser degree on the speed of processors with which the local processor is frequently communicating) and not on the average global message length as in DDCLN rings.[14]

In the design and construction of our bus and its interfaces, it became apparent that a further possibility of reducing the propagation delay and speeding-up operations existed without having to go for faster and more expensive processors on the bus. This time saving was related to the time delay required for the decoding of frame addresses. As mentioned above, an RBI register was implemented by using master-slave flip-flops. We decided to use edge triggered flip-flops; a frame enters the first stage of a flip-flop on a positive edge and the second stage on a negative edge of a clock pulse. The valley between the two pulses was to be used by the RBI to perform address decoding and unloading of the frame by the associated processor if the addresses indicated such action. If address decoding takes $p1$ time and frame unloading/loading $p2$ time the required length of the clock pulse valley interval is $L2 = p1 + p2$. However, when a frame is in the second stage of the flip-flop of station i on the bus, during the clock period Tk, it is also available to the decoding circuitry in the station $i + 1$. The decoding in the station $i + 1$ can take place through the $L2$ of Tk clock pulse and $L1$ of pulse $Tk + 1$, where $L1$ is the high level part of a clock period (i.e. $L1 + L2 = T$). This reduces the required length of the clock pulse valley intervals, as during this time only loading/unloading of frames takes place, i.e. it is enough that $L2 = p2$, and $L1 \geq p1$. The only modification required to RBIs as shown in Fig. 8, is that the address portion of inputs to RBIs registers are routed to decoding circuitry and that output from the OR gate resulting in the signal CL2 is activated with the negative edge of each clock pulse. This is in order to ensure RBI-processor synchronization and to provide for the fact that station i may change the contents of the frame towards the end of the clock interval Tk. It is apparent that this implementational feature is not available to serial structures.

Speed, expansion and load consideration

The bus clock works at very high frequencies since the speed of the bus is influenced only by the speed of logic used in the control gates and flip-flops incorporated in the RBI registers. Using fast TTL logic, clock rates between 10 to 50 MHz are reasonable, which results in a bus with large data transfer rates.

A further advantage to a rotating type of bus is that it is readily expandable. All that is needed to add a processor into the bus is to plug-in its RBI at the desired place in the bus. Additional independent RBI registers can also be added to the bus to provide extra message slots—this is particularly useful if one expects heavy traffic in the bus.

A study of the loading characteristics of the bus is currently in progress, in particular, to determine the availability of free message slots. At first glance it would appear that a few communicating processes could tie up the available message slots on the bus and lock out all the other processes. Study by Hayes and Sherman has shown that this is not the case.[19] Although they considered Pierce communication ring networks,[12] the results can be directly interpreted and extended to the rotating bus. However, let us further examine the problem through a discussion of some practical alternative solutions.

For the moment we will assume that the number of slots available on the bus is equal to the number of RBI's, which is equal to the number of processors attached to the bus, let us say N. Now, suppose that in the bus protocol, we limit the availability of slots to one per communicating pair, i.e. if process A sends a message to B, which in turn acknowledges or replies to this message, then only one slot is allocated for this exchange (that is, B replies in the same slot that A used). With this restriction, if one process only talks to one other process at a time, then the total number of slots required is $N/2$. If we allow one process to talk to M other processes at the same time then we require $NM/2$ slots. The extreme case is where $M = N - 1$, which is the case where every process talks to every other process simultaneously.

Now let us suppose that we limit M to be 2, that is we limit each process to communicating with only two other processes at a time, then we require precisely N slots which is just the number we assumed we had to begin with. Now if we alter the original assumption and allow the average M to be 2 we conclude that the number of slots on the bus would provide for smooth communication between all processors on the bus. We should note here that both the above restriction of limiting each communicating pair to a single slot and the assumption for the average M to be 2, are very practical in general and, even more so, in the systems for which use of the rotating bus is suggested in this paper.

A second approach would be to incorporate the full number of $N(N-1)/2$ slots into the bus. We could augment the N RBIs with $(N(N-1)/2 - N)$ slot registers. Simple slot registers can be made very inexpensively (at the cost of a few dollars) so this is certainly a feasible alternative.

Another approach to slot management is as follows: we make no restriction on the number of slots that a given process uses, or on the number of other processes with which it communicates at any time—however, we insist that when a message is received by a process then the process immediately clears the slot, thereby releasing it (converting it into an empty slot). In order to reply or acknowledge the process must wait for the next free slot to arrive. This ensures that at least one free slot propagates through the system.

Clearly, there are a number of promising approaches for efficient bus management. Investigation of these approaches will be reported in a future paper.

Reliability and security

Like all loop type structures the potential weakness of the rotating bus is its reliability. This question for loop topologies is considered in references.[13,15,16]

As a rotating bus structure would be dedicated mainly to multi-processor distributed systems and in-house message distribution networks the problem of line failure is not a serious one. The pertinent difficulty is failure of processors and RBI's on the bus.

The bus functions, to a large degree, independently of the processors. Consequently, it is simple to ensure that its operation is not directly affected by the failure or shutting down of the processors. This is achieved by automatically setting CL1 line high when either the processor is to be shut down or the processor does not react to the signal on CL2 line after a certain time interval. However, if a processor fails, bus performance may be degraded due to a buildup of messages addressed to that processor. In such circumstances these messages eventually return to their senders. When the same message returns to the sender for the third time, the sender aborts the communication with the failed processor and issues a warning message to the system operator.

Very little additional circuitry is required to protect the system against failure of RBI's. This is realized with a slight increase in cost by incorporating into the RBI's a duplicate slot register. The output of each of the RBI slot registers is XORed together bit by bit. Each bit of the result is then ORed together producing a warning signal. This signal will be true if, and only if, the output from the two RBI registers is not identical. This is shown in Fig. 4 where RBI-Ai and RBI-Bi denote corresponding bit positions in the two RBI slot registers within an RBI. One should note here that only one RBI slot register (RBI-A in Fig. 4) is actually used for the bus communi-

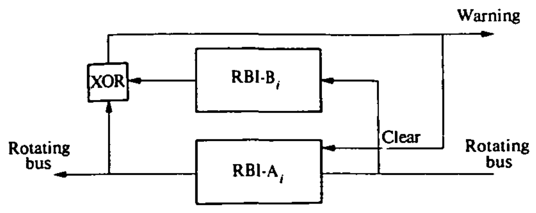

Figure 4. Modified RBI.

cation. The other is used solely for verification purposes. Although a fault in the RBI has been indicated, the potentially incorrect information could leave that RBI to rotate around the bus. This situation would exist until the RBI is replaced. In order to avoid propagation of these potentially corrupted messages, the warning signal is also used to clear the RBI slot register participating in the bus. This results in replacing the potentially corrupted message with zeros, which is interpreted as an empty slot. This will prevent reception and acknowledgement of incorrect messages. As the messages sent out by the processors will not be acknowledged communications in the system will gradually cease.

To ensure security and privacy of information the following procedure can be used. Before an RBI is incorporated into the bus, its decoders are set to the desired addresses. The RBI is then closed and sealed to prevent tampering with decoders in order to illegally extract information rotating around the bus. As a result of this, an RBI can alert its associated processor only when a data frame destined for, or originating from, that processor is in the RBIs slot register, or when a required empty slot is available, but not under any other circumstances. However, the danger still exists that a processor could unload all information passing by on the bus, even when not being interrupted by the CL2 line of its RBI. This can be prevented simply by incorporating latching buffers in the RBI which are controlled by its CL2 line. This will disable flow of any information between the RBI and the corresponding processor unless the CL2 line is high, and will consequently guarantee security of information on the bus, short of taping the actual cables between the RBIs.

COMMUNICATION PROTOCOL

In this section we discuss the basic objectives of protocols for the rotating bus. In computing systems with distributed architectures interprocess messages are in the form of individual basic data units (words or bytes). Under these circumstances, a conventional bus protocol can be directly transported to the rotating bus as the *frame protocol*, and the processes embed within themselves all the required higher level protocols.

However, if we consider distributed systems in general, interprocess messages comprise more than one basic data unit. Since, in the rotating bus, only one basic data unit of information is passed in each frame, a message consisting of several basic data units would have to be built up one unit at a time. This requires a higher level protocol, i.e. *message protocol*, which subsumes the basic data unit (frame) protocol. At a still higher level, we may wish to consider *interprocess and application protocols.*

These would, in turn, be implemented upon the message level protocol.

The message and higher level protocols can be made independent of the device-processors on the bus. This is due to the adaptability of the devices as discussed in the Appendix, and the fact that the protocols can be implemented using the frame level protocol primitives and a set of programming elements, which model different level virtual processors on the rotating bus.

The actual implementation of the basic data unit 'frame' protocol for a specific device depends on the following communication properties for the device: (i) is it serial or parallel; (ii) are there control lines which may be used as CL1 and CL2 lines (see the Appendix); and (iii) can the CL2 line interrupt the processor, or must a polling procedure be used.

The frame level protocol for general distributed systems, and the message distribution networks of intelligent terminals, is somewhat elaborate because of our wish to include a broadcast mode of communication. The frame protocol recognizes five different types of frames: (i) 'personal' frame; (ii) broadcast frame; (iii) personal broadcast frame; (iv) personal frame acknowledgement; (v) broadcast frame acknowledgement.

A personal frame is sent from one process to another, and a personal frame acknowledgement is sent in response to the correct reception of a personal frame.

A broadcast frame message is one sent from one process to every other process on the bus, and correct reception of such a frame message is given by a broadcast frame message acknowledgement. A broadcast frame sent out by a process is removed from the bus when it returns to its sender. Amongst its other activities the sender then waits for acknowledgements of the broadcast frame to arrive. If, after a reasonable interval of time has passed, one or more stations on the bus have failed to acknowledge reception of the broadcast frame, then the sending station will send out a personal broadcast frame to the stations which have failed to acknowledge receipt.

For this more elaborate version of the rotating bus frame level protocol, another three one-bit fields are added to each slot, i.e. width of the bus, as shown in Fig. 5(a). As mentioned previously, broadcast frames are indicated by each bit in the address-to (AT) field having value '1'. A personal broadcast frame is indicated by a personal address in the AT field and the bit B set. The field MP, meaning Message-Present, indicates that the data field contains a message and AP standing for Acknowledge-Present indicates that the frame is also used to acknowledge the last frame in the opposite direction, i.e. sent by the process with address AT to the process with address as in the field AF.

In other words, a frame may contain a message only, an acknowledgement only, or both an acknowledgement and message frame. When both AP and B bits are set, the frame acknowledges receipt of a broadcast frame from the process AT, besides containing a possible frame message.

If we wish to ensure, to the greatest possible extent, correct operation of the protocol, synchronization of processes, and data flow control, the fields MP and AP are expanded into sequencing fields S and R of HDLC protocols.[20,21]

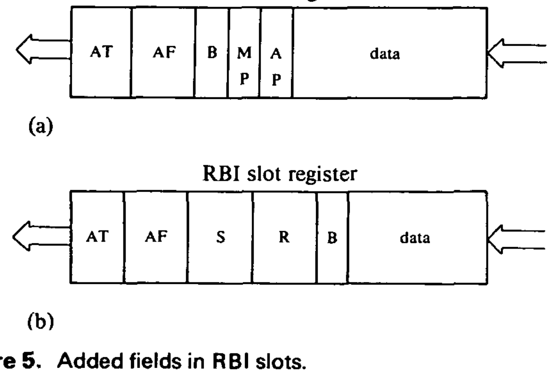

Figure 5. Added fields in RBI slots.

GENERALIZATION OF ROTATING BUS AND FUTURE WORK

Although the rotating bus was developed mainly for multi-processor computer systems with distributed architectures and in-house networks of computers and microcomputer-based intelligent terminals, it can be adapted to include remote processes as well. The required modifications are illustrated in Fig. 6. This shows the replacement of parts of the bus by serial communication facilities, which is achieved by using simple parallel to serial and serial to parallel adapters. They can be in the form of parallel-in-serial-out and serial-in-parallel-out shift registers. One should note here that the processes are still being incorporated into the system in the parallel sections only, i.e. in the rotating bus sections.

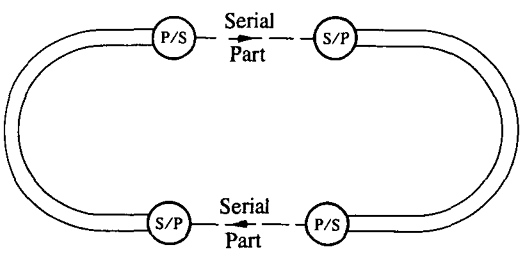

Figure 6. Rotating bus with serial components. P/S: parallel-to-serial adapter; S/P: serial-to-parallel adapter.

In order to ensure the desired (rotating bus) rates of information within the system, transfer rates along the serial parts of the structure, being bit rates, must be proportionally higher than the rates along the parallel parts.

Multi-path rotating bus

Addition of new paths to the rotating bus can be very simply implemented by the use of branching-out semaphores. This is illustrated in Fig. 7(a). This shows three paths in the bus, paths 0, 1 and 2. Processes are separated into these paths in accordance with the frequency of communication between them. Suppose that processes in path 0, have to communicate frequently with processes in paths 1 and 2, but processes in paths 1 and 2 infrequently communicate with each other. Addresses of processes in the same path are chosen so that they all

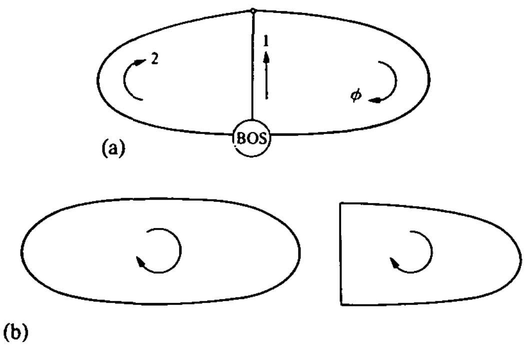

Figure 7. Multi-path rotating bus. BOS: branching-out-semaphore.

have the same values in one or more address bit positions. Then routing of messages from path 0 to paths 1 and 2 can be achieved by using branching-out semaphores. These semaphores examine the address bits in the AT (address-to) field and, in accordance with their value, let the clocking pulses on the *T* line pass along the appropriate path and, at the same time, block the passage of the clocking pulses to other paths. This ensures message transfer to the first RBI on that path and inhibits transfer to RBI's on other paths of the branch. At converging points, clocking pulses from the path which received clocking pulses at the branching-out semaphore pass the converging point into the common path (path 0) and enable transmission of message data from the last RBI on the transmitting path to the first RBI on the common path. Behaviour of the entire bus is as if it was decomposed into two separate circular paths, as shown in Fig. 7(b), only one circular path being active at a time. If the bus contains more than one common path, branching-in semaphores are used at the converging point and their mechanism is similar to that used for branching-out semaphores.

Rotating bus in uniprocessor environment

We are pursuing further research into the use of the rotating bus as a means of communication within a uniprocessor, i.e. a conventional computer. We can consider a uniprocessor computer also as a system of processors, some of which are active like CPUs, DMA and device controllers, and some of which are passive like memories, memory-like addressed I/O devices and so forth. All the remarks of the section titled 'The rotating bus' concerning the advantages and disadvantages of the conventional bus structure for inter-process communication still apply. Using the rotating bus, the data transfer within the computer can be interleaved and pipelined to a very high degree and instruction fetching, decoding and execution overlapped. From the CPU's point of view the rotating bus 'looks' like an infinite sequence of information filled temporary registers or a cache memory of infinite size. The rotating bus further increases throughput by allowing a number of processes to proceed concurrently, whereas in computers with a classical bus structure they are sequential. For example, the operation of CPUs, DMA controllers and so forth can be overlapped to a high degree. Overall, the concurrency and parallelism can be considerably enhanced by application of the rotating bus in such systems. However, the considerations and principles of the use of a rotating bus in such an environment is outside the scope of this paper and will be reported fully elsewhere.[22]

Flow control

One of the most frequent questions put forward at presentations of the rotating bus principles relates to the flow control in the system. If a processing unit is much slower than others in the system, it uses the CL1 – R*i* line to stretch the clock cycle to obtain enough time to unload a frame data destined for it. To avoid this necessity of the clock stretching, which slows the entire system, a queue of buffers is inserted between the RBI for that unit and the unit itself, Fig. 8. If a sequence of

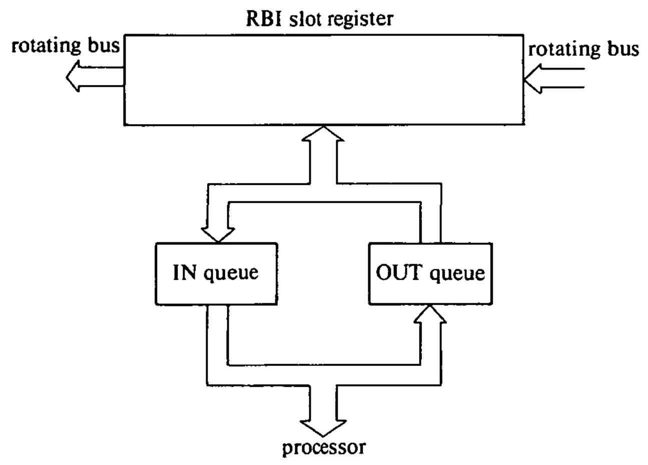

Figure 8. Non-stop rotating bus interface.

frames for a device is longer than the length of its queue (which is determined by statistical analysis) the device can still use the clock stretching technique to decrease its queue length. Even such rare occurrences of clock stretching of the bus can be completely avoided by use of an appropriate flow control strategy in the higher level point-to-point protocol.

CONCLUSION

An interprocess communication mechanism is presented, based upon a 'rotating bus'. The overall objective for such a mechanism is a simple, cost-effective, and flexible means for fast data transfer, particularly suitable for multi-processor systems.

We have implemented a system based on the concept of the rotating bus. It has a clocking speed of 5MHz and it uses clock stretching without buffering queues. For testing purposes we used SWTC6800 computers, based on Motorola 6800 processors, which were connected to RBIs on the bus via two ports in their 6820 PIA interfaces. The design, implementational and testing observations were reported in references.[23, 24]

Acknowledgements

I wish to thank Jack Krivanek, David Diamond, Brian Smith, and Selim Akl for helpful comments and assistance in preparation of this paper.

The research reported in this paper was partially supported by the National Research Council of Canada.

REFERENCES

1. P. H. Enslow Jr, What is a distributed processing system. *IEEE Transactions Computer* (January 1978).
2. G. Mazare, A few examples of how to use a symmetrical multi-micro-processor. Preliminary document for seminar on the design of distributed systems, NICE, IRIA (1978).
3. R. J. Swan *et al.*, Cm* A modular multi-micro-processor, AFIPS Conference, AFIPS Montvale, New Jersey (1977).
4. R. J. Farber *et al.*, *The system architecture of distributed computer system—the communication systems.* Symposium on computer communication networks and teletraffic (April 1972).
5. A. D. Jensen, The Honeywell experimental distributed processor—an overview. *IEEE Transactions Computer* (January 1978).
6. P. H. Enslow Jr, *Multiprocessors and Parallel Processing*, J. Wiley Interscience, New York (1974).
7. K. J. Thurber *et al.*, A systematic approach to the design of digital bussing structures. *FJCC* (1972).
8. G. A. Anderson *et al.*, Computer interconnection structures: taxonomy, characteristics and examples. *Computer Surveys* (December 1975).
9. E. D. Jensen *et al.*, A review of systematic methods in distributed processor interconnection. *IEEE Conference in Communication*, Philadelphia (14 June 1976).
10. P. Zave, On the formal definition of processes, in *Proceedings of the 1976 International Conference on Parallel processing*, ed. by P. H. Enslow IEEE, London (1976).
11. W. D. Farmer and E. E. Newhall, An experimental distributed switching system to handle bursty computer traffic, *Proceedings of ACM Symposium on Problems in the Optimization of Data Communication Systems* (1969).
12. J. R. Pierce, Network for block switching of data. *The Bell System Technical Journal* **51** (No. 6), (July–August 1972).
13. E. R. Hafner, Digital communication loops—a survey, *Proceedings of the 1974 International Zurich Seminar on Digital Communications* (1974).
14. C. C. Reames and M. T. Liu, A loop network for simultaneous transmission of variable-length messages, *Proceedings of the Second Annual Symposium on computer architecture*, pp. 7–12 (January 1975).
15. P. Zafiropulo, Reliability—a key element in loop systems, *Proceedings of the 1974 International Zurich Seminar on Digital Communications* (1974).
16. E. Hafner and Z. Nenadal, Enhancing the availability of a loop system by meshing, *Proceedings of the 1976 International Zurich Seminar on Digital Communications* (1976).
17. G. Le Lann, Distributed systems—toward a formal approach, *Proceedings of the IFIP Congress*, Toronto (August 1977).
18. G. Le Lann, Algorithms for distributed data-sharing systems which use tickets, *Proceedings of the 3rd Berkeley Workshop on Distributed Data Management and Computer Networks* (August 1978).
19. J. F. Hayes and D. N. Sherman, Traffic analysis of a ring switched data transmission system. *The Bell system technical journal* **50** (No. 9), (November 1971).
20. N. V. Stenning, A data transfer protocol. *Computer Networks* **1** (No. 2), (1976).
21. G. V. Bochman, Finite state description of communication protocols, *Proceedings of the Computer Networks Protocols Symposium*, Liege, F3–1 (1978).
22. N. Marovac, *The rotating bus as a basis for novel computer architectures*, in preparation.
23. N. Marovac and B. Smith, The rotating bus as a packet switching medium, Annual Canadian Computer Conference, Quebec, CIPS (1979).
24. B. Smith, A packet rotating bus, MSc. report, McGill University, School of Computer Science (1979).

Received May 1980

APPENDIX

An interprocess communication procedure

There is a variety of interconnecting structures used to link computers to other computers or peripherals. One of the simplest methods is to interconnect them via bi-directional parallel ports together with two associated control lines, using the 'handshaking' procedure to control the communication interplay. The corresponding interconnection structure is shown in Fig. A1. The vector path between the two computers represents an n-bit bi-directional line between a single n-bit bi-directional port in each computer, and CL1 and CL2 represent associated one-directional control signal lines used to transmit and sense Ready-to-Receive (RR), Ready (R), Data Ready (DR) and Acknowledge (ACK) control signals. The flow of processes in the two computers, as well as the flow of data and control signals between the computers when computer 1 (C1) is to transmit data to computer 2 (C2) is shown in Fig. A2.

Besides using the handshaking procedure in communication between computers, the procedure is also used in communication between asynchronous processors in computers themselves, as well as between CPU and memory. This is shown in Fig. A3.

The handshaking method presented with the aid of Figs A1 and A2 can be directly implemented in linking two MOTOROLA 6800 microcomputers via PIA 6820 devices.

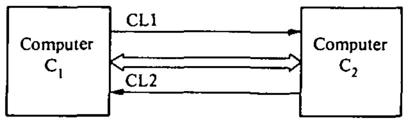

Figure A.1. Linking two computers for handshaking.

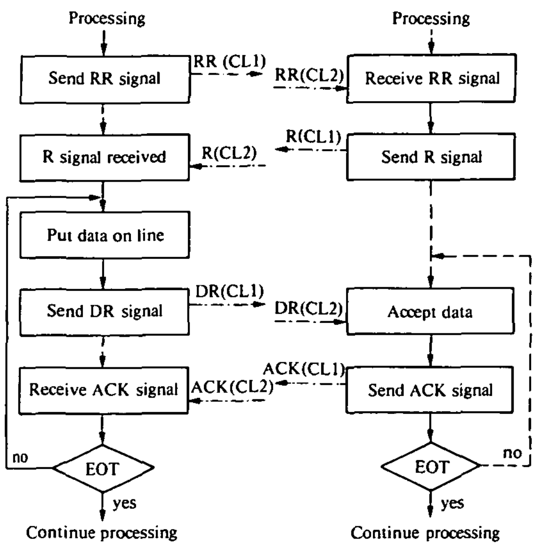

Figure A.2. Interplay between two computers in handshaking.

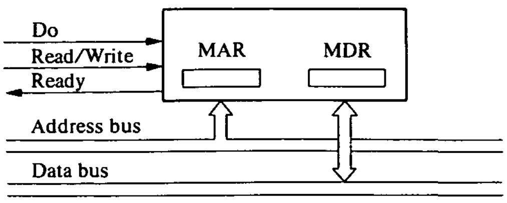

Figure A.3. CPU to memory communication.

Linking computers with no control lines

Some computers have parallel interface devices with bi-directional parallel ports without associated control signal lines. However, the same mechanism can still be applied with two simple modifications. One port in such devices is used in the same manner, i.e. as the bi-directional data port—while one bit of another port is used in output mode to simulate the CL1 control signal line, and another bit is used in input mode to simulate the CL2 sense. If the CL2 sense bit cannot interrupt the computer then the latter has to examine (poll) this bit periodically to establish reception of RR, R, DR and ACK control signals. An example of such a communication procedure arises in linking MOS Technology KIM-1 microcomputers via 6530 parallel interface devices.

Processors with serial data ports only

Although it is always theoretically possible to incorporate parallel interface devices such as the MOTOROLA PIA 6820 into the memory space of any microcomputer, it is often necessary to link a processor directly. Sometimes, such processors have only serial data ports. This is the case when linking a NATIONAL SC/MP microprocessor via its serial I/O ports. However, it is still possible to apply our communication mechanism by adapting the processor with some simple additional circuitry. The required modifications are shown in Fig. A4.

In Fig. A4, MM74C164 is a serial-in-parallel-out 8 bit shift register, and MM74C165 is a parallel-in-serial-out

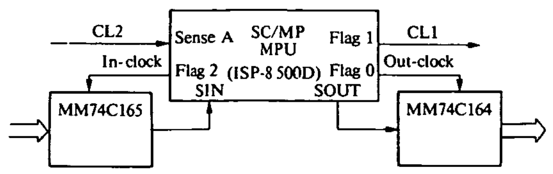

Figure A.4. SC MP in handshaking.

8 bit shift register. Flag 0 and Flag 2 signals from the MPU are used to clock the shifting of output and input bits within the output and input shift registers respectively, and Flag 1 simulates the CL1 control signal. Sense A input line on the MPU simulates the CL2 control sense. SIN and SOUT denote serial input and output ports, respectively. At the reception of data into the input shift register indicated by the control signal on sense A line, SC/MP extracts all 8 bits, bit by bit. When sending data, the SC/MP MPU outputs 8 bits, bit by bit, into the output shift register. After outputting the last (8th) bit, the MPU sets FLAG 1 simulating the CL1 control line.

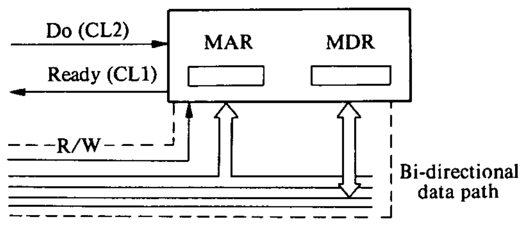

Figure A.5. Modification of CPU to memory interconnections.

Passive devices

Interconnections between memories and CPUs, as shown in Fig. A3, can also be rearranged to conform to the structure in Fig. A1. The modification is shown in Fig. A.5 where address and data buses as well as the R/W line are combined together into the bi-directional *n*-bit parallel data path.